

VMware Tanzu Service Manager v1.0

VMware Tanzu Service Manager 1.0

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

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

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

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

Contents

Tanzu Service Manager	7
Overview	7
Key Features	7
Product Snapshot	7
Requirements	8
Limitations	8
Feedback	8
Release Notes	9
v1.0.11	9
Features	9
Known Issue	9
v1.0.1	10
Features	10
Known Issue	10
Architecture and Concepts	11
Overview	11
TSMGR Components	11
TSMGR Daemon	12
TSMGR Broker	12
TSMGR Resources	12
S3-Compatible Bucket	12
Kubernetes Clusters	12
TSMGR Life Cycle	13
Create	13
Cluster and Namespace Scopes	13
Bind	14
Update	14
Delete	14
Overview of Platform Operator Tasks	15
Task List for Getting Started with TSMGR	15
Next Steps	15

Installing and Configuring Tanzu Service Manager	17
Overview	17
Prerequisites	17
Get TSMGR Resources From VMware Tanzu Network	18
Install the TSMGR CLI	18
Replicate TSMGR Images	19
(Recommended) Configure Service Mesh	20
Configure the TSMGR Helm Chart	20
Install the TSMGR Helm Chart	24
(Recommended) Configure Security	25
(Optional) Install Prometheus	25
Next Steps	25
Installing and Configuring Tanzu Service Manager in Development Mode	27
Overview	27
Prerequisites	27
Install Using the install-dev.sh Script	28
Install Manually	29
Get TSMGR Resources from VMware Tanzu Network	30
Install the TSMGR CLI	30
Configure the TSMGR Helm Chart	30
Install the TSMGR Helm Chart	31
Configure TSMGR for Cloud Foundry	31
Upgrading Tanzu Service Manager	33
Overview	33
Upgrade From v0.11 to v1.0 or Later	33
Upgrade From v1.0 or Later	33
Uninstall TSMGR v0.11	34
Configuring External Storage	36
Overview	36
Configure Non-S3 Storage	36
Managing Kubernetes Clusters for Tanzu Service Manager	37
Overview	37
Create a Cluster Credentials File	37
Kubernetes Cluster Registration	39
Register a Kubernetes Cluster	39
Set a Default Kubernetes Cluster	39

List Registered Kubernetes Clusters	40
Deregister a Kubernetes Cluster	40
Update Kubernetes Clusters	40
Using Services with Tanzu Service Manager	41
Overview	41
Prerequisites	41
Configure the TSMGR CLI	41
Add a Service Offering to TSMGR	42
Enable Service Offering Access	43
Provision a Service Instance	43
Update a Service Offering	44
List Service Instances	45
Upgrade a Service Instance	45
Delete a Service Offering	46
About User Authentication with TSMGR	47
About Service Offerings	48
Overview	48
Service Offering Directory	48
Service Offerings in Cloud Foundry Marketplace	49
Preparing a Service Offering	50
Overview	50
Prerequisites	50
Create a Sample Service Offering Directory	51
(Optional) Create tsmgr.yaml	51
(Optional) Override Chart Values	52
(Optional) Define Plans Configuration	52
(Optional) Offer Multiple Charts in a Single Offering	54
(Optional) Create Binding Template for App Consumption	54
Load Container Images into a Private Container Registry	57
Ensure Your Images are Accessible	58
Configuring a Helm Chart	59
Overview	59
Override Chart Values	60
Define Custom Resource Definitions	61
Prevent Secret Regeneration	61
Configure Template Values	62

Troubleshooting Errors	63
Troubleshoot Errors	63
Techniques for Troubleshooting	64
Check CLI and Server Versions	64
Run Helm Commands	64
Clean Up a Service Instance	65
View Service Instance Metadata in Kubernetes	66
View TSMGR Logs	67
Modify Timeouts	68
Modify the Provisioning Timeout	68
Modify the Cloud Foundry Broker Registration Timeout	68

Tanzu Service Manager



Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

This topic is an overview of Tanzu Service Manager (TSMGR).

Overview

Tanzu Service Manager (TSMGR) is a feature within VMware Tanzu Application Service for VMs (TAS for VMs) that enables operators to provide app developers with access to services through the `cf marketplace`. These services run on Kubernetes clusters such as those created with Tanzu Kubernetes Grid Integrated Edition for consumption by TAS for VMs apps. TSMGR is available to all customers who use TAS for VMs.

For more information about the `cf marketplace`, see [Managing Service Instances with the cf CLI](#). For more information about TKGI, see [Tanzu Kubernetes Grid Integrated Edition](#).

TSMGR enables platform operators to:

- Offer ISV and open-source Helm charts as on-demand services to TAS for VMs developers.

TSMGR enables app developers to:

- Provision dedicated service instances onto a Kubernetes cluster through the `cf marketplace`.
- Consume services running on TKGI in apps running on TAS for VMs.

Key Features

TSMGR includes the following key features:

- A service broker that provisions dedicated service instances into a Kubernetes cluster and binds service instances to TAS for VMs apps.
- A Command Line Interface (CLI) for managing service offerings.
- The ability to create and manage a catalog of services to the `cf marketplace`.

Product Snapshot

The following table provides version and version-support information about TSMGR:

Element	Details
TSMGR version	1.0.11
Release date	October 9, 2020
Compatible TAS for VMs versions	2.10, 2.9, 2.8, 2.7, 2.6, 2.5, and 2.4
Compatible TKGI versions	1.7, 1.6, 1.5, 1.4, and 1.3
Compatible Kubernetes versions	1.18, 1.17, and 1.16
IaaS support	AWS, Azure, GCP, OpenStack, and vSphere
IPsec support	No

Requirements

TSMGR requires:

- **TAS for VMs Deployment:** A running TAS for VMs deployment.
- **Kubernetes Cluster:** A running Kubernetes cluster. TSMGR supports Tanzu Kubernetes Grid Integrated Edition clusters. For information about TKGI, see [Tanzu Kubernetes Grid Integrated Edition](#).

Limitations

Dedicated service instances created by TSMGR either provision an IaaS load balancer or use an ingress controller for ingress into each service instance. You cannot use ClusterIP for ingress.

Feedback

If you have a feature request, questions or information about an issue, send an email to [Tanzu Service Manager \(TSMGR\) Feedback](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Release Notes



Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

These are release notes for Tanzu Service Manager (TSMGR).

For product versions and upgrade paths, see [Upgrade Planner](#).



Breaking Change: This release is incompatible with the v0.11 beta release of TSMGR. Before you install TSMGR v1.0, you must uninstall TSMGR v0.11. After you install TSMGR v1.0, you must re-add your service offerings.

v1.0.11

Release Date: October 9, 2020

Features

New features and changes in this release:

- **Improvements to the install dev script:** `install-dev.sh` pulls images directly from registry.pivotal.io.
- **Installation images are no longer on VMware Tanzu Network:** You pull images directly from registry.pivotal.io.
- **Improve output of `tsmgr instance list -o json` command:** This makes the output more usable in scripts.
- **Labels to better support Istio service mesh:** Additional labels have been added to the pods.

Known Issue

This release has the following issue:

- **False reports of service instance creation:** If your chart uses custom resources, KSM might incorrectly report that a service instance was created successfully before the custom resources are created. This is because Kubernetes does not have a generic method to detect resource readiness. Kubernetes primitives have known patterns for detecting readiness, but custom resource readiness is defined by its author.

In the unlikely event that this happens, wait until the custom resources are created before

binding a service instances to an app. This mitigates any issues with missing details in the bind response.

v1.0.1

Release Date: October 5, 2020

Features

New features and changes in this release:

- **Token authentication:** TSMGR CLI uses token authentication. For more information, see [About User Authentication with TSMGR](#).
- **Offer name:** The Marketplace name is now referred to as the offer name.
- **Broker component split:** The broker component is now split into two Pods: broker and reconciler.
- **Structured JSON logging:** Logging is now structured JSON.

Known Issue

This release has the following issue:

- **False reports of service instance creation:** If your chart uses custom resources, KSM might incorrectly report that a service instance was created successfully before the custom resources are created. This is because Kubernetes does not have a generic method to detect resource readiness. Kubernetes primitives have known patterns for detecting readiness, but custom resource readiness is defined by its author.

In the unlikely event that this happens, wait until the custom resources are created before binding a service instances to an app. This mitigates any issues with missing details in the bind response.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Architecture and Concepts



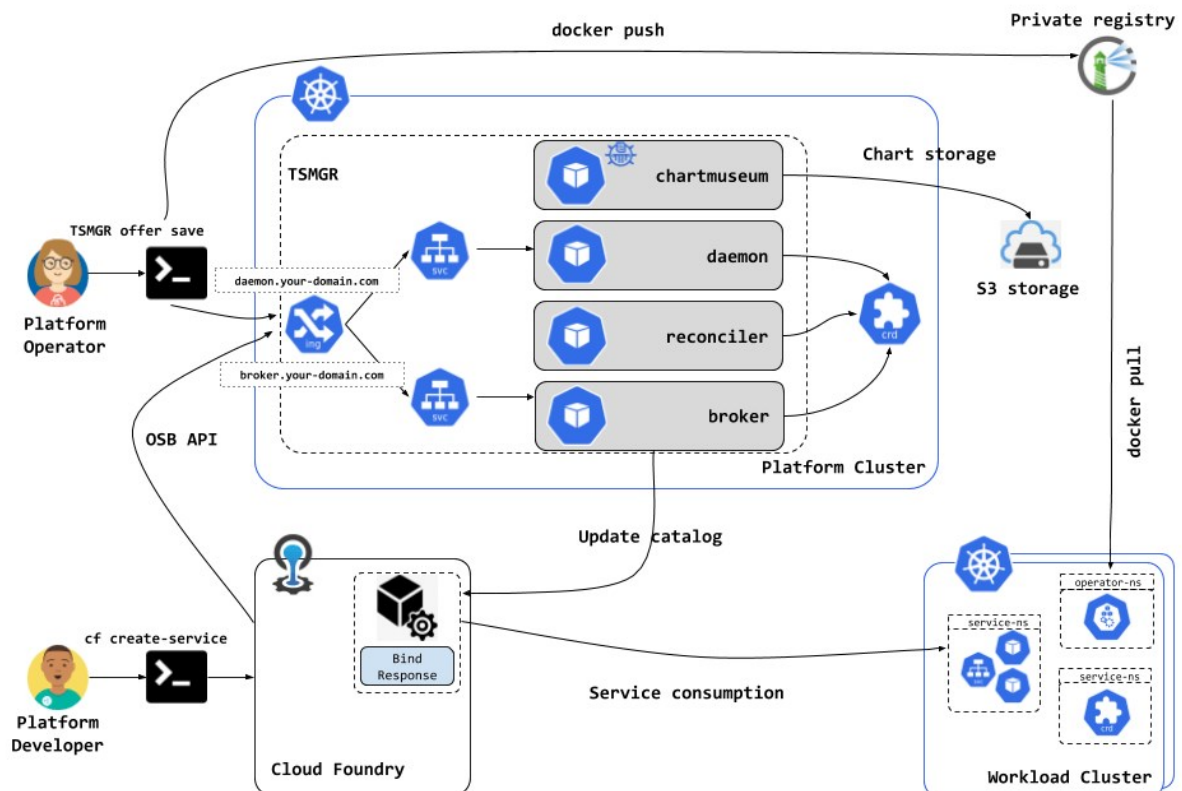
Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

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

Overview

The following diagram describes how TSMGR deployed on Kubernetes communicates with TAS for VMs and other workload clusters:



[View a larger version of this diagram.](#)

TSMGR 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:`

- The daemon validates the offer.
- The daemon stores the Helm charts in ChartMuseum. ChartMuseum uses S3 Storage.
- The daemon stores the offer in the Kubernetes cluster where TSMGR is installed.
- 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 VMware Tanzu Application Service for VMs (TAS for VMs) developer runs `cf create-service:`

- Cloud Controller sends that request to the broker.
- 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 about configuring a default Kubernetes cluster, see [Managing Kubernetes Clusters for Tanzu Service Manager](#).

TSMGR Life Cycle

This section outlines what happens when a TAS for VMs 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 TAS for VMs developer runs `cf create-service`, TSMGR deploys Helm charts into 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. Gets the Kubernetes cluster defined in the specific plan. If no cluster is defined, TSMGR uses the default cluster.
3. Deploys the chart into 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 your 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 TAS for VMs 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**, `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**, `services` and `secrets` are used directly as credentials.

Update

When a TAS for VMs developer runs `cf update-service`, TSMGR triggers the Helm upgrade process. For information about the Helm upgrade, see 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 TAS for VMs 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 this instance is the last instance of a `cluster` scope chart.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Overview of Platform Operator Tasks



Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

This topic provides an overview of configuring and installing Tanzu Service Manager (TSMGR).

Task List for Getting Started with TSMGR

The platform operator sets up TSMGR by following the steps below. Each step links to more detailed instructions.

To get started with TSMGR:

1. **Install and configure a Kubernetes cluster:** TSMGR supports Tanzu Kubernetes Grid Integrated Edition clusters. For information about TKGI, see [Tanzu Kubernetes Grid Integrated Edition](#).
2. **Install and configure TSMGR:** You can choose to install TSMGR quickly for evaluation or to install it configured for a production environment.
 - ♦ For lab and evaluation environments, see [Installing and Configuring in Development Mode](#).
 - ♦ For production environments, see [Installing and Configuring](#).
3. **Configure the TSMGR CLI:** See [Configure the TSMGR CLI](#).
4. **Register your Kubernetes workload clusters with TSMGR:** You can register clusters as well as set a default cluster. See [Register a Kubernetes Cluster](#) or [Set a Default Kubernetes Cluster](#).
5. **Add an example service offering:** For evaluation or testing purposes, you can add the Bitnami MySQL chart to TSMGR. To add the Bitnami MySQL chart:
 1. Clone the [bitnami/charts](#) repo:

```
git clone https://github.com/bitnami/charts.git
```
 2. Add the MySQL service offering by following the procedure in [Add a Service Offering to TSMGR](#).
6. **Enable developer access to the service offering:** See [Enable Service Offering Access](#).

Next Steps

After completing the process outlined above, developers can use Cloud Foundry Command Line Interface (cf CLI) commands to manage a MySQL service instance. This service instance runs on a Kubernetes cluster.

Follow the procedures in [Configuring a Helm Chart](#) and [Preparing a Service Offering](#) to add additional services to your marketplace.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Installing and Configuring Tanzu Service Manager



Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

This topic describes how to install and configure Tanzu Service Manager (TSMGR) for production using Helm.

Overview

TSMGR is installed and configured using a Helm chart.

To install and configure TSMGR:

1. [Get TSMGR Resources From VMware Tanzu Network](#)
2. [Install the TSMGR CLI](#)
3. [Replicate TSMGR Images](#)
4. [\(Recommended\) Configure Service Mesh](#)
5. [Configure the TSMGR Helm Chart](#)
6. [Install the TSMGR Helm Chart](#)
7. [\(Recommended\) Configure Security](#)
8. [\(Optional\) Install Prometheus](#)
9. [Next Steps](#)

Prerequisites

Before you install TSMGR using Helm, you must have:

- **Helm 3 CLI (v3.2.4 or later):** For information about installing the Helm CLI, see the [Helm documentation](#).
- **Docker CLI:** For information about installing Docker, see the [Docker documentation](#).
- **TAS for VMs Deployment:** A running TAS for VMs deployment.
- **Kubernetes Cluster:** A running Kubernetes cluster. TSMGR supports Tanzu Kubernetes Grid Integrated Edition clusters. For supported cluster versions, see the [Product Snapshot](#). For information about TKGI, see [Tanzu Kubernetes Grid Integrated Edition](#).

- **Kubernetes CLI:** For information about installing kubectl, see the [Kubernetes documentation](#).
- **S3 Compatible Storage:** TSMGR requires a S3-compatible bucket to store offered charts and the chart cache. For more information, see [Configuring External Storage](#).
- **Private Container Image Registry:** You need this to manage container images in air-gapped environments. VMware recommends using a registry in production deployments. You can use a registry such as [Harbor](#).
- **Install the Ingress controller on the cluster where you want to install TSMGR:** You must also reserve a subdomain for TSMGR in your DNS. For information about Ingress, see the [Kubernetes documentation](#).
- **Cloud Foundry CLI (cf CLI):** To download the cf CLI, see [Cloud Foundry CLI](#) in GitHub. Although, the cf CLI is not required for the procedures in this topic, it is required to give developers access to services. See [Using Services with Tanzu Service Manager](#).

Get TSMGR Resources From VMware Tanzu Network

To get the resources needed from VMware Tanzu Network to install TSMGR:

1. Log in and navigate to **Tanzu Service Manager (TSMGR)** in [VMware Tanzu Network](#).
2. Get the following resources:
 - ✦ **The TSMGR Command Line Interface (CLI):** Click **CLIs** and download the CLI for your operating system.
 - ✦ **Docker pull commands:** Click **Artifact References** and record the `docker pull` commands for the **broker**, **daemon**, and **chartmuseum**.
 - ✦ **Helm chart TGZ file:** Click **tsmgr-VERSION-NUMBER.tgz** and download the Helm chart for TSMGR.
 - ✦ **values-production.yaml file:** Download the `values-production.yaml` override configuration file.

Install the TSMGR CLI

To install the TSMGR Command Line Interface (CLI):

1. Rename the downloaded TSMGR CLI file as `tsmgr`.
2. Make the TSMGR binary act as an executable file by running:

```
chmod +x tsmgr
```

3. Move the binary file into your `PATH` by running:

```
mv tsmgr /usr/local/bin/tsmgr
```

4. Ensure TSMGR CLI is properly working:

```
tsmgr version
```

Replicate TSMGR Images

To replicate your TSMGR images to a private container image registry:

1. Relocate the images to your local machine: Run the `docker pull` commands recorded in [Get TSMGR Resources From VMware Tanzu Network](#) above, to pull the images. The registry credentials are the same as your VMware Tanzu Network credentials.
2. Tag the images for your registry by running these commands:

```
docker tag registry.pivotal.io/tanzu-service-manager/broker:VERSION-NUMBER \
  REGISTRY/tanzu-service-manager/broker:VERSION-NUMBER
```

```
docker tag registry.pivotal.io/tanzu-service-manager/daemon:VERSION-NUMBER \
  REGISTRY/tanzu-service-manager/daemon:VERSION-NUMBER
```

```
docker tag registry.pivotal.io/tanzu-service-manager/chartmuseum:CHARTMUSEUM-VERSION-NUMBER \
  REGISTRY/tanzu-service-manager/chartmuseum:CHARTMUSEUM-VERSION-NUMBER
```

Where:

- `VERSION-NUMBER` is the TSMGR release version number. This value is in the `docker pull` command that you recorded in [Get TSMGR Resources From VMware Tanzu Network](#) above.
- `REGISTRY` is the private container image registry you configured for TSMGR.
- `CHARTMUSEUM-VERSION-NUMBER` is the ChartMuseum version number. This value is in the `docker pull` command that you recorded in [Get TSMGR Resources From VMware Tanzu Network](#) above.

For example:

```
$ docker tag registry.pivotal.io/tanzu-service-manager/broker:1.0.1 \
  privateregistry.domain.com/tanzu-service-manager/broker:1.0.1

$ docker tag registry.pivotal.io/tanzu-service-manager/daemon:1.0.1 \
  privateregistry.domain.com/tanzu-service-manager/daemon:1.0.1

$ docker tag registry.pivotal.io/tanzu-service-manager/chartmuseum:1.0.1 \
  privateregistry.domain.com/tanzu-service-manager/chartmuseum:1.0.1
```

3. Create the `tanzu-service-manager` project in your registry.
4. Push the images to your registry by running these commands:

```
docker push REGISTRY/tanzu-service-manager/broker:VERSION-NUMBER
```

```
docker push REGISTRY/tanzu-service-manager/daemon:VERSION-NUMBER
```

```
docker push REGISTRY/tanzu-service-manager/chartmuseum:CHARTMUSEUM-VERSION-NUMBER
```

Where **REGISTRY** is the private container image registry path.

For example:

```
$ docker push privateregistry.domain.com/tanzu-service-manager/broker:1.0.1
$ docker push privateregistry.domain.com/tanzu-service-manager/daemon:1.0.1
$ docker push privateregistry.domain.com/tanzu-service-manager/chartmuseum:1.0.1
```

(Recommended) Configure Service Mesh

You can secure traffic between TSMGR components by using a service mesh, such as [Istio](#).

To configure an Istio service mesh:

1. Install Istio on your Kubernetes cluster by following the [Istio documentation](#). Follow the steps in the installation guide that best meets your needs.
2. Inject Istio into the namespace where TSMGR will be deployed by running:

```
kubectl create ns TSMGR-NAMESPACE
```

```
kubectl label namespace TSMGR-NAMESPACE istio-injection=enabled
```

Where **TSMGR-NAMESPACE** is a name you choose for the TSMGR dedicated namespace.

Configure the TSMGR Helm Chart



Note: To see a detailed description of each value and its default, run: `helm show values tsmgr-VERSION-NUMBER.tgz`.

To configure the TSMGR Helm chart, edit the `values-production.yaml` file:

1. Add the credentials for the registry where you replicated the TSMGR images:

```
imageCredentialsForTSMGRImages:
  registry: REGISTRY
  username: REGISTRY-USERNAME
  password: REGISTRY-PASSWORD
```

Where:

- ♦ **REGISTRY** is the registry you configured for installation images, for example, `privateregistry.domain.com/tanzu-service-manager`.
- ♦ **REGISTRY-USERNAME** is the username for the registry.
- ♦ **REGISTRY-PASSWORD** is the password for the registry.

TSMGR uses this registry for TSMGR installation Docker images. A new secret named `registrySecretName` of type `dockerconfigjson` is created with these credentials.

2. Add the credentials for the registry where the service instance images come from.

Service instance refers to the Helm chart files that TSMGR manages as services, such as mysql, postgresql, and etc-operator.

```
imageCredentialsForServiceInstances:
  registry: REGISTRY-INSTANCES
  username: REGISTRY-INSTANCES-USERNAME
  password: REGISTRY-INSTANCES-PASSWORD
```

Where:

- ✦ `REGISTRY-INSTANCES` is the registry you configured to offer images, for example, `anotherregistry.domain.com/project`.
- ✦ `REGISTRY-INSTANCES-USERNAME` is the username for the registry. This user can have read-only access.
- ✦ `REGISTRY-INSTANCES-PASSWORD` is the password for the registry.

TSMGR uses this registry as the backing registry for the services that TSMGR deploys. TSMGR modifies the Helm charts that you offer to point to images in the registry.



Note: Although this configuration is optional, VMware recommends using a private container registry in production.

3. Define values for your registry by configuring the repository attributes:

```
broker:
  image:
    repository: REGISTRY/tanzu-service-manager/broker
daemon:
  image:
    repository: REGISTRY/tanzu-service-manager/daemon
chartmuseum:
  image:
    repository: REGISTRY/tanzu-service-manager/chartmuseum
```

Where `REGISTRY` is your private container image registry, for example, `privateregistry.domain.com`.

4. Define a secure password to authenticate your services by configuring the password attributes:

```
broker:
  password: BROKER-PASSWORD
chartmuseum:
  env:
    open:
      BASIC_AUTH_PASS: CHARTMUSEUM-PASSWORD
```

Where:

- ✦ `BROKER-PASSWORD` is a secure password for the TSMGR broker.
- ✦ `CHARTMUSEUM-PASSWORD` is a secure password for ChartMuseum.

5. Create a User Account and Authentication (UAA) client for TSMGR to use to register the

broker and populate the catalog:

```
uaac target uaa.SYSTEM-DOMAIN
uaac token client get admin -s UAA-ADMIN-CLIENT-SECRET
uaac client add CLIENT-ID -s CLIENT-SECRET \
  --authorized_grant_types client_credentials,refresh_token \
  --scope cloud_controller.read,cloud_controller.write \
  --authorities cloud_controller.admin
```

Where:

- ✦ `SYSTEM-DOMAIN` is the system domain for TAS for VMs.
- ✦ `UAA-ADMIN-CLIENT-SECRET` is the UAA Admin Client Credentials for TAS for VMs.
- ✦ `CLIENT-ID` is the name of the client.
- ✦ `CLIENT-SECRET` is the secret of the client.

For information about UAA clients, see [User Account and Authentication \(UAA\) Server](#) in the Cloud Foundry documentation.

6. Get the annotation information required by your Ingress controller. Use the appropriate section below:

- ✦ **If you are using an automated certificate management provider such as cert-manager:** Follow the procedures to install and configure the prerequisites for the certificate management provider that you are using.
For example, the prerequisite for `cert-manager` is to set up an `Issuer` on the cluster.
For more information, see the [cert-manager documentation](#).
- ✦ **If you are using your own TLS certificates:** Create secrets with TLS certificate data.

```
kubectl create secret tls daemon-cert --key DAEMON-KEY-FILE --cert DAEMON-
CERT-FILE -n NAMESPACE
```

```
kubectl create secret tls broker-cert --key BROKER-KEY-FILE --cert BROKER-
CERT-FILE -n NAMESPACE
```

Where:

- `NAMESPACE` is the namespace where TSMGR will be installed or, if you use an Istio Ingress controller, `NAMESPACE` is the same namespace as the `istio-ingressgateway` deployment, typically `istio-system`.
 - `DAEMON-KEY-FILE` and `DAEMON-CERT-FILE` are the paths to your TLS private key and certificate for the daemon.
 - `BROKER-KEY-FILE` and `BROKER-CERT-FILE` are the paths to your TLS private key and certificate for the broker.
7. If you are using your own TLS certificates, follow the steps in [Setting Trusted Certificates](#) to ensure that Ops Manager and TAS for VMs trust the certificate.
 8. Add the following Ingress section to your `tsmgr/values.yml` file:

```

ingress:
  enabled: true
  hosts:
    - name: INGRESS-DOMAIN
      path: INGRESS-PATH
  annotations:
    ANNOTATION-KEY: ANNOTATION-VALUE
  tls:
    - secretName: daemon-cert
      hosts:
        - daemon.INGRESS-DOMAIN
    - secretName: broker-cert
      hosts:
        - broker.INGRESS-DOMAIN

```

Where:

- ✦ `INGRESS-DOMAIN` is the name of your provisioned domain.
- ✦ `INGRESS-PATH` is the path expression to match all paths for your particular ingress controller.
 Istio Ingress controller is set to `"/*"`.
 NGINX Ingress controller is set to `"/"`.
 Contour Ingress controller is set to `"/"`.
- ✦ `ANNOTATION-KEY` and `ANNOTATION-VALUE` is the annotation your Ingress controller requires.

These values depend on the Ingress controller and certificate management option you use. For example, see the annotations for Istio Ingress controller and cert-manager:

```

annotations:
  kubernetes.io/ingress.class: istio
  cert-manager.io/issuer: "letsencrypt-prod"

```



Note: Implementation details vary by Ingress controller. For more detailed usage instructions, see the documentation for your chosen Ingress controller.

9. Configure the Cloud Foundry environment details:

```

cf:
  apiAddress: https://api.SYSTEM-DOMAIN
  client: CLIENT-ID
  clientSecret: CLIENT-SECRET
  brokerName: tsmgr
  brokerUrl: https://broker.INGRESS-DOMAIN

```

Where:

- ✦ `SYSTEM-DOMAIN` is the system domain for TAS for VMs.
- ✦ `CLIENT-ID` is the client ID created in the step [Create a User Account and Authentication client](#) above. Alternatively, you can use an existing client ID for a TAS for VMs account with `cloud_controller.admin` permissions.

- ✦ `CLIENT-SECRET` is the client secret for the TAS for VMs account.
- ✦ `INGRESS-DOMAIN` is the name of your provisioned domain.



Note: Alternatively, you can use the Cloud Foundry username and password, but this is discouraged for production. When using the username and password, replace `client:` with `username:` and `clientSecret:` with `password:`.

10. Verify the Cloud Foundry `apiAddress` by running the `cf target` command.
11. Add the credentials for your S3-compatible bucket using the template below:

```
chartmuseum:
  env:
    open:
      STORAGE_AMAZON_BUCKET: BUCKET-NAME
      STORAGE_AMAZON_ENDPOINT: ENDPOINT
    secret:
      AWS_ACCESS_KEY_ID: ACCESS-KEY
      AWS_SECRET_ACCESS_KEY: SECRET
```

Where:

- ✦ `BUCKET-NAME` is your S3 bucket name.
- ✦ `ENDPOINT` is your S3 endpoint. For example, in Google Cloud Platform (GCP) it is `storage.googleapis.com`.
- ✦ `ACCESS-KEY` is your S3 access key ID.
- ✦ `SECRET` is your S3 secret access key.

The above credentials are for AWS. Depending on your IaaS, the credentials might not be a comprehensive list of the keys you need. For example, if you are not using the default region, you might need to add the `STORAGE_AMAZON_REGION`:

```
chartmuseum:
  env:
    open:
      STORAGE_AMAZON_REGION: us-east-1
```

For more information about configurations, see [ChartMuseum Helm Chart](#) in GitHub.

12. Save the `values-production.yaml` file.

Install the TSMGR Helm Chart

To install the TSMGR Helm chart:

1. From the root level of the chart, install the TSMGR Helm chart by running these commands:

```
kubectl create ns TSMGR-NAMESPACE
```

```
helm install RELEASE-NAME tsmgr-VERSION-NUMBER.tgz -n TSMGR-NAMESPACE --wait -
```

```
f values-production.yaml
```

Where:

- ✦ `TSMGR-NAMESPACE` is a name you choose for the TSMGR dedicated namespace.
- ✦ `RELEASE-NAME` is a name you choose for the release. Helm release names must begin and end with lowercase alphanumeric characters and can only contain lowercase alphanumeric characters and hyphens.
- ✦ `tsmgr-VERSION-NUMBER.tgz` is the TSMGR Helm chart file you downloaded earlier.

(Recommended) Configure Security

VMware recommends configuring security on your Kubernetes cluster for TSMGR.

To configure security:

1. Secure TSMGR secrets by using a secret provider. See [Encrypting Secret Data at Rest](#) in the Kubernetes documentation.
2. Enable network policies on the cluster to secure traffic between services. See [Network Policies](#) in the Kubernetes documentation. Some settings can vary between clouds. For example, in GKE, network policies are not enabled by default. For more information, see your cloud-specific documentation.

(Optional) Install Prometheus

You can view metrics for TSMGR if you have Prometheus running in the cluster. You must install Prometheus in each cluster that you want to view metrics for.

To install Prometheus to a cluster:

1. Install the Prometheus Helm chart by running these commands:

```
kubectl create ns prometheus

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

helm install my-prometheus prometheus-community/prometheus -n prometheus
```

2. Create a Kubernetes port forward to your local host by running these commands:

```
export POD_NAME=$(kubectl get pods --namespace prometheus -l "app=prometheus,component=server" -o jsonpath="{.items[0].metadata.name}")

kubectl --namespace prometheus port-forward $POD_NAME 9090
```

3. Access the Prometheus UI in your web browser at <http://localhost:9090>.
4. To view metrics for TSMGR, type `{app_kubernetes_io_name=~".*-tsmgr"}` in the expression box and click **Execute**.

Next Steps

After installing and configuring TSMGR, you can start using TSMGR. For information, see [Using Tanzu Service Manager](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Installing and Configuring Tanzu Service Manager in Development Mode



Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

This topic describes how to install and configure Tanzu Service Manager (TSMGR) for development using Helm.



Note: This topic describes how to quickly install TSMGR for the purpose of testing or learning. This is **NOT SECURE FOR PRODUCTION**. For production installation, see [Installing and Configuring Tanzu Service Manager](#).

Overview

There are two methods for installing the TSMGR environment for development:

- **Install Using the `install-dev.sh` Script:** The `install-dev.sh` script automates the installation steps. This option only requires you to download files and provide private registry endpoint, username, and password.
- **Install Manually:** This option requires more input from you, but gives you more visibility into the process.

Prerequisites

Before you install TSMGR using Helm, you need:

- **Helm CLI v3.2.4 or later:** For information about installing the Helm CLI, see the [Helm documentation](#).
- **Docker CLI:** For information about installing Docker, see the [Docker documentation](#).
- **TAS for VMs Deployment:** A running TAS for VMs deployment.
- **A Kubernetes Cluster:** A running Kubernetes cluster. TSMGR supports Tanzu Kubernetes Grid Integrated Edition clusters. For supported cluster versions, see the [Product Snapshot](#). For information about TKGI, see [Tanzu Kubernetes Grid Integrated Edition](#).
- **Kubernetes CLI:** For more information, see the [Kubernetes documentation](#).
- **Private Container Image Registry:** You need this to manage container images in air-gapped

environments. VMware recommends using a registry in production deployments. You can use a registry such as [Harbor](#).

- **A default StorageClass on the cluster where you want to install TSMGR:** If you are using TKGI, see [Specify a Default StorageClass](#). For more information about storage classes, see the [Kubernetes documentation](#).
- **Cloud Foundry CLI (cf CLI):** To download the cf CLI, see [Cloud Foundry CLI](#) in GitHub. Although, the cf CLI is not required for the procedures in this topic, it is required to give developers access to services. See [Using Services with Tanzu Service Manager](#).

Install Using the install-dev.sh Script

To install using the `install-dev.sh` script:

1. Download artifacts from [VMware Tanzu Network](#) to an empty directory:
 - ✦ **The TSMGR Command Line Interface (CLI):** Click **CLIs** and download the CLI for your operating system.
 - ✦ **TSMGR Helm chart TGZ file:** Click **Helm Chart for TSMGR** and download the Helm chart TGZ file.
 - ✦ **install-dev.sh script:** Click **TSMGR Dev Environment Installer** and download the `install-dev.sh` file.
2. Run `docker login registry.pivotal.io` so that the script can pull the installation images. The credentials are the same as the VMware Tanzu Network credentials.
3. Make the `install-dev.sh` file executable by running:

```
chmod +x install-dev-VERSION.sh
```

Where `VERSION` is the version number of TSMGR that you are installing.

For example:

```
$ chmod +x install-dev-1.0.11.sh
```

4. For the private image registry that you will use to upload the Docker installation images, verify and record the following:
 - ✦ Endpoint
 - ✦ Username
 - ✦ Password
5. Provide the registry endpoint, username, and password during the installation by doing one of these methods:
 - ✦ Run `./install-dev.sh` and then answer the registry details questions, or
 - ✦ Create a `registry.yaml` file containing the needed information. See an example `registry.yaml` template below:

```
registry: MY-REGISTRY-ENDPOINT/PROJECT/REPOSITORY
```



```
username: USERNAME
password: PASSWORD
```

and then run:

```
install-dev.sh registry.yaml
```

6. Monitor the provisioning of the external IP addresses for the daemon and broker services by running:

```
kubectl get services -n tsmgr-dev
```

7. After the external IP address is available for the daemon, configure the TSMGR environment variables by running:

```
source ./set-tsmgr-env.sh
```

8. After the external IP address is available for the broker, configure TSMGR for Cloud Foundry by running the following script:

```
./set-cf-details.sh BROKER-NAME CF-API-ENDPOINT CF-USERNAME CF-PASSWORD
```

Where:

- ✦ **BROKER-NAME** is a name you provide to your broker to be identified in Cloud Foundry.
- ✦ **CF-API-ENDPOINT** is the Cloud Foundry endpoint. To verify that, run the `cf target` command.
- ✦ **CF-USERNAME** is the username for a TAS for VMs account with `cloud_controller.admin` permissions. For example: `.uaa.admin_credentials`.
- ✦ **CF-PASSWORD** is the password for the TAS for VMs account.

9. Test the TSMGR by running either:

```
tsmgr version
```

or

```
tsmgr offer list
```

Install Manually

To install TSMGR manually, follow the procedures below:

1. [Get TSMGR Resources from VMware Tanzu Network](#)
2. [Install the TSMGR CLI](#)
3. [Configure the TSMGR Helm Chart](#)
4. [Install the TSMGR Helm Chart](#)
5. [Configure TSMGR for Cloud Foundry](#)

Get TSMGR Resources from VMware Tanzu Network

To get the resources needed from VMware Tanzu Network to install TSMGR:

1. Go to [VMware Tanzu Network](#).
2. Sign in, if you are not already signed in.
3. Get the following resources:
 - ♦ **The TSMGR Command Line Interface (CLI):** Click **CLIs** and download the CLI for your operating system.
 - ♦ **TSMGR Helm chart TGZ file:** Click **tsmgr-VERSION-NUMBER.tgz** and download the Helm chart for TSMGR.

Install the TSMGR CLI

To install the TSMGR Command Line Interface (CLI):

1. Rename the downloaded TSMGR CLI file as `tsmgr`.
2. Make the TSMGR binary file act as an executable file by running:

```
chmod +x tsmgr
```

3. Move the binary file into your `PATH` by running:

```
mv tsmgr /usr/local/bin/tsmgr
```

4. Ensure TSMGR CLI is properly working by running:

```
tsmgr version
```

Configure the TSMGR Helm Chart



Note: To see a detailed description of each value and its default, run: `helm show values tsmgr-VERSION-NUMBER.tgz`

To configure the TSMGR, edit the `values-dev.yaml` file:

1. Add the credentials for the VMware Tanzu Network registry:

```
imageCredentialsForTSMGRImages:
  registry: registry.pivotal.io
  username: REGISTRY-USERNAME
  password: REGISTRY-PASSWORD
```

Where:

- ♦ `REGISTRY-USERNAME` is your VMware Tanzu Network username.
 - ♦ `REGISTRY-PASSWORD` is your VMware Tanzu Network password.
2. (Recommended) Set unique passwords:

The `values-dev.yaml` is pre-configured with default passwords, but VMware recommends changing them to unique passwords in these sections:

```
broker:
  password: DevPassword
```

```
daemon:
  password: DevPassword
```

```
chartmuseum:
  env:
    open:
      BASIC_AUTH_PASS: "chartpass"
```

```
minio:
  accessKey:
    password: DevPassword
  secretKey:
    password: DevPassword
```

Install the TSMGR Helm Chart

To install the TSMGR Helm chart:

1. From the root level of the chart, install the TSMGR Helm chart by running:

```
kubectl create ns TSMGR-NAMESPACE
```

```
helm install RELEASE-NAME tsmgr-VERSION-NUMBER.tgz -n TSMGR-NAMESPACE --wait -f values-dev.yaml
```

Where:

- ✦ `TSMGR-NAMESPACE` is a name you choose for the TSMGR dedicated namespace.
- ✦ `RELEASE-NAME` is a name you choose for the release. Helm release names must begin and end with lowercase alphanumeric characters and can only contain lowercase alphanumeric characters and hyphens.
- ✦ `tsmgr-VERSION-NUMBER.tgz` is the TSMGR Helm Chart file you downloaded earlier.

Configure TSMGR for Cloud Foundry

To configure TSMGR for Cloud Foundry:

1. Retrieve the broker IP address:
 - ✦ **If the Kubernetes provider is not host-based, run:**

```
export BROKER_IP=$(kubectl get service RELEASE-NAME-tsmgr-broker -n TSMGR-NAMESPACE -o=jsonpath='{@.status.loadBalancer.ingress[0].ip}')
```

- ✦ **If the Kubernetes provider is host-based, such as Amazon's EKS, run:**

```
export BROKER_IP=$(kubectl get service RELEASE-NAME-tsmgr-broker -n
TSMGR-NAMESPACE -o=jsonpath='{@.status.loadBalancer.ingress[0].hostname}')
```

2. Upgrade your TSMGR Helm release by running:

```
helm upgrade RELEASE-NAME tsmgr-VERSION-NUMBER.tgz --reuse-values \
--set cf.brokerUrl="http://${BROKER_IP}" \
--set cf.brokerName=tsmgr \
--set cf.apiAddress=http://api.SYSTEM-DOMAIN \
--set cf.username=CF-USERNAME \
--set cf.password=CF-PASSWORD \
--set cf.skipSslValidation=true \
-n TSMGR-NAMESPACE
```

Where:

- ✦ `RELEASE-NAME` is the name of the release.
- ✦ `SYSTEM-DOMAIN` is the system domain for TAS for VMs.
- ✦ `CF-USERNAME` is the username for a TAS for VMs account with `cloud_controller.admin` permissions, such as the `.uaa.admin_credentials`.
- ✦ `CF-PASSWORD` is the password for the TAS for VMs account.
- ✦ `TSMGR-NAMESPACE` is a name you choose for the TSMGR dedicated namespace.



Note: You can verify `cf.apiAddress` by running the `cf target` command.

Create a pull request or raise an issue on the source for this page in GitHub

Upgrading Tanzu Service Manager



Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

This topic explains how to upgrade Tanzu Service Manager (TSMGR) using Helm.

Overview

The steps to upgrade vary depending on the version of TSMGR installed. Follow the procedure below for your version.

Upgrade From v0.11 to v1.0 or Later



Breaking Change: This release is incompatible with the v0.11 beta release of TSMGR. Before you install TSMGR v1.0, you must uninstall TSMGR v0.11. After you install TSMGR v1.0, you must re-add your service offerings.

To upgrade from TSMGR v0.11:

1. [Uninstall TSMGR v0.11.](#)
2. Install the latest version of TSMGR by following the procedures in [Installing and Configure Tanzu Service Manager](#).

Upgrade From v1.0 or Later

To upgrade from v1.0 or later:

1. [Get TSMGR resources from VMware Tanzu Network.](#)
2. Follow the steps in [Install the TSMGR CLI](#) to replace your current version of TSMGR CLI with the latest version.
3. [Replicate TSMGR images.](#)
4. Upgrade the TSMGR Helm chart by running:

```
helm upgrade RELEASE-NAME tsmgr-VERSION-NUMBER.tgz -n TSMGR-NAMESPACE --wait -
-reuse-values
```

Where:

- ✦ `TSMGR-NAMESPACE` is a name you previously used as the TSMGR-dedicated namespace.
- ✦ `RELEASE-NAME` is the name you previously used as the TSMGR release.
- ✦ `tsmgr-VERSION-NUMBER.tgz` is the new TSMGR Helm chart file you downloaded.

Uninstall TSMGR v0.11

To uninstall TSMGR:

1. View all of your installed releases across all namespaces by running:

```
helm list --all --all-namespaces
```

2. Record the name and namespace of the TSMGR release that you want to uninstall.
3. Remove the Helm installation by running:

```
helm uninstall RELEASE-NAME -n RELEASE-NAMESPACE
```

Where:

- ✦ `RELEASE-NAME` is the name of the TSMGR release that you recorded in the previous step.
- ✦ `RELEASE-NAMESPACE` is the namespace of the TSMGR release that you recorded in the previous step.

For example:

```
helm uninstall my-tsmgr-release -n tsmgr-namespace
```

4. Remove the remaining `tsmgr` namespaces by running these commands:

```
kubectl delete ns tsmgr-system

kubectl delete ns RELEASE-NAMESPACE
```

Where `RELEASE-NAME` is the name of the TSMGR release that you recorded above.

5. Delete the custom resources by running these commands:

```
kubectl delete crd bindingmetas.services.tanzu.vmware.com

kubectl delete crd clusters.services.tanzu.vmware.com

kubectl delete crd instances.services.tanzu.vmware.com

kubectl delete crd offers.services.tanzu.vmware.com
```

6. Delete `clusterrole` and `clusterrolebinding` by running these commands:

```
kubectl delete clusterrole RELEASE-NAME-tsmgr

kubectl delete clusterrolebinding RELEASE-NAME-tsmgr
```

Where `RELEASE-NAME` is the name of the TSMGR release that you recorded above.

Create a pull request or raise an issue on the source for this page in [GitHub](#)

Configuring External Storage



Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

This topic describes how to configure storage for Tanzu Service Manager (TSMGR).

Overview

TSMGR requires an S3-compatible bucket to store offered charts and the chart cache. The operator is responsible for backing up this bucket.

To configure external storage, do one of the following:

- **To configure an S3-compatible bucket**, follow the procedures for your external storage solution.
- **If you use a external storage solution that does not natively support S3 buckets**, you must configure your cloud storage to be compatible with S3. See [Configure Non-S3 Storage](#) below.



Warning: If you change the storage configurations, the service offering data is lost and you must re-add the service offering to TSMGR. For offerings with multiple versions provisioned, the old versions should also be re-added. See [Add a Service Offering to TSMGR](#).

Configure Non-S3 Storage

To configure your cloud storage provider for TSMGR, you must create a policy and an access key for your cloud storage provider.

Policy creation varies depending on your cloud storage provider. VMware recommends that only TSMGR uses this account. You should apply a minimal policy that lets the user account upload backups to your S3 store. The policy must give TSMGR permissions to list and upload buckets. For instructions on how to create a policy and access key, see your provider's documentation.

If you are using Google Cloud Storage (GCS), you must generate a developer key for GCS to use an S3-compatible API. This developer key provides the credentials required to configure storage in TSMGR. For instructions on how to generate a developer key, see the [GCP documentation](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Managing Kubernetes Clusters for Tanzu Service Manager



Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

This topic describes how to register, manage, and deregister Kubernetes clusters for use with Tanzu Service Manager (TSMGR).

Overview

Before offering a service with TSMGR, you must register the Kubernetes clusters that service instances use. For more information about how to register your clusters with TSMGR, see [Kubernetes Cluster Registration](#) below.

To register clusters, you must first create a cluster credentials file. See [Create a Cluster Credentials File](#) below.

After you have registered your clusters with TSMGR, you can provision each plan in your TSMGR service offering to a different cluster. Provisioning to different clusters allows you to require special cluster configurations. You can also set a default cluster to provision plans to. See [Set a Default Kubernetes Cluster](#) below.

Create a Cluster Credentials File

You need a cluster credential file to register Kubernetes clusters with TSMGR. A valid cluster credentials file has the following structure:

```
token: TOKEN
server: SERVER
caData: CA-DATA
```

Where:

- **TOKEN** is a valid Kubernetes authentication token.
- **SERVER** is the server address of the Kubernetes cluster.
- **CA-DATA** is valid certificate authority (CA) data for the Kubernetes cluster.

To create a cluster credentials file:

1. Get your server and certificate authority by running:

```
cluster=$(kubectl config view -o jsonpath="{.contexts[?(@.name == \"$(kubectl c
onfig current-context)\"]}.context.cluster}")

server=$(kubectl config view -o jsonpath="{.clusters[?(@.name == \"${cluster}\")]
.cluster.server}")

certificate=$(kubectl config view --raw --flatten -o jsonpath="{.clusters[?(@.n
ame == \"${cluster}\"]}.cluster.certificate-authority-data}")

echo "Using cluster $cluster"
```

2. To create a dedicated service account for TSMGR with the `cluster-admin` role in a dedicated namespace:

1. Create the namespace by running:

```
kubectl create ns tsmgr-system
```

2. Create a YAML file for the TSMGR service account and `ClusterRoleBinding` using the following YAML:

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: tsmgr-admin
  namespace: tsmgr-system
---
apiVersion: rbac.authorization.k8s.io/v1beta1
kind: ClusterRoleBinding
metadata:
  name: tsmgr-cluster-admin
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: tsmgr-admin
  namespace: tsmgr-system
```

For information about service accounts, see [Managing Service Accounts](#) in the Kubernetes documentation.

For information about `ClusterRoleBinding`, see [RoleBinding and ClusterRoleBinding](#) in the Kubernetes documentation.

3. Create the service account and `ClusterRoleBinding` defined in your YAML file by running:

```
kubectl apply -f SERVICE-ACCOUNT.yml
```

Where `SERVICE-ACCOUNT` is name of the YAML file you created in the previous step.

3. Get your service account token by running:

```
secret_name=$(kubectl get serviceaccount tsmgr-admin \
--namespace=tsmgr-system -o jsonpath='{.secrets[0].name}')

secret_val=$(kubectl --namespace=tsmgr-system get secret $secret_name \
-o jsonpath='{.data.token}')

secret_val=$(echo ${secret_val} | base64 --decode)
```

4. Save your service account token, server, and certificate to a file by running:

```
cat > cluster-creds.yaml << EOF
token: ${secret_val}
server: ${server}
caData: ${certificate}
EOF
```

Kubernetes Cluster Registration

The following sections describe how to list, register, and deregister Kubernetes clusters.

You can register individual clusters and set a default cluster.

See the following sections:

- **Register clusters individually:** See [Register a Kubernetes Cluster](#) below.
- **Set a default registered cluster:** See [Set a Default Kubernetes Cluster](#) below.
- **List clusters that are registered:** See [List Registered Kubernetes Clusters](#) below.
- **Deregister a cluster:** See [Deregister a Kubernetes Cluster](#) below.

Register a Kubernetes Cluster

Before creating a plan, you must register the Kubernetes cluster that you want service instances to use.

To register a cluster with TSMGR:

1. Create a cluster credentials YAML file with the service account token, server, and certificate. See [Create a Cluster Credentials File](#).
2. Register the cluster with TSMGR by running:

```
tsmgr cluster register CLUSTER-NAME PATH-TO-CLUSTER-CREDS.YAML
```

Where:

- **CLUSTER-NAME** is the name of your cluster. TSMGR uses this name to track the cluster. Valid characters include lowercase letters, digits, and `-`.
- **PATH-TO-CLUSTER-CREDS.YAML** is the path to your cluster credentials file that you created in [Create a Cluster Credentials File](#) above.

Set a Default Kubernetes Cluster

If you set a default cluster, you do not need to specify a cluster when you create plans for a service

offering.

To set a default cluster, you must first register the cluster. For more information, see [Register a Kubernetes Cluster](#) above.

To set a default cluster:

1. Set a default cluster by running:

```
tsmgr cluster set-default CLUSTER-NAME
```

Where `CLUSTER-NAME` is the name of a cluster that is registered with TSMGR.

List Registered Kubernetes Clusters

To see a list of registered Kubernetes clusters:

1. List registered clusters by running:

```
tsmgr cluster list
```

Deregister a Kubernetes Cluster

Before deregistering a cluster, ensure that it is not the default cluster, it does not have any instances, and the cluster is not associated with any plans in any service offering.

To deregister a cluster:

1. Deregister the cluster by running:

```
tsmgr cluster deregister CLUSTER-NAME
```

Where `CLUSTER-NAME` is the name of a cluster that is registered with TSMGR.

Update Kubernetes Clusters

To update the credentials for an existing cluster that you have already registered with TSMGR:

1. Create a cluster credentials YAML file with server, token, and caData. See [Create a Cluster Credentials File](#) above.
2. Update the Kubernetes cluster credentials by running:

```
tsmgr cluster register CLUSTER-NAME PATH-TO-CLUSTER-CREDS.YAML --update
```

Where:

- ✦ `CLUSTER-NAME` is the name of a cluster that is registered with TSMGR.
- ✦ `PATH-TO-CLUSTER-CREDS.YAML` is the path to your cluster credentials file that you created in the previous step.



Note: If existing instances are running on the cluster, use the `--force` flag.

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Using Services with Tanzu Service Manager



Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

This topic describes how to use Tanzu Service Manager (TSMGR).

Overview

TSMGR uses the TSMGR Command Line Interface (CLI) and a generic broker. You can use TSMGR to:

- Offer on-demand provisioning of a dedicated service instance for developers through `cf marketplace`.
- Create a catalog of services with customizable plans that can be deployed on multiple clusters.

To add a service offering:

1. [Configure the TSMGR CLI](#)
2. [Add a Service Offering to TSMGR](#)
3. [Enable Service Offering Access](#)

Prerequisites

Before you deploy a service offering, you must have:

- A configured Helm chart. For more information, see [Configuring a Helm Chart](#).
- A service offering. For more information, see [Preparing a Service Offering](#).
- A default cluster set for TSMGR. For more information, see [Set a Default Kubernetes Cluster](#).

Configure the TSMGR CLI

To configure the Tanzu Service Manager CLI, retrieve the API credentials and export the environment variables:

1. Output the API credentials and environment variables by running:

```
helm status RELEASE-NAME -n TSMGR-NAMESPACE
```

Where:

- ✦ `RELEASE-NAME` is the name you chose for the Tanzu Service Manager release during installation.
- ✦ `TSMGR-NAMESPACE` is the name you chose for the TSMGR dedicated namespace.

2. Follow the instructions in the output of the command above to configure the TSMGR CLI.

Add a Service Offering to TSMGR

You must package your Helm charts and add the offering to TSMGR to enable developers to use the service.

To add your offering to TSMGR:

1. Package your Helm chart by running the following command from the root level of your Helm chart directory:

```
helm package .
```

2. Verify that you now have a file named `CHART-NAME-#.##.#.tgz` in the root level of your Helm chart directory.
3. Add your Helm charts as a dedicated instance service offering to TSMGR by doing one of the following:

- ✦ **If you are adding a single chart, run:**

```
tsmgr offer save /PATH-TO-TSMGR-DIRECTORY/ /PATH-TO-YOUR-CHART.tgz
```

Where `PATH-TO-YOUR-CHART.tgz` is the path to the packaged chart you created in the above step.

- ✦ **If you are adding multiple charts, run:**

```
tsmgr offer save /PATH-TO-TSMGR-DIRECTORY/ /PATH-TO-YOUR-CHART_1.tgz /PATH-TO-YOUR-CHART_2.tgz
```

Where:

- `PATH-TO-TSMGR-DIRECTORY/` is the path to the `tsmgr` directory that includes the `tsmgr.yaml` file for your service offering.
- `PATH-TO-YOUR-CHART_1.tgz` is the path to a packaged chart you referenced in your `tsmgr.yaml` file.
- `PATH-TO-YOUR-CHART_2.tgz` is the path to a packaged chart you referenced in your `tsmgr.yaml` file.



Note: If you have already added a packaged chart to TSMGR, you do not need to provide the path to the same chart when you subsequently add service offerings.

For more information about offering multiple charts, see [\(Optional\) Offer Multiple Charts in a Single Offering](#).



Note: If the Helm chart version specified in `tsmgr.yaml` and `Chart.yaml` do not match, TSMGR returns a error message similar to `Error: The offer yaml version [2.3.0] for chart [mysql] does not match the version [1.3.0] in chart`

4. Ensure your offering was saved to TSMGR by running:

```
tsmgr offer list
```

Enable Service Offering Access

After you add your service offering, you must enable access to the service offering.

To enable access to your service offering:

1. Log in to your VMware Tanzu Application Service for VMs (TAS for VMs) deployment by running:

```
cf login -a API-URL -u USERNAME -p PASSWORD
```

Where:

- ♦ `API-URL` is the API endpoint for your TAS for VMs instance.
- ♦ `USERNAME` is your username for your TAS for VMs instance.
- ♦ `PASSWORD` is your password for your TAS for VMs instance.

For more information, see [Getting Started with the cf CLI](#).

2. Enable access to the service for all orgs by running:

```
cf enable-service-access SERVICE-NAME
```

Where `SERVICE-NAME` is the name of the new service offering that you viewed in step 6 of [Add a Service Offering to TSMGR](#) above.

3. View the newly created service plan by running:

```
cf marketplace
```

Provision a Service Instance

After enabling service access for the service offering, the service is available to developers through the Cloud Foundry Marketplace.

Developers can see the services that they have access to by running:

```
cf marketplace
```

Developers can then provision a service instance by running:

```
cf create-service SERVICE-NAME PLAN-NAME SERVICE-INSTANCE-NAME
```



Note: Developers can specify service-specific configuration parameters to override plan and Helm chart values. They pass these parameters using the `-c` flag when they provision or update a service instance through the cf CLI. The `-c` flag accepts a JSON object.

The JSON configuration values are mapped directly onto the chart values and take precedence over any chart default and plan values. For information about service-specific configuration parameters, see [Arbitrary Parameters](#).

For more information about cf CLI commands that developers can use with Cloud Foundry Marketplace services, see the [Cloud Foundry CLI Reference Guide](#).

Update a Service Offering

When you update a service offering in TSMGR, any newly-created service instances use the latest version of the service offering. Existing instances must be upgraded using the cf CLI. See [Upgrade a Service Instance](#) below.

To update a service offering in TSMGR:

1. For each chart you are updating, after you edit your Helm chart, increment the chart version in `Chart.yaml`.
2. If you have a `tsmgr.yaml` file for your service offering, update the chart versions in `tsmgr.yaml` to match the the version in `Chart.yaml`. For more information about multiple charts and `tsmgr.yaml`, see [\(Optional\) Offer Multiple Charts in a Single Offering](#).
3. If any of your charts include modified Custom Resource Definitions (CRDs), for each CRD manifest, manually apply the changes to the cluster by running:

```
kubectl apply -f PATH-TO-CRD-MANIFEST.yaml
```

Where `PATH-TO-CRD-MANIFEST.yaml` is the path to the CRD manifest. The CRD manifest is in the `crds` directory in the chart.



Note: By default, TSMGR rejects an updated service offering with any CRD changes in the chart. This is because TSMGR does not modify any CRDs that currently exist on the Kubernetes cluster.

4. Package an updated version of your Helm chart by running the following command from the root level of your Helm chart directory:

```
helm package .
```

5. Verify that you now have a file named `CHART-NAME-#.##.tgz` in the root level of your Helm chart directory.



Note: If you are updating multiple charts, run the above command for each updated chart.

6. Update the service offering by running one of the following commands:



Note: If you manually applied CRD changes to the cluster earlier, you can safely ignore a CRD warning from TSMGR by adding the `--ignore-crd-warning` flag to the below commands.

- **If you are adding a single chart, run:**

```
tsmgr offer save /PATH-TO-YOUR-CHART.tgz --update
```

Where `PATH-TO-YOUR-CHART.tgz` is the name of the service offering you want to delete.

- **If you are adding a multiple charts, run:**

```
tsmgr offer save /PATH-TO-TSMGR-DIRECTORY/ /PATH-TO-YOUR-CHART_1.tgz /PATH-TO-YOUR-CHART_2.tgz --update
```

Where:

- `PATH-TO-TSMGR-DIRECTORY/` is the path to the `tsmgr` directory that includes the `tsmgr.yaml` file for your service offering.
- `PATH-TO-YOUR-CHART_1.tgz` is the path to a packaged chart you referenced in your `tsmgr.yaml` file.
- `PATH-TO-YOUR-CHART_2.tgz` is the path to a packaged chart you referenced in your `tsmgr.yaml` file.

List Service Instances

You can list existing service instances and filter by offer name or cluster name.

To filter services instances by offer name or cluster name, choose one of the following options:

- **Filter by offer name by running:**

```
tsmgr instance list OFFER-NAME
```

Where `OFFER-NAME` is the name of the offer.

- **Filter by cluster by running:**

```
tsmgr instance list --cluster-name CLUSTER-NAME
```

Where `CLUSTER-NAME` is the name of the cluster.

Upgrade a Service Instance

After you update your service offering, ask developers to follow this procedure to upgrade their service instances. For instructions for updating a service offering, see [Update a Service Offering](#) above.

To upgrade service instances:

1. Find the service instances that you can upgrade by running:

```
cf services
```

For example:

```
$ cf services
name                service  plan    bound apps    last operation    broker
                    upgrade available
mysql-instance      mysql    small   spring-music   create succeeded   tanzu-serv
ice-manager         true
```

Service instances that are available for upgrade have `true` under the `upgrade available` column.

2. Upgrade the service instance by running:

```
cf update-service SERVICE-INSTANCE-NAME -u
```

3. See the status of your service instance and watch for completion by running:

```
watch cf services
```

For example:

```
$ watch cf services
Getting services in org my-org / space my-space as user...
OK
name                service  plan    bound apps    last operation    broker
                    upgrade available
mysql-instance      mysql    small   spring-music   upgrade succeeded   tanzu-serv
ice-manager         false
```

Service instances are upgraded asynchronously. Wait for `last operation` for your instance to show as `create succeeded`.

Delete a Service Offering



Note: TSMGR does not permit deleting service offerings if service instances of that offering exist. This is because if an offering was deleted while service instances of the offering existed, you could not perform operations against that service instance. For example, you could not run `cf update-service`.

To delete a service offering:

1. If service instances of the service offering exist, for each service instance, delete the instance by running:

```
cf delete-service SERVICE-INSTANCE-NAME
```

2. Delete a service offering by running:

```
tsmgr offer delete OFFER-NAME
```

Where `OFFER-NAME` is the name of the service offering you want to delete.



Note: If you only have one service offering in TSMGR, you cannot delete it.

About User Authentication with TSMGR

Users authenticate with TSMGR CLI using a Kubernetes token. The steps described in [Configuring the TSMGR CLI](#) above instruct you to configure the CLI with a pre-created Kubernetes service account. This service account has the appropriate permissions to work with TSMGR.

To use a different token, ensure that the Kubernetes actor – a user, service account, or group – has the correct Kubernetes RBAC.

The RBAC permissions required within the `TSMGR-NAMESPACE` are:

```
rules:
- apiGroups:
  - services.tanzu.vmware.com
  resources:
  - "*"
  verbs:
  - "*"

```

For more information about RBAC and how to set up permissions see the [Kubernetes documentation](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

About Service Offerings



Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

This topic describes the structure of a Tanzu Service Manager (TSMGR) service offering and how an offering displays in `cf marketplace`.

Overview

A service offering includes:

- **One or more Helm charts:** You must configure a Helm chart to be compatible with TSMGR. See [Configuring a Helm Chart](#).
- **A TSMGR package:** The TSMGR package contains YAML files that TSMGR uses to customize your service offering. See [Preparing a Service Offering](#).

Service Offering Directory

The following code snippet shows an example of a service offering directory structure:

```
offer-name/  
  bind.yaml  
  tsmgr.yaml  
  plans.yaml  
  overrides/  
    override-values.yaml  
  plans/  
    small.yaml  
    medium.yaml  
  resource-quotas/  
    quota.yaml  
  
helm-chart.tgz
```

In the above example:

- `offer-name/` is the directory for the TSMGR package. Depending on your customization, you must include some or all of the above YAML files and subdirectories in the TSMGR package.
- `helm-chart.tgz` is the Helm chart for the service offering.

Reference the following table to learn which files you must include in your TSMGR package:

If you want to...	Required File	More Information
Provide custom formatting for service credentials when a developer binds a service instance.	<code>bind.yaml</code>	(Optional) Create Binding Template for App Consumption
Add a custom <code>cf marketplace</code> name	<code>tsmgr.yaml</code>	Service Offerings in CF Marketplace
Offer more than one Helm chart		(Optional) Offer Multiple Charts in a Single Offering
Add a service offering with an Operator pattern		For information about Operators, see the Kubernetes documentation
Override default chart values		(Optional) Override Chart Values
Create a non-default plan in the <code>cf marketplace</code>	<code>plans.yaml</code>	(Optional) Define Plans Configuration
Provision a service on a non-default cluster		
Define resource quotas that limit memory and CPU consumption		

Service Offerings in Cloud Foundry Marketplace

Uploading a Helm chart to TSMGR with either a `tsmgr.yaml` or `plans.yaml` file overrides the default values for the service offering name, description, and plan names. These values are displayed when a TAS for VMs developer runs `cf marketplace`.

The following table describes the values that TSMGR uses when you upload a Helm chart either without additional configurations, with a `tsmgr.yaml` file, or with a `plans.yaml` file:

Additional YAML file	Name	Description	Plan Name
None	<code>name</code> value in <code>Chart.yaml</code>	<code>description</code> value in <code>Chart.yaml</code>	<code>default</code>
<code>tsmgr.yaml</code>	<code>offer-name</code> value in <code>tsmgr.yaml</code>	No effect	No effect
<code>plans.yaml</code>	No effect	For each plan, <code>description</code> value in <code>plans.yaml</code>	For each plan, <code>name</code> value in <code>plans.yaml</code>

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Preparing a Service Offering



Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

This topic describes how to prepare an offered service for Tanzu Service Manager (TSMGR).

For working examples, see [tsmgr-samples](#) in GitHub.

Overview

TSMGR enables platform operators to offer Helm charts as on-demand services to developers on VMware Tanzu Application Service for VMs (TAS for VMs).

Before you add a service offering to `cf marketplace`, you must configure the TSMGR package. You can also configure the package to enable additional features.

To prepare a service offering:

1. [Create a Sample Service Offering Directory](#)
2. [\(Optional\) Create tsmgr.yaml](#)
3. [\(Optional\) Override Chart Values](#)
4. [\(Optional\) Define Plans Configuration](#)
5. [\(Optional\) Offer Multiple Charts in a Single Offering](#)
6. [\(Optional\) Create Binding Template for App Consumption](#)
7. [Load Container Images into a Private Container Registry](#)
8. [Ensure Your Images are Accessible](#)

Prerequisites

Before you add your service offering to TSMGR, you must:

- **Determine which YAML files you want to include in the service offering:** For more information, see [About Service Offerings](#).
- **Install the TSMGR Command Line Interface (CLI):** For more information, see [Install the TSMGR CLI](#).
- **Register a Kubernetes cluster using the TSMGR CLI:** For more information, see [Managing Kubernetes Clusters for Tanzu Service Manager](#).

- **Install External Dependencies:** If your Helm charts have any external dependencies, they must be installed before you offer your service. For example, if your Helm chart requires an Ingress controller to enable service through ingress, the Ingress controller must be manually installed onto the cluster.

For information about Ingress, see the [Kubernetes documentation](#).

Create a Sample Service Offering Directory

You can use the TSMGR CLI to create a directory structure for your service offering. When you create the service offering directory with the TSMGR CLI, the directory includes the following sample files:

- `OFFER-NAME/bind.yaml`
- `OFFER-NAME/tsmgr.yaml`
- `OFFER-NAME/plans.yaml`
- `OFFER-NAME/plans/small.yaml`
- `OFFER-NAME/plans/medium.yaml`
- `OFFER-NAME/overrides/override-values.yaml`

To create a sample service offering directory:

1. Create a directory for your service offering by running:

```
tsmgr offer init OFFER-NAME
```

Where `OFFER-NAME` is the name of the service offering.



Note: If you do not include a `plans.yaml`, a `tsmgr.yaml` file, or override files with your service offering, one plan is offered. This plan uses the default values for the Helm chart.

(Optional) Create tsmgr.yaml

The `tsmgr.yaml` file is required to either offer multiple charts or specify a file overriding a chart's default values.

1. Create the `tsmgr.yaml` file using either the `init` output or the following as a template:

```
offer-name: OFFER-NAME
charts:
  - chart: CHART-NAME
    version: CHART-VERSION-NUMBER
    offered: OFFERED-VALUE
    scope: SCOPE-VALUE
    overrideValuesFile: OVERRIDE-FILE.yaml
```

Where:

- ✦ `OFFER-NAME` is the name of the service offering that developers see when they run `cf marketplace`. The service name is listed under `service`.
- ✦ `CHART-NAME` is the name of a chart included in the service offering.
- ✦ `CHART-VERSION-NUMBER` is the version of the chart included in the service offering.
- ✦ `OFFERED-VALUE` is either `true` or `false`. For more information, see the below table.
- ✦ `SCOPE-VALUE` is either `cluster` or `namespace`. For more information, see the below table.
- ✦ (Optional) `OVERRIDE-FILE.yaml` is the name of a file in the overrides directory that overrides the chart's default `values.yaml` file.

The following table describes the Helm chart configurations in `tsmgr.yaml`:

Key	Values	Description
<code>scope</code>	<code>cluster</code> or <code>namespace</code>	<p>If you use <code>cluster</code>: The chart is only deployed once on a cluster. Subsequent service instances created on the same Kubernetes cluster do not cause a cluster-scoped chart to be deployed additional times. Use this scope for operators and controllers that manage multiple instances from a single deployment.</p> <p>If you use <code>namespace</code>: The chart is deployed on the cluster every time the service is created.</p>
<code>offered</code>	<code>true</code> or <code>false</code>	<p>If you use <code>true</code>: The <code>scope</code> must be set to <code>namespace</code>. Only one chart can be <code>true</code>. The offered chart represents the service the user binds to. However, you can expose more resources with <code>cf bind-service</code>.</p> <p>If you use <code>false</code>: The chart is not offered.</p>

(Optional) Override Chart Values

Within an `OVERRIDE-FILE.yaml` file, you can specify any values that you want to apply across all plans that differ from a chart's default `values.yaml`.

To override chart values:

1. Create an `overrides` directory and an `overrides/OVERRIDE-FILE.yaml` file.

Ensure that the properties you add to `OVERRIDE-FILE.yaml` match the properties in `values.yaml`.

For some common scenarios, see [Override Chart Values](#) for some common scenarios.

(Optional) Define Plans Configuration



Note: The properties in `plans.yaml` and `PLAN-FILE.yaml` files override configurations in override files. For information about override files, see [Override Chart Values](#).

In `cf marketplace`, each service has a set of plans that developers can provision. A plan is a template for service instances. For example, different plans might represent a large or a small

instance of a database.

For Kubernetes services, each TSMGR plan represents a set of values overriding the default values in the Helm chart. These values are configured in `plans.yaml`. For examples of `plans.yaml` files, see [tsmgr-sample](#) on GitHub.

To override the default settings in `values.yaml` and offer custom plans:

1. Create and edit `plans/PLAN-FILE.yaml` with the Helm values that you want to override. `PLAN-FILE.yaml` must consist of only lowercase letters, digits, `.`, or `-`.

Ensure that the properties you add to `PLAN-FILE.yaml` match the properties in `values.yaml` and that you only change the values.

For example:

```
---
resources:
  requests:
    memory: 128Mi
    cpu: 100m
```

2. (Optional) Create and edit `resource-quotas/RESOURCE-QUOTA-FILE.yaml` with the `ResourceQuota` definition you want to use. This `ResourceQuota` is created in each Kubernetes namespace where a service instance is provisioned. For more information about `ResourceQuota` objects, see the [Kubernetes documentation](#).

For example:

```
---
apiVersion: v1
kind: ResourceQuota
metadata:
  name: mem-cpu-demo
spec:
  hard:
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

3. Edit your `plans.yaml` file to add your custom plan information using the following template:

```
- name: PLAN-NAME
  description: PLAN-DESCRIPTION
  file: PLAN-FILE
  clusterName: CLUSTER-NAME
  resourceQuotaPath: RESOURCE-QUOTA-FILE
```

Where:

- `PLAN-NAME` is the name of the plan that developers see when they run `cf marketplace`. The plan name is listed under `plans`. Ensure that the plan name is lowercase only and contains no special characters.

- ✦ `PLAN-DESCRIPTION` is the description of the plan that developers see when they run `cf marketplace`. The plan description is listed under `description`.
- ✦ `PLAN-FILE` is the name of the plan file in the `plans` subdirectory.
- ✦ (Optional) `CLUSTER-NAME` is the name of a cluster that is registered with TSMGR. For more information, see [Register a Kubernetes Cluster](#). If omitted, service instances are provisioned to the default cluster. For more information, see [Set a Default Cluster](#).
- ✦ (Optional) `RESOURCE-QUOTA-FILE` is name of the resource quota file for the plan.

For example:

```
- name: "small"
  description: "default (small) plan for mysql"
  file: "small.yaml"
  resourceQuotaPath: "quota.yaml"
- name: "medium"
  description: "medium sized plan for mysql"
  file: "medium.yaml"
  clusterName: "medium-cluster"
```



Note: Developers can specify service-specific configuration parameters to override plan and Helm chart values. They pass these parameters using the `-c` flag when they provision or update a service instance through the cf CLI. The `-c` flag accepts a JSON object.

The JSON configuration values are mapped directly onto the chart values and take precedence over any chart default and plan values. For information about service-specific configuration parameters, see [Arbitrary Parameters](#).

(Optional) Offer Multiple Charts in a Single Offering

TSMGR enables you to offer multiple Helm charts in a single service offering. You might want to offer multiple Helm charts in the following cases:

- A cluster-scoped operator or controller where the individual charts are custom resources
- A namespace-scoped operator chart with another chart that represents the custom resource
- A service that uses multiple charts. For example, a service that uses a data service chart and a monitoring service chart.

To offer a service with multiple Helm charts, you must edit the `tsmgr.yaml` file. The `charts` section in a `tsmgr.yaml` file is an ordered list of the charts that are included in the service offering.

For the full details of this file, see [\(Optional\) Create tsmgr.yaml](#).

(Optional) Create Binding Template for App Consumption

Apps running in TAS for VMs gain access to bound service instances through an environment variable credentials hash called `VCAP_SERVICES`. Libraries such as Spring Cloud Cloud Foundry Connector automatically support services if their credentials in `VCAP_SERVICES` are formatted in a valid

JSON object.

For an example of the requirements for using Spring Cloud Cloud Foundry Connector with a MySQL service, see the [Spring Cloud Cloud Foundry Connector documentation](#).

Chart authors can edit the `bind.yaml` file to enable the broker to return a valid JSON object containing service credentials. This `bind.yaml` file contains a Jsonnet template. When a developer binds the service instance to their app, the service credentials are delivered to `VCAP_SERVICES` for app consumption.

For information about Jsonnet templates, see [Jsonnet](#).

To edit and test the `bind.yaml` file:

1. Edit the `bind.yaml` file.

For example, the following `bind.yaml` enables the broker to return MySQL credentials in a valid JSON object:

```
template: |
{
  hostname: $.services[0].status.loadBalancer.ingress[0].ip,
  name: $.services[0].name,
  jdbcUrl: "jdbc:mysql://" + self.hostname + "/my_db?user=" + self.username +
  "&password=" + self.password + "&useSSL=false",
  uri: "mysql://" + self.username + ":" + self.password + "@" + self.hostname
  + ":" + self.port + "/my_db?reconnect=true",
  password: $.secrets[0].data['mysql-root-password'],
  port: 3306,
  username: "root"
}
```



Note: The values of `$.services`, `$.secrets` and `$.ingresses` are from the namespace where the chart is deployed.

2. Test if the `bind.yaml` template creates the expected JSON object:
 1. Ensure that you have registered a cluster. See [Managing Clusters](#).
 2. Ensure that your `kubectl` points to the registered cluster.
 3. Install the Helm chart that you are including in the service offering. Wait for the service to be ready and for the public IP address to be available:

```
kubectl create namespace NAMESPACE
helm install RELEASE-NAME HELM-CHART.tgz -n NAMESPACE --set service.type=
LoadBalancer --wait
```

Where:

- `RELEASE-NAME` is the name of the release instance you are creating.
- `HELM-CHART` is the Helm chart filename.

- **NAMESPACE** is the name of the namespace in which you are installing your Helm chart.

For example:

```
kubectl create namespace mysql-ns
helm install mysql mysql-0.1.4.tgz -n mysql-ns --set service.type=LoadBalancer --wait
```

4. Execute the **test-bind-template** by running:

```
tsmgr test-bind-template CLUSTER-NAME NAMESPACE bind.yaml
```

Where:

- **CLUSTER-NAME** is the name of the TSMGR-registered cluster where your chart is installed.
- **NAMESPACE** is the name of the namespace in which you installed your Helm chart.

For example:

```
$ tsmgr test-bind-template my-cluster mysql-ns ./bind.yaml
Services, secrets, and ingress:
{
  "secrets": [
    {
      "data": {
        "mysql-password": "aBcDefG1HIJ",
        "mysql-root-password": "kLmNoP2qrSt3"
      },
      "name": "1-23456789-mysql"
    },
  ],
  "services": [
    {
      "metadata": {
        "name": "1-23456789-mysql",
        "namespace": "tsmgr-f5d531d5-7b07-4bc0-8114-8aba5e2bd706",
        "uid": "9f2d94c1-4900-11ea-a03a-42010a800053",
        "resourceVersion": "4943530",
        "creationTimestamp": "2020-02-06T16:49:20Z",
        "labels": {
          "app": "1-23456789-mysql",
          "heritage": "Helm",
          "release": "1-23456789"
        }
      },
      "name": "1-23456789-mysql",
      "spec": {
        "ports": [
          {
            "name": "mysql",
            "protocol": "TCP",
            "port": 3306,
            "targetPort": "mysql",
```

```

        "nodePort": 32257
      },
    ],
    "selector": {
      "app": "1-23456789-mysql"
    },
    "clusterIP": "10.24.2.216",
    "type": "LoadBalancer",
    "sessionAffinity": "None",
    "externalTrafficPolicy": "Cluster"
  },
  "status": {
    "loadBalancer": {
      "ingress": [
        {
          "ip": "33.44.55.66"
        }
      ]
    }
  }
}

}

]

}

Rendered template {
  "hostname": "33.44.55.66",
  "jdbcUrl": "jdbc:mysql://33.44.55.66/my_db?user=root&password=kLmNoP
2qrSt3&useSSL=false",
  "name": "1-23456789-mysql",
  "password": "kLmNoP2qrSt3",
  "port": 3306,
  "uri": "mysql://root:kLmNoP2qrSt3@33.44.55.66:3306/my_db?reconnect=t
rue",
  "username": "root"
}

```

5. Examine the output. The first part shows the available secrets and services information. The `Rendered template` section shows how the `bind.yaml` template rendered the data.

Load Container Images into a Private Container Registry

VMware recommends using a private container image registry in production. TSMGR can then modify your Helm charts to point to images in the private container registry.

If either of the following applies, you must configure a private container registry:

- Your environment is air-gapped and cannot connect to public container registries such as Docker Hub or Quay. For information about commonly used public container registries, see [Docker Hub](#), [Harbor](#), and [Quay](#).
- The images referenced in the Helm chart are not public.

If you do not load your container images in to a private container registry, a developer cannot create a service instance offered on TSMGR.

To load your container images:

1. Load your container images by following the instructions for your specific container registry.

For example, if you use Harbor, follow the instructions in [Pulling and pushing images using Docker client](#) in GitHub.

Ensure Your Images are Accessible

When you provision a service, the image paths in your Helm charts need to be accurate so that Kubernetes can pull the images from the correct repository.

To configure the image paths:

1. Search your service's Helm chart `values.yaml` file for `global.imageRegistry`:
 - ✦ If `global.imageRegistry` is set in the chart's `values.yaml`, no further action is required.
 - ✦ If `global.imageRegistry` is supported and unset in the chart's `values.yaml`, set `imageCredentialsForServiceInstances.registry` in the TSMGR Helm chart. TSMGR pulls the images from the registry at `imageCredentialsForServiceInstances.registry`.
 - ✦ If your chart does not support the `global.imageRegistry` pattern, edit `overrides.yaml` to override the image fields to point to the private registry where the images are hosted.

Charts often contain multiple images. For example, the chart values excerpt below sets values for both the `mysql` and `mysqld-exporter` images:

```
...
image: "mysql"
imageTag: "5.7.30"
...
metrics:
  enabled: false
  image: prom/mysqld-exporter
  imageTag: v0.10.0
  imagePullPolicy: IfNotPresent
```

If the registry path were `registry.example.com/my-project`, then `overrides.yaml` must contain:

```
image: "registry.example.com/my-project/mysql"
metrics:
  image: registry.example.com/my-project/prom/mysqld-exporter
```

To edit `overrides.yaml`, follow the procedure in [Override Chart Values](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Configuring a Helm Chart



Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

This topic describes how to configure a Helm chart for Tanzu Service Manager (TSMGR).

Overview

TSMGR enables platform operators to offer Helm charts as on-demand services to developers on VMware Tanzu Application Service for VMs (TAS for VMs). TSMGR uses the description in `Chart.yaml` as the description of the offer in `cf marketplace`.

TSMGR surfaces all values specified in the `values.schema.json` file to the developer at the time of provision through Apps Manager. For more information about schema files, see the [Helm documentation](#). Given that the `cf` CLI does not enforce the schema during provision, developers can also override the Helm chart's values file by using the `-c` flag when they provision a service instance. For more information, see [Provision a Service Instance](#).

Before you add a service offering to `cf marketplace`, you must configure your Helm charts to be compatible with TSMGR. For examples of TSMGR-compatible charts, see [tsmgr-sample](#) on GitHub.

To configure a Helm chart to be compatible with TSMGR:

1. [Override Chart Values](#)
2. [Define Custom Resource Definitions](#)



Note: Helm 3 handles CRDs differently from other Kubernetes Resources. For information about best practices for using CRDs, see the [Helm documentation](#).

3. [Prevent Secret Regeneration](#)
4. [Configure Template Values](#)

VMware recommends that you read the following resources when creating your Helm chart:

- For information about creating your first Helm chart, see the [Bitnami documentation](#).
- For information about best practices for creating production-ready Helm charts, see the [Bitnami documentation](#).
- For more information about best practices, see the [Helm documentation](#).

Override Chart Values

To ensure that your chart is compatible with TSMGR, you might need to override specific values in `values.yaml`.

You must follow the procedure in this section if your chart uses any of the following properties:

- `storageClass`
- `services.type`
- `hosts.name`

You can choose to override other properties defined in the chart `values.yaml` by adding those properties to an `OVERRIDE-FILE.yaml` file.



Warning: You cannot change the chart name or resize a PersistentVolumeClaim after you add the service offering to TSMGR.

Below are some common scenarios where overriding chart values is required.

1. Edit your override file to configure storage. Edit the `storageClass:` value to correspond to the StorageClass name for your configured Tanzu Kubernetes Grid Integrated Edition cluster. The default name for TKGI is `standard`. For more information, see [Configuring and Using PersistentVolumes](#).

For example:

```
...
persistence:
  enabled: true
  storageClass: "standard"
...
```

2. Edit your override file to enable service access. TSMGR supports using a load balancer, Ingress, or ExternalDNS to access services.

- ✦ **If you want your service to be accessed directly using a load balancer,** set `services.type:` to `LoadBalancer`. For example:

```
...
service:
  type: LoadBalancer
  port: 3306
...
```

- ✦ **If you want your service to be accessed using Ingress or ExternalDNS:** Edit the value of the `hosts.name` to include a templated value for uniqueness. For example:

```
...
ingress:
  enabled: true
  hosts:
    - name: "{{ .Release.Name }}.example.com"
```


...

For more information, see [\(Optional\) Template Values](#).



Note: The Ingress controller or ExternalDNS controller needs to be installed and configured manually.

For information about Ingress, see the [Kubernetes documentation](#). For information about ExternalDNS, see [external-dns](#) in GitHub.

3. Edit the `tsmgr.yaml` file to use the override file you created earlier.

For example:

```
offer-name: OFFER-NAME
charts:
  - chart: CHART-NAME
    version: CHART-VERSION-NUMBER
    offered: OFFERED-VALUE
    scope: SCOPE-VALUE
    overrideValuesFile: OVERRIDE-FILE
```

For information about the properties used in `tsmgr.yaml`, see [\(Optional\) Create tsmgr.yaml](#).

Define Custom Resource Definitions

TSMGR supports charts that use the Helm 2 and Helm 3 CLIs. Helm 2 and Helm 3 use different methods to install CRDs.

If you are using a Helm 2 chart, use both the Helm 2 CLI and Helm 3 CLI methods. If you are using a Helm 3 chart, use the Helm 3 CLI method only.

To define CRDs:

1. Define your CRDs using the Helm 3 method by referring to the [Helm 3 documentation](#). Helm 3 uses a `crds` directory to install CRDs.
2. If you are using a Helm 2 chart, define your CRDs using the Helm 2 method by referring to the [Helm 2 documentation](#). Helm 2 uses a hook annotation to install CRDs.

Prevent Secret Regeneration

When Helm charts use secret patterns, the secrets regenerate on upgrade by default. Many services do not support secret regeneration.

VMware recommends configuring your Helm chart to prevent secret regeneration where it defines secret patterns.

To prevent secret regeneration:

1. Search your Helm chart for a line containing `randAlphaNum`. For example:

```
mysql-root-password: {{ randAlphaNum 10 | b64enc | quote }}
```

2. If a file contains `randAlphaNum`, add the following in the same file:

```
metadata:
  annotations:
    "helm.sh/hook": "pre-install"
    "helm.sh/hook-delete-policy": "before-hook-creation"
```

For more information about this issue, see [Random passwords change on upgrade](#) in GitHub.

Configure Template Values

If you know that you do not need, or want, to configure template values then you can skip this section.

Helm uses the literal contents of a `values.yaml` file as the concrete set of substitutions when creating a release. In the context of TSMGR, this is the contents of the plan file and the override file.

TSMGR uses these values as a template for multiple releases, which can create cases where values need to go through the Helm templating engine before any template files in the `/templates` directory are rendered with those values.

There are various cases where the contents of the plan file and the override file might need to go through the Helm templating engine. Such cases include when a chart:

- Requires a unique name in addition to the release name.
- Uses Ingress controller. For more information about Ingress, see the [Kubernetes documentation](#).

For example, in a chart with ingress hosts you can, if necessary, edit the plan file or override file to use the release name as the unique subdomain discriminator in the ingress definition, as seen in the snippet below:

```
...
ingress:
  enabled: true
  hosts:
    - name: "{{ .Release.Name }}.example.com"
  ...
```

To configure the override file and plan file:

1. Follow the procedure in [Override Chart Values](#).
2. Follow the procedure in [Define Plans Configuration](#).

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Troubleshooting Errors



Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

This topic provides platform operators with solutions for specific errors that might occur in Tanzu Service Manager (TSMGR).

Troubleshoot Errors

Incompatible Helm CLI Versions

Symptom	<p>When you run <code>tsmgr offer save</code>, you see an error similar to the following:</p> <pre>helm.sh/hook: crd-install found in [Chart1, Chart2....] but no Helm /crds dir; this looks like a Helm 2 only chart</pre>
Cause	<p>TSMGR supports charts that use the Helm 2 and Helm 3 CLIs. However, Helm 2 and Helm 3 use different methods to install Custom Resource Definitions (CRDs).</p> <p>If a Helm 2 chart does not define CRDs using both methods, adding or updating the service offering might fail.</p>
Solution	<p>Define the CRDs in your Helm chart using both the Helm 2 and Helm 3 methods by doing the procedure in Define Custom Resource Definitions.</p>

[Create a pull request or raise an issue on the source for this page in GitHub](#)

Techniques for Troubleshooting



Warning: Tanzu Service Manager has reached End of General Support (EOGS) and has been withdrawn.

Page last updated:

This topic provides platform operators with techniques for troubleshooting Tanzu Service Manager (TSMGR).

Check CLI and Server Versions

If the version for the TSMGR CLI and server are not the same, errors can occur.

To verify that the TSMGR CLI and server versions match:

1. Verify the TSMGR CLI and server versions by running:

```
tsmgr version -t http://tsmgr.SYSTEM-DOMAIN
```

Where **SYSTEM-DOMAIN** is your VMware Tanzu Application Service for VMs system domain URL.

For example:

```
$ tsmgr version -t http://tsmgr.my-env.com
Client Version [0.6.40]
Server Version [0.6.40]
```

The value of **Client Version** is the TSMGR CLI version.

The value of **Server Version** is the TSMGR server version.

2. If the versions are not the same, do one of the following:
 - ♦ **If you want to upgrade the TSMGR CLI**, follow the procedure in [Install the TSMGR CLI](#).
 - ♦ **If you want to upgrade the TSMGR server**, follow the procedure in [Upgrading Tanzu Service Manager](#).

Run Helm Commands

Platform operators can use the Helm and kubectl CLIs for advanced debugging.

To manually run **helm** commands:

1. Target your Kubernetes cluster by following the procedure in [Retrieving Cluster Credentials and Configuration](#).
2. You can run `helm` commands against the cluster targeted in the previous step. For example:

```
helm list --all-namespaces
```

For more information about Helm commands, see [Helm](#) in the Helm documentation.

Clean Up a Service Instance

VMware Tanzu Application Service for VMs (TAS for VMs) gives an asynchronously provisioned service instance seven days to become healthy. In some failure modes, you might have to manually clean up services earlier.

You can use the cf CLI to purge a service instance from VMware Tanzu Application Service for VMs (TAS for VMs). However, this can leave resources in the Kubernetes cluster. To fully clean up a service instance, you must delete any remaining namespaces.

For information about purging a service instance, see [Purge a Service Instance](#) in the Cloud Foundry documentation.

To delete Helm releases and namespaces in a Kubernetes cluster:

1. Retrieve the names of the remaining namespaces by running:

```
kubectl get namespaces
```

2. Record the namespaces that are used by the service instances deployed by TSMGR.

The namespaces are named `tsmgr-GUID`, where `GUID` is the `service-instance-guid` for the service instance. You can view the `service-instance-guid` for a service instance by running `cf service INSTANCE-NAME --guid`.

3. For each namespace, retrieve the Helm release names by running:

```
helm ls --namespace=NAMESPACE
```

Where `NAMESPACE` is the name of the namespace you are deleting.

Record the value in the `NAME` column. This is the release name.



Note: If your service offering has multiple charts, the above command might list multiple releases.

4. For each namespace and associated Helm release, delete the Helm release by running:

```
helm delete RELEASE --namespace=NAMESPACE
```

Where:

- ◆ `NAMESPACE` is the name of the namespace for the Helm release you are deleting.

- `RELEASE` is the name of the Helm release you are deleting.

- For each namespace, delete the namespace by running:

```
kubectl delete namespace NAMESPACE
```

Where `NAMESPACE` is the name of the namespace you are deleting.

View Service Instance Metadata in Kubernetes

When a developer provisions a service instance onto a Kubernetes cluster, Tanzu Service Manager creates a Custom Resource for the service instance in the cluster. Platform operators can use this Custom Resource to identify which TAS for VMs apps or services depend on a resource in the Kubernetes cluster.

To view the Custom Resource:

- Retrieve the GUID for the service instance by running:

```
cf service SERVICE-INSTANCE --guid
```

Record the output.

- View the Custom Resource by running: `kubectl get instance GUID -n TSMGR-GUID -o yaml`
Where `GUID` is the output you recorded above.

For example:

```
$ kubectl get instance GUID -n TSMGR-GUID -o yaml

apiVersion: tsm.vmware.com/v1alpha1
kind: Instance
metadata:
  generation: 1
  labels:
    clusterName: cluster-a
    offerName: mysql
    offerVersion: "1"
  name: GUID
  namespace: TSMGR-GUID
spec:
  chartVersions:
    mysql: 1.6.6
  clusterName: cluster-a
  configParams:
    name: mydb
    namespace: TSMGR-GUID
  instanceID: GUID
  instanceName: mydb
  offerName: mysql
  offerVersion: 1
  orgID: "a12345b6-cf17-4109-aefb-0510d09d1374"
  orgName: myorg
  spaceID: "zyx9u8765-e6c2-4a52-861f-1b0663cf6524"
  spaceName: myspace
```

View TSMGR Logs

You can view TSMGR server logs to troubleshoot TSMGR by running the `kubectl logs` command.

To view the TSMGR server logs:

1. To record the service you want to verify, view your TSMGR services by running:

```
kubectl get services --namespace YOUR-NAMESPACE
```

For example:

```
$ kubectl get services --namespace tsmgr-test
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
tsmgr-test-release-chartmuseum	ClusterIP	XXX.XXX.5.87	
tsmgr-test-release-tsmgr-broker	LoadBalancer	XXX.XXX.0.197	XXX.XXX.16.30
tsmgr-test-release-tsmgr-daemon	LoadBalancer	XXX.XXX.10.29	XXX.XXX.23.47

2. View TSMGR server logs by running:

```
kubectl logs --namespace YOUR-NAMESPACE \
service/SERVICE-TO-VERIFY --all-containers
```

For example:

```
$ kubectl logs --namespace tsmgr-test service/tsmgr-test-release-chartmuseum --all-containers
```

```
{
  "L": "INFO",
  "T": "2020-07-23T16:19:48.640Z",
  "M": "Starting ChartMuseum",
  "port": 80
}
{
  "L": "INFO",
  "T": "2020-07-23T16:21:16.740Z",
  "M": "[19] Request served",
  "path": "/api/charts",
  "comment": "",
  "latency": "4.530608ms",
  "clientIP": "XXX.XXX.2.73",
  "method": "GET",
  "statusCode": 200,
  "reqID": "1c38d89c-63bc-4c90-bb18-89edeb780ac1"
}
{
  "L": "INFO",
  "T": "2020-07-23T16:21:16.785Z",
  "M": "[20] Request served",
  "path": "/api/charts",
  "comment": "",
  "latency": "34.642387ms",
  "clientIP": "XXX.XXX.2.73",
  "method": "POST",
  "statusCode": 201,
  "reqID": "d0c77f8d-6485-486f-8232-9dfe9a425c61"
}
{
  "L": "INFO",
  "T": "2020-07-23T16:21:18.097Z",
  "M": "[21] Request served",
  "path": "/api/charts/minio/5.0.11",
  "comment": "",
  "latency": "8.332745ms",
  "clientIP": "XXX.XXX.1.26",
  "method": "GET",
  "statusCode": 200,
  "reqID": "6965f38a-06d3-4746-9327-57c2ca3190e6"
}
{
  "L": "INFO",
  "T": "2020-07-23T16:21:18.106Z",
  "M": "[22] Request served",
  "path": "/charts/minio-5.0.11.tgz",
  "comment": "",
  "latency": "6.144315ms",
  "clientIP": "XXX.XXX.1.26",
  "method": "GET",
  "statusCode": 200,
  "reqID": "92e9af1a-a9ca-40db-bfb0-3f069deda036"
}
{
  "L": "INFO",
  "T": "2020-07-23T16:23:18.323Z",
  "M": "[47] Request served",
  "path": "/api/charts",
  "comment": "",
  "latency": "29.234962ms",
  "clientIP": "XXX.XXX.2.73",
  "method": "GET",
  "statusCode": 200,
  "reqID": "711b9b4e-c626-4366-8861-e4e8443dc715"
}
{
  "L": "INFO",
  "T": "2020-07-23T16:23:18.367Z",
  "M": "[48] Request served",
  "path": "/api/charts",
  "comment": "",
  "latency": "18.952619ms",
  "clientIP": "XXX.XXX.2.73",
  "method": "POST",
  "statusCode": 201,
  "reqID": "b4465eca-4ac0-4fa8-ae8a-bdd91792c428"
}
```

Modify Timeouts

You can modify the following timeouts:

- [Modify the Provisioning Timeout](#)
- [Modify the Cloud Foundry Broker Registration Timeout](#)

Modify the Provisioning Timeout

When TSMGR provisions an offer, by default, it allows 20 minutes for the Kubernetes resources for each Helm chart to become available. If any charts take longer than 20 minutes, TSMGR fails the provision.

To modify the provisioning timeout:

1. Configure the `broker.serviceInstanceTimeout` property in the `values-production.yaml` file.

```
broker:
  serviceInstanceTimeout: TIMEOUT
```

Where `TIMEOUT` is the length of the timeout including a unit for time, for example, `25m`

2. Apply the change to the timeout by running:

```
helm upgrade RELEASE-NAME -n TSMGR-NAMESPACE -f values-production.yaml
```

Where:

- ♦ `RELEASE-NAME` is the name of the release.
- ♦ `TSMGR-NAMESPACE` is the namespace of the release.

Modify the Cloud Foundry Broker Registration Timeout

TSMGR attempts to register with Cloud Foundry when it starts. If it cannot register, it re-attempts to register four times. By default, each attempt to register is 20 seconds with a 5-second interval between each attempt. After four attempts, if it is still failing, the Pod crashes.

A reason Cloud Foundry registration can fail is if Kubernetes is not yet serving Pod traffic.

To modify the Cloud Foundry broker registration timeout:

1. Configure the `broker.initialBrokerCFRegisterTimeout` and `broker.initialBrokerCFRegisterInterval` properties in the `values-production.yaml` file.

```
broker:
  initialBrokerCFRegisterTimeout: TIMEOUT
  initialBrokerCFRegisterInterval: INTERVAL
```

Where:

- ♦ `TIMEOUT` is the length of the timeout including a unit for time, for example, `30s`.
- ♦ `INTERVAL` is the interval between attempts to register including a unit for time, for example, `10s`.

2. Apply the change to the timeout by running:

```
helm upgrade RELEASE-NAME -n TSMGR-NAMESPACE -f values-production.yaml
```

Where:

- `RELEASE-NAME` is the name of the release.
- `TSMGR-NAMESPACE` is the namespace of the release.

[Create a pull request or raise an issue on the source for this page in GitHub](#)