Oracle® Communications Solution Test Automation Platform

Deployment Guide

Release 1.26.1.0.0
G48767-01
April 2026

ORACLE

Oracle Communications Solution Test Automation Platform Deployment Guide, Release 1.26.1.0.0

G48767-01

Contents

About This Content

This document describes how to implement and use Oracle Communications Solution Testing Automation Platform.

Audience

This document is intended for anyone who installs, configures, administers, or customizes Solution Testing Automation Platform.

1

Overview of STAP Deployment

Learn about Oracle Communications Solution Test Automation Platform (STAP) deployment.

> **Note**
>
> STAP is a cloud native application.

Topics in this document:

Overview of STAP Deployment

Oracle Communications Solution Test Automation Platform (STAP) supports a Kubernetes-orchestrated containerized microservice architecture to facilitate continuous integration, continuous delivery, and DevOps practices. This allows you to harness the benefits of the cloud with STAP's services.

STAP cloud-native offers these key features:

- Kubernetes orchestrates container images (Docker, CRI-O), providing production support for STAP deployment.
- Helm charts simplify installation and management.
- Images and scripts facilitate development and testing.
- A containerized microservice architecture includes the following three essential services:
  - STAP Test Data Service (TDS)
  - STAP Test Execution Service (TES)
  - STAP UI

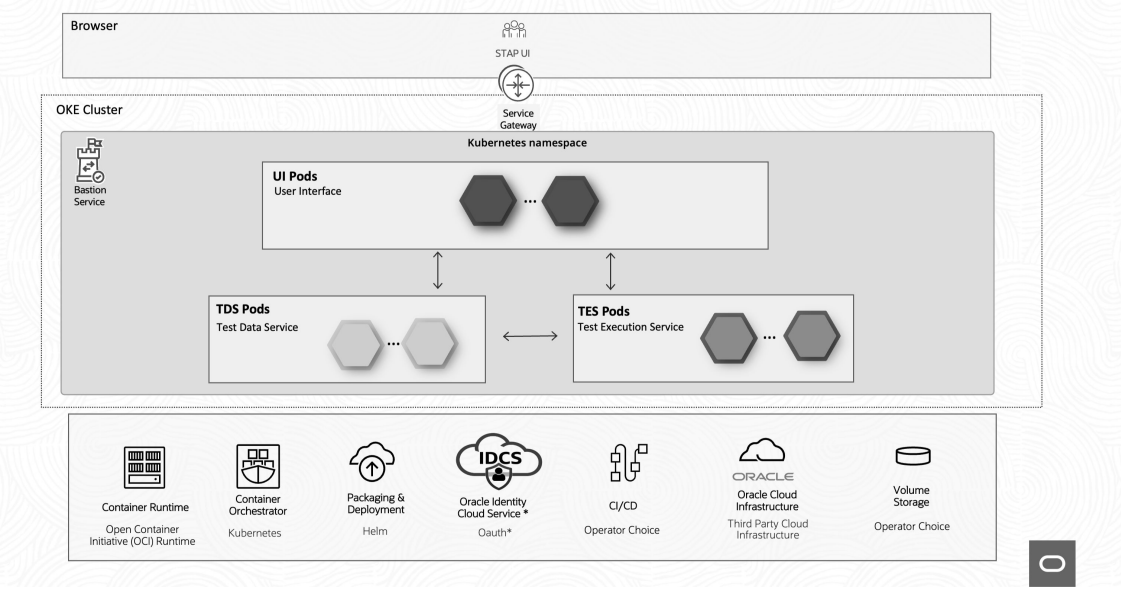
STAP Deployment Architecture

Figure 1-1 describes a high-level conceptual architecture for STAP deployment in a single OCI region. You use this architectural blueprint as a starting point to leverage the benefits of OCI. Your specific deployment topology will depend on multiple factors specific to the operator's requirements, and may differ from this architecture.

In this conceptual reference architecture, STAP is deployed on the Oracle Cloud Infrastructure Kubernetes Engine (OKE). The STAP Cloud Native Toolkit (CNTK) contains artifacts to deploy all three microservices: TDS, TES, and the STAP UI. The OKE Persistent Volumes (PVs) use NFS-based persistence for shared storage. Images shipped in the CNTK are used alongside the shipped helm charts to deploy each individual microservice. TDS, TES, and the UI microservices communicate with each other using Kubernetes services and are exposed to the operator via the STAP UI.

STAP supports Basic and OAuth Authentication. For OAuth, STAP supports Oracle Cloud Identity Service.

Figure 1-1 STAP Deployment Architecture



This figure shows three microservice deployments: TDS, TES, and UI.

- The TDS microservice manages the data used in STAP, storing test case data, test results, and other critical testing information. It uses an embedded database to persist all data.
- The TES microservice is stateless and runs all test cases.
- The UI microservice offers a web-based interface for interacting with the STAP application.

Each microservice communicates using Kubernetes services, which can be encrypted with TLS.

2

Overview of the STAP Cloud Native Deployment Toolkit

Learn about the Oracle Communications Solution Test Automation Platform (STAP) Cloud Native Deployment Toolkit (CNTK) that assists in deploying and managing STAP services in Kubernetes.

Topics in this document:

STAP Cloud Native Deployment Toolkit Components

The STAP CNTK is an archive file which provides default configuration files and scripts for deploying STAP.

The toolkit includes the following:

- Ready-to-use Helm charts to help you orchestrate containers in Kubernetes.
- Configuration files (**values.yaml**) to configure and manage STAP microservices: Test Data Service, Test Execution Service, and STAP UI.
- Ready-to-use scripts to create and manage secrets for STAP microservices.
- Images containing STAP microservices.

About the STAP Cloud Native Deployment Toolkit Artifacts

Each of the microservices, TDS, TES, and UI, has the following artifacts in the CNTK:

- **Image:** Use the tarred image to deploy the respective STAP microservice in a Kubernetes cloud native deployment.
- **Helm Chart:** Use Helm charts to orchestrate the deployment of the respective STAP microservice.
- **Scripts:** Use scripts to generate Kubernetes secrets, which securely encode and mount sensitive credentials and files to the respective STAP microservice.

3
Getting Started with STAP Deployment

Learn about getting started with your Oracle Communications Solution Test Automation Platform (STAP) deployment.

> **Note**
>
> STAP can be used for testing in a lab environment and is licensed to be used only on test or lab platforms and environments.

Topics in this document:

About Configuring and Deploying STAP

You install the STAP deployment package by configuring and deploying its Helm charts. The Helm charts include YAML template descriptors for all Kubernetes resources and a **values.yaml** file that provides default configuration values for each chart.

Create a copy of **values.yaml** and rename it to **override-values.yaml**. You set custom values for your environment in this file, rather than updating the original **values.yaml** file.

Installing a Helm chart generates valid Kubernetes manifest files by replacing default values from the **values.yaml** file with custom values from your **override-values.yaml** file, and creates Kubernetes resources. Helm calls this a new release. You use the release name to track and maintain this installation.

> **Note**
>
> The deployment process automatically detects and applies proxy configuration specified in the override-values.yaml file. If you need to change the proxy settings, update the proxy parameters in override-values.yaml and perform a rolling restart of the Kubernetes deployment. This action ensures all pods receive and use the updated configuration.

The STAP CNTK includes the Helm charts in Table 3-1.

Table 3-1 STAP CNTK Helm Charts

Helm Chart	Description	Notes
stap-comms-tds-chart	This chart is used to deploy the STAP TDS microservice.	Requires NFS and IDCS configuration (If you are using OAuth type authorization). **Note:** Ensure the NFS db folder is empty before deploying.
stap-comms-tes-chart	This chart is used to deploy STAP TES microservice.	Requires IDCS configuration (If you are using OAuth type authorization). **Note:** Deploy TES after deploying TDS.
stap-comms-ui-chart	This chart is used to deploy STAP UI microservice.	Requires IDCS configuration (If you are using type authorization). **Note:** Deploy the STAP UI after deploying TDS and TES.

High-Level Installation Tasks

You install STAP on your system by performing these high-level tasks:

1. Install all prerequisite software for your STAP cloud native environment.
   See "Setting Up Prerequisite Software".
2. Prepare your deployment environment by downloading the STAP Cloud Native Deployment Toolkit, extracting the Helm charts, and loading the STAP component images.
   See "Installing STAP".
3. Configure and deploy the TDS microservice in your cloud native environment.
   See "Installing TDS".
4. Configure and deploy the TES microservice in your cloud native environment.
   See "Installing TES".
5. Configure and deploy the UI in your cloud native environment.
   See "Installing the UI".

4

Setting Up Prerequisite Software

Learn about prerequisite tasks to perform before installing the Oracle Communications Solution Test Automation Platform (STAP) deployment toolkit, such as installing and configuring third-party software.

Topics in this document:

STAP Prerequisite Tasks

As part of preparing your environment for STAP, you choose, install, and set up various components and services in ways that are best suited for your environment.

The high-level prerequisite tasks for STAP are:

1. Ensure you have downloaded the correct versions of the third-party tools. See "Common Software Compatibility" in *STAP Compatibility Matrix* for information about the compatible versions.
2. Install Kubernetes and create a cluster. See "Creating a Kubernetes Cluster".
3. Install Podman and a container runtime supported by Kubernetes. See "Installing Podman".
4. Install Helm. See "Installing Helm".

Prepare your environment with these technologies installed, configured, and tuned for performance, networking, security, and high availability.

Creating a Kubernetes Cluster

Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications. It groups containers into logical units for easy management and discovery. When you deploy Kubernetes, you get a physical cluster with machines called nodes. A reliable cluster must have multiple worker nodes spread over separate physical components, and a very reliable cluster must have multiple primary nodes spread over separate physical components.

To prepare and secure the Kubernetes environment for STAP, follow the steps below:

1. Set up a Kubernetes cluster for your STAP deployment.
2. Secure access to the cluster and its objects by using service accounts.
3. Configure authentication and authorization modules.

For more information about Kubernetes, see the Kubernetes documentation:

https://kubernetes.io/docs/concepts/

Installing Podman

You use the Podman platform to containerize STAP. Install Podman to use the prebuilt images provided with the STAP cloud native deployment package.

To install Podman, refer to the Podman documentation:

https://podman.io/

You can use Podman or any container runtime that supports the Open Container Initiative if it supports the Kubernetes version specified in Compatibility Matrix.

Installing Helm

Helm is a package manager that helps you install and maintain software on a Kubernetes system. In Helm, a package is called a chart, which consists of YAML files and templates rendered into Kubernetes manifest files. The STAP deployment package includes Helm charts that help create Kubernetes objects with a single command.

To install Helm and configure access, follow the steps below:

1. Install Helm. See the information and downloads on the Helm website: https://github.com/helm/helm/releases
2. Verify supported versions by checking the *Compatibility Matrix*.
3. Configure cluster access:
   a. Helm uses a configuration file to enable the helm command to access the Kubernetes cluster. By default, Helm reads this file from **$HOME/.kube/config**
   b. To use a different file, set the **$KUBECONFIG** environment variable to the required location.
4. Ensure appropriate permissions:
   a. Helm inherits the permissions configured for cluster access.
   b. If role-based access control (RBAC) is configured, you must grant Helm users sufficient cluster permissions.

Creating a STAP Application in Oracle Identity Cloud Service

> **Note**
>
> Create this only if you use OAuth authentication.

When you create a confidential OAuth application in Oracle Identity Cloud Service (IDCS), it provides you with a client ID and client secret. Your client will need the client ID and client secret to request OAuth access tokens for accessing STAP.

To create a confidential OAuth application in IDCS:

1. Log in to the IDCS Admin Console.

2. Create a new application by selecting **Add** and choosing **Confidential Application**.
3. Under application URL, enter **https:**//*STAP-UI:PORT/* (ensure the trailing slash is included) where:
   - *STAP-UI* is the UI host
   - *PORT* is the UI port
4. In the Resource server configuration section, do the following:
   a. Select the Configure this application as a resource server now option.
   b. Set the access token expiration time.
   c. Enable **Allow refresh token** and provide the refresh token expiration time.
   d. Under **Primary Audience**, enter **https:**//*STAP-UI:PORT/* (include the trailing slash)
   e. Add a scope named **stap**.
5. In the Client configuration section, select the **Configure this application as a client now** option.
6. In the Authorization section, do the following:
   a. In the **Allowed Grant Types** field, select **Resource owner, Client credentials, Authorization code, Refresh token** options.
   b. In the **Allowed Operations** field, select **Introspect, On behalf of** options.
   c. In the **Authorized Resources** field, select **All**.
7. Set **Redirect URL** to **https:**//*STAP-UI:PORT/oidc/redirect*.
8. Set **Post Logout Redirect URL** to **https:**//*STAP-UI:PORT/*.
9. Add a scope using the application name. This creates **https:**//*STAP-UI:PORT/stap*
10. In the Application Added pop-up window, make note of the client ID and client secret. You will provide this to the person who needs to generate the OAuth access token.
11. Click **Activate** and then click **Activate Application** to confirm the activation.

For more information on Oracle IDCS, see the *Oracle Identity Cloud Service* documentation.

Setting Up Persistent Volume

STAP uses two Persistent Volumes: one stores data in its microservices, the other contains published results of a scenario.

Topics in this section:

Setting Up Persistent Volume for STAP Microservices

To set up Persistent Volume for STAP microservices, you define it for just one microservice. TES and TDS share the same Persistent Volume. It is recommended to set up Persistent Volume for TES.

Before setting up Persistent Volume for the TES microservice, ensure you have a Persistent Volume available with a preferred access mode. It is recommended you use an RWO for a single writer, and an RWX for multiple replicas.

1. Create the **/data/config** directory. Ensure that following directories are set within **/data/config**:
   - **/data/config/attributeConfig**: Copy the **attributeData.properties** file and data faker plugin files here.
   - **/data/config/adapters**: Copy any adapter configuration files here. For example, the configuration file for PDF report generation.
   - **/data/config/context**: Copy the **global.ctx** file here, with key value pairs, if any.
   - **/data/config/actions**: Copy the UI action files and page property files here.
   - **/data/config/plugins**: Contains browser driver binaries and configurations. Copy browser drivers and configurations here, and run **chmod+x** on the executables.
2. Configure the path for the following helm values:

Table 4-1 Helm Values and the Corresponding Paths

Helm Value	Path
attributeData.home	/data/config/attributeConfig
adapters.home	/data/config/adapters
globalContext.home	/data/config/context
uiActions.home	/data/config/actions
plugins.home	/data/config/plugins

3. Mount the Persistent Volume named **tdaas-persistent-storage** in the **/data/config** directory with **readOnly** set to **false** so TES can write as needed.
4. Configure the pod's security context: run as a user, group, or FsGroup, or adjust ownership in an initContainer to ensure TES has write access.

> **Note**
>
> If you get a **Permission Denied** error, adjust the fsGroup or ownership under plug-ins.

5. Verify that your cluster's security policies, such as SELinux, Security Context Constraints, or Pod Security, allow writes to the Persistent Volume.
6. To verify the Persistent Volume, run the following command in the TES pod:

```
ls -l /data/config /data/config/context/global.ctx
```

If the TES logs confirms that configurations are loaded without any missing errors, the Persistent Volume is set up successfully.

If your TES logs return missing errors, confirm that mount path for the **/data/config** directory and its Helm values are correctly matched.

Setting Up Persistent Volume to Publish Data Files

1. Ensure TES and the host mount the PV at **/data/config** so TES can use the files. You can check the host mount by running **df -h** and verifying the PV is mounted on /**data/config**.
2. Create a directory **/data/config/dataFiles** is to store published data files.
3. Set the PV location in **publish-automation.properties** file by setting these values:

- tdaas.connection: tdaasEnvironment
- persistence-volume.environment: persistence-volume-environment
- **/data/config/dataFiles**: persistence-volume.location

4. Fill in the host connection details in **persistence-volume-environment.properties** so the publisher can write to the PV path:
   - name: persistence-volume
   - type: SSH
   - hostname: hostIp
   - port: 22
   - authorization: YES
   - authorization.type: basic
   - username: *username*
   - password: *password*
5. To test that you can connect and write, run an SSH command to the host, and create a test file in /data/config/dataFiles. Once successful, you can delete it.
6. Publish your scenarios so the PV copies each scenario's data folder into the path you set in **persistence-volume.location**.
7. To verify the Persistent Volume, run the following command in the TES pod:

   ```
   ls -l /data/config/dataFiles
   ```

   If the TES logs confirms that configurations are loaded without any missing errors, the Persistent Volume is set up successfully.

   If your TES logs return missing errors, confirm that mount path for the **/data/config** directory and its Helm values are correctly matched.

5
Installing STAP

Learn about the process of installing Oracle Communications Solution Test Automation Platform (STAP) on your system.

Topics in this document:

Downloading the STAP CNTK

Before setting up the STAP CNTK, ensure you meet the following prerequisites:

- If you are using OAuth, create an application for STAP within Oracle Identity Cloud Service.
- Provision a Network File System (NFS) that is accessible from the cluster.
- Install the necessary SSL certificates.
- Create a cluster running Kubernetes with Helm installed.

To set up the CNTK:

1. Create a namespace by running the following command:

   `kubectl create ns` *`namespace_name`*

2. Apply the image repository secret by running the following command:

   `kubectl apply -f` *`file_name`*

   where *file_name* is the YAML containing the secret to access your image repository
3. Download the Oracle STAP software pack from the Oracle software delivery Web site, located at:

   http://edelivery.oracle.com

4. Download the CNTK by running the following command:

   `curl oc-stap-1.25.1.x.zip`

   where:

   - *x* is the latest patch set

5. Unzip the CNTK by running the following command:

   ```
   unzip oc-stap-1.25.1.x.zip
   ```

6. Unzip the DE and microservices by running the following command:

   ```
   cd oc-stap-1.25.1.x
   unzip DE.zip ui.zip tds.zip tes.zip
   ```

Setting Up the Microservice Images

> **Note**
>
> Follow the steps below for each STAP microservice respectively.

1. Navigate to the **image.tar** folder for the STAP microservice, within its zip file in the STAP CNTK.
2. Load the image by running the following command:

   ```
   podman load -i image.tar
   ```

3. Tag the image by running the following command:

   ```
   podman tag image_name:tag repo_dest
   ```

   where

   - *image_name* is the title of the microservice image
   - *repo_dest* is your microservice image repository

4. Log in to your image repository and push the image to the repository by running the following command:

   ```
   podman push repo_dest
   ```

Installing STAP Microservices

Learn about installing the different microservices of STAP.

Topics in this section:

Installing TDS

To install TDS, perform the following steps:

1. Navigate to the TDS Helm chart.

2. Copy the **values.yaml** file and rename it to **override-values.yaml**.
3. Update the **override-values.yaml** file with the values listed in Table 5-1:

Table 5-1 TDS Helm Chart YAML Override Values

Key	Description
image.imageRepository	Contains the image repository.
image.imageName	Contains the image name.
image.imageTag	Contains the associated image tag.
image.imagePullSecret	Contains the secret name to pull the image.
image.imagePullPolicy	Contains the image pull policy. These are **Always**, **IfNotPresent**, and **Never**.
tdaasDB.host	Refers to the TDS database host name that runs in the container.
tdaasDB.port	Refers to the port where the TDS database runs in the container.
tdaasDB.username	Contains the TDS database username.
service.port	Refers to the port exposed by the Kubernetes service inside the cluster.
tdaas.port	Refers to the port where TDS runs in the container.
tdaas.externalport	Any external port not in use.
tdaas.host	Refers to TDS's host name.
taasUI.host	Refers to the UI's host name.
nfs.path	Contains the NFS path.
nfs.server	Refers to the NFS server IP. It should be accessible from the cluster.
storageClassName	Refers to the storageclassname of the PV.
pv.storage	Refers to the storage.
pv.mountPath	Refers to the mountPath to persist the database.
pv.volumeName	Refers to the volume name of the PV to mount in pod.
pvc.storage	Refers to the storage to claim from PV.
idcs.idcsUri	Refers to the IDCS configured URI (If you are using OAuth authentication type).
idcs.idcsClientId	Refers to the IDCS configured clientID (If you are using OAuth authentication type).
idcs.idcsOidcAudience	Refers to the IDCS OIDC audience (If you are using OAuth authentication type). The format for it is **idcs-url:443**.
ssl.enabled	Select true if SSL is enabled, else false.
ssl.secretName	Use the same secret name that you will use in Step 4.
securityType	Select the authentication type.
logLevel	Select the log level.
mail.host	Refers to the host name of the mail server (use only for basic authentication).
mail.port	Refers to the mail server port.
mail.from	Refers to the mail from configuration.
mail.auth	Refers to the authentication enabled details.

Table 5-1 (Cont.) TDS Helm Chart YAML Override Values

Key	Description
mail.username	Contains the user name.
mail.startTLS	Refers to the TLS enabled configuration.
mail.sslProtocol	Refers to the SSL protocol to be used to send mails.

4. If SSL is enabled, create a certificate secret by running the following command:

   ```
   kubectl create secret generic cert-secret --from-file=ssl cert file -n namespace
   ```

   > **Note**
   >
   > This ensures transport layer security over communication to the microservices.

5. Run the **tdaas-secret.sh** script under **STAP CNTK tds/scripts** and pass relevant options to create **TDS-secret**.
   where:
   - **Namespace**: Mandatory
   - **Keystore Password**: Optional (If SSL is enabled, enter the value)
   - **IDCS_CLIENT_ID**: Optional (If using OAuth authentication type, enter the value)
   - **IDCS_CLIENT_SECRET**: Optional (If using OAuth authentication type, enter the value)
   - **BASIC_PASSWORD**: Optional (If using Basic Authorization, enter the value)
   - **DB_PASSWORD**: Mandatory
   - **SMTP_PASSWORD**: Optional (For Basic Authorization only)
6. Install the Helm chart by running the following command:

   ```
   helm install chart_name path_to_chart_dir --values override-values.yaml -n namespace
   ```

7. Verify the deployment by running the following command:

   ```
   kubectl get all -n namespace
   ```

8. Perform a sanity check. For more information, see "Verifying a Successful Deployment".

Installing TES

To install TES, perform the following steps:

1. Navigate to the TES Helm chart.
2. Copy the **values.yaml** file and rename the file to **override-values.yaml**.
3. Update the **override-values.yaml** file with the values listed in Table 5-2.

Table 5-2 TES Helm Chart Override Values YAML Values

Key	Description
image.imageRepository	Contains the image repository.
image.imageName	Contains the image name.
image.imageTag	Contains the associated image tag.
image.imagePullSecret	Contains the secret name to pull the image.
image.imagePullPolicy	Contains the policy.
tls.enabled	Set to false to disable OAuth.
tls.secretName	The file containing TLS secrets for secure communication.
tls.trustKeystoreResourcePath	Path to the TLS trust keystore file used for secure communication. **Note:** Do not change this path.
tdaas.url	TDS URL.
tdaas.port	TDS port.
tdaas.username	TDS user name.
tes.host	TES hostname.
tes.port	Container port where TES is to be run.
taasUI.host	UI hostname.
cors.enabled	cors enabled option.
cors.allowOrigin	N/A
service.port	Port exposed by the Kubernetes Service inside the cluster.
idcs.idcsUri	IDCS configured URI (applicable only if the authorization is OAuth).
idcs.idcsClientId	IDCS configured Client ID (applicable only if the authorization is OAuth).
idcs.idcsOidcAudience	IDCS OIDC audience (applicable only if the authorization is OAuth). The format for it is **idcs-url:443**.
ssl.enabled	Select this as true if the SSL is enabled, otherwise false.
ssl.secretName	Secret name used to set up the secret containing the TLS cert. **Note:** Use the same secret name that you used in Step 4.
ssl.useCustomTruststore	If mounting custom trust store, select true. When true, ensure the secret **truststore-secret** is created before installing the chart.
securityType	Select the authentication type.
attributeData.home	Path in the pv attributeData folder path. **Note:** Copy the attribute data to the PV to run attributeData scenario.
adapters.home	The path to the PV adapters folder.
adapters.config.home	The path to the **config/adapters** folder. **Note:** Copy the adapters folder to the PV to configure them.
globalContext.home	The path to the PV globalContent folder. **Note:** Copy the **global.ctx** file to the PV to use global context, and edit global variables.

Table 5-2 (Cont.) TES Helm Chart Override Values YAML Values

Key	Description
uiActions.home	The path to the Persistent Volume UI actions folder.
plugins.home	The path to the Persistent Volume plugins folder.
env.proxy	Select true if proxy is to be set in pod.
env.http_proxy	Contains the HTTP proxy to be set in pod.
env.https_proxy	Contains the HTTP proxy to be set in pod.
env.no_proxy	Contains the no proxy to be set in pod.

4. If SSL is enabled, create a certificate secret:

   `kubectl create secret generic cert-secret --from-file=`*`cert file`* **`-n`** *`namespace`*

5. Create a secret to import the trust store by running the following command:

   `kubectl create secret generic truststore-secret --from-file=`*`jks file`***`-n`** *`namespace`*

6. Run the **tes-secret.sh** script and pass relevant options to create **TES-secret** *where*:
   - **Namespace**: Mandatory
   - **Keystore Password**: Optional (If SSL is enabled, enter the value)
   - **IDCS_CLIENT_ID**: Optional (If using OAuth authentication type, enter the value)
   - **IDCS_CLIENT_SECRET**: Optional (If using OAuth authentication type, enter the value)
   - **BASIC_PASSWORD**: Optional (If using Basic Authorization, enter the value)

   This contains the IDCS credentials, TLS passkey, and the DB password.
7. Install the Helm chart by running the following command:

   `helm install` *`chart_name path_to_chart_dir`* **`--values override-values.yaml -n`** *`namespace`*

8. Verify the deployment by running the following command:

   `kubectl get all -n` *`namespace`*

9. Perform a sanity check. For more information, see "Verifying a Successful Deployment".

> **Note**
>
> For your first login to the UI, the credentials will be the following:
>
> **User:** tesuser
> **Password:** As entered in the TDS secret script.
>
> You can use either **tesuser** or **admin** as the user name for your first log in.

> **Note**
>
> Ensure that you mount the NFS on the VM from which the publish will be initiated. The mount path on the VM is: **/data/config/attributeConfig**.

Installing the UI

To install STAP UI, perform the following steps:

1. Navigate to the UI Helm Chart located in *oc-stap-1.25.1.x***/ui/scripts**.
2. Apply the **ocir-secret.yaml** file with the namespace updated:

   ```
   kubectl apply -f file_name
   ```

3. Run the **ui-secret.sh** script to generate the **ui-secrets** file, noting the mandatory and optional fields:
   - **Namespace**: Mandatory
   - **Keystore Password**: Optional (If SSL is enabled, enter the value)
   - **IDCS_CLIENT_SECRET**: Optional (If using OAuth authentication type, enter the value)
   - **BASIC_PASSWORD**: Optional (If using Basic Authorization, enter the value)
4. Update the **override-values.yaml** file with the values listed in Table 5-3:

Table 5-3 STAP UI Helm Chart Override Values YAML Values

Key	Description
image.imageRepository	Contains the image repository.
image.imageName	Contains the image name.
image.imageTag	Contains the associated image tag.
image.imagePullSecret	Contains the secret name to pull the image.
image.imagePullPolicy	Contains the policy.
service.port	Port where UI is accessible within container.
tdaas.host	Host name of TDS.
tdaas.url	TDS URL at which the service is accessible.
tdaas.redirect	TDS URL or an alternative URL at which the service is accessible. Follow either of these formats: • http://ui:host:uiport/tdaas • https://ui:host:uiport/tdaas To access the UI microservice locally after deployment, set the localhost: https://localhost:UI_PORT/tdaas
tdaas.port	Port where TDS is running in cluster.
ui.host	Name of the UI service.
ui.port	Port where UI is running in cluster.
ui.path	Path where the UI project directory is present in the container: **/usr/share/test/ui**.
ui.externalPort	Any external port not in use.

Table 5-3 (Cont.) STAP UI Helm Chart Override Values YAML Values

Key	Description
ui.enable.reroute	If enable.reroute=false, TES and TDS urls are set as cookies to below configured values, and API calls from UI are directly transferred to TES and TDS without coming to this server.
frontendUri.host	HOST IP of the UI Service. To access the UI service locally after deployment, set **frontendUri.host** to **localhost.** Additionally, set **tes.redirect** and **tdaas.redirect** as mentioned in this table.
oauth.authurl	The authorization server URL for initiating authentication. Update only when authentication type is OAuth.
oauth.redirectUrl	The URL where users are redirected after authentication. Update only when authentication type is OAuth.
oauth.postLogoutUrl	The URL users are redirected to after logging out. Update only when authentication type is OAuth.
oauth.clientId	The unique identifier for the OAuth client application. Update only when authentication type is OAuth.
oauth.scope	The permissions requested for OAuth authentication. Update only when authentication type is OAuth.
oauth.login_url	The endpoint for initiating user login. Update only when authentication type is OAuth.
oauth.websocket_url	The WebSocket endpoint for real-time communication. Update only when authentication type is OAuth.
tls.enabled	Set to false, to disable OAuth.
tls.secretName	The file containing TLS secrets for secure communication.
tls.trustKeystoreResourcePath	Path to the TLS trust keystore file used for secure communication. **Note:** Do not change this path.
tes.host	Host name of TES.
tes.url	TES URL at which the service is accessible.
tes.redirect	TES URL or an alternative URL at which the service is accessible. Follow either of these format: • **http://***ui:host:uiport/tes* • **https://***ui:host:uiport/tes* To access the UI microservice locally after deployment, set the localhost: **https://***localhost:UI_PORT/tes*
tes.port	Port where TES is running in container.
security.type	BASIC/OAUTH

5. Install the Helm chart by running the following command:

   ```
   helm install chart_name path_to_chart_dir --values override-values.yaml -n namespace
   ```

6. Verify the deployment by running the following command:

   ```
   kubectl get all -n namespace
   ```

7. Perform a sanity check. For more information, see "Verifying a Successful Deployment".

> **Note**
>
> For your first login to the UI, the credentials will be the following:
>
> **User:** tesuser/admin
>
> **Password:** As entered in the TDS secret script. You can use either **tesuser** or **admin** as the user name for your first log in.
>
> Oracle recommends that you change the password after the first log in to a strong, more suitable password.

Configuring JVM Heap Size to Avoid “Out of Memory" Error Messages

To prevent “Out of Memory” error messages in the log file after installation, it is recommended to configure the heap size for the Java Virtual Machine (JVM) before running your application.

By default, the JVM may use limited heap memory, which can lead to memory allocation errors when handling larger workloads. Setting the initial and maximum heap size allows the JVM to manage memory more efficiently, avoiding unexpected failures or periodic performance slowdowns due to dynamic heap resizing.

You can specify the minimum and maximum heap size by including the following options:

-Xms specifies the initial heap size

-Xmx specifies the maximum heap size

For example:

```
-Xms1g -Xmx2g
```

where **1g** and **2g** indicate the initial and maximum heap sizes (in gigabytes) respectively. Adjust these values according to your system’s available resources and expected workload.

It is recommended to define heap size options using the JAVA_TOOL_OPTIONS environment variable, which is recognized automatically by the JVM at startup. To do so, use these commands for your deployment environment:

If you're using Linux:

```
export JAVA_TOOL_OPTIONS="-Xms1g -Xmx2g"
```

If you're using Windows:

```
set JAVA_TOOL_OPTIONS=-Xms1g -Xmx2g
```

To set it in your Dockerfile:

```
ENV JAVA_TOOL_OPTIONS="-Xms1g -Xmx2g"
```

Alternatively, you can update your startup scripts in the STAP DE to include the heap size options directly:

```
java -Xms1g -Xmx2g -XX:-PrintWarnings -XshowSettings:vm -jar "%STAP_HOME%"/lib/
STAP-1.25.1.1-DE.jar %*
```

where:

- *1g* and *2g* indicate the initial and maximum heap sizes (in gigabytes) respectively
- *-PrintWarnings* enables or disables JVM warnings
- *-XshowSettings:vm* is the JVM memory configuration during startup
- *%STAP_HOME%"/lib/STAP-1.25.1.1-DE.jar* is the STAP DE jar file

6
Verifying the Installation

Learn about verifying a successful deployment for Oracle Communications Solution Test Automation Platform.

Topics in this document:

Verifying a Successful Deployment

To check if your deployment has run successfully:

1. Check if the pod is in **Running** state by running the following command:

   ```
   kubectl get pods -n namespace_name
   ```

2. Check the logs for any errors by running the following command:

   ```
   kubectl logs pod_name -n namespace_name
   ```

   If no errors appear, the deployment has run successfully.

3. Run the following command to retrieve External IP and Port Number for the services:

   ```
   kubectl get po -n namespace_name -o wide
   ```

   To validate Basic Auth deployment:

   - **TDS Service**: Run the following command to check if the service is accessible:

     ```
     curl http://TDS_NodeIP:port/job
     ```

     The command should run without errors or connection failures.

   - **TES Service**: Verify TES service accessibility using the following command:

     ```
     curl http://TES_NodeIP:port/taas
     ```

     The command should return a response without errors.

   - **UI Service**: Open the UI in a browser using the following URL:

     ```
     http://UI_NodeIP:port
     ```

     If you are able to login to the UI, the deployment is working as expected.

4. To validate OAuth deployment:
   - Use the **IDCS credentials** to generate an OAuth access token. The token should be included in the **Authorization Header** as a **Bearer Token** for subsequent requests.
   - Validate Service Accessibility:

     TDS Service

     – Open **Postman** and create a **GET** request to the following:

       `https:`*`//TDS_NodeIP:port/job`*

     – In the **Headers** section, add the following:

       ```
       Authorization: Bearer access_token
       ```

       Send the request. The response should return with no errors or connection failures.

     TES Service

     – Open **Postman** and create a **GET** request to the following:

       `https:`*`//TES_NodeIP:port/taas`*

     – In the **Headers** section, add the following:

       ```
       Authorization: Bearer access_token
       ```

     – Send the request. The response should indicate successful connectivity.

     UI Service

     – Open the UI in a browser using the following URL:

       `https:`*`//UI_NodeIP:port`*

     – If required, tunnel the connection locally before accessing the URL. You need to tunnel the connection locally if:
       * You want to access the UI through a browser but don't have VNC set up.
       * Network blocks or firewalls prevent direct access to the UI.
       * You can't set up a load balancer for access.
       * You're working from a location without direct access to the UI.

     – Log in using OAuth credentials. If authentication is successful, the deployment is functioning correctly.

> **Note**
>
> You can verify if your environment variables are correctly set in the container by following these steps:
>
> - Run the following command:
>
>   ```
>   kubectl exec -it pod_name -n namespace -- /bin/sh
>   ```
>
> - Navigate to the **config** directory and verify variables and values.
> - If an environment variable is missing, check the deployment manifest or the **override-values.yaml** file for incorrect configurations.

Setting Up the STAP Design Experience

The STAP Design Experience package simplifies the automation of end-to-end scenarios by offering a user-friendly Behavior-Driven Development (BDD) environment for creating, testing, and deploying automation. For more information on the STAP Design Experience, see "Using the STAP Design Experience Package" in *User Operations Guide*.

Before setting up the STAP Design Experience, ensure you have installed the correct version of Java. See "Common Software Compatibility" in *Compatibility Matrix*,

To set up the STAP Design Experience package, follow these steps.

1. Download the STAP Design Experience package, titled **oc-stap-1.25.1.1.zip** inside the **DE.zip** file.
2. Unzip the files in the package. It contains the following folders:
   - **lib:** Contains the STAP library.
   - **sampleWorkSpace:** Contains the sample automation workspace.
   - **stap:** Contains the STAP command-line script.
   - **run.sh**, **compile.sh**, **run.cmd**, and **compile.cmd:** Contain sample scripts to perform STAP operations.

To set up STAP Design Experience on a Linux or a Mac, see "Configuring the STAP Design Experience on Linux".

To set up STAP Design Experience on Windows, see "Configuring STAP Design Experience on Windows".

Configuring the STAP Design Experience on Linux

To configure STAP Design Experience on Linux, follow these steps:

1. Set the STAP environment variables using the following command:

   ```
   export STAP_HOME=packageLoc
   export PATH=$STAP_HOME:$PATH
   ```

   where:

   - *packageLoc* is the location of your STAP DE package

- *STAP_HOME:PATH* is the path to your STAP repository

2. In a new terminal window, start WireMock, which is a mock server to run the sample scenarios.
   If you are using the Bourne shell, run the following commands:

   ```
   cd
   $STAP_HOME/sampleWorkSpace/WireMock
   startWireMock.sh
   ```

   If you are using another shell, run the following commands:

   ```
   cd
   $STAP_HOME/sampleWorkSpace/WireMock
   sh startWireMock.sh
   ```

3. Compile sample automation scenarios.
   If you are using the Bourne shell, run the following commands:

   ```
   cd
   $STAP_HOME/
   ./compile.sh
   ```

   If you are using another shell, run the following commands:

   ```
   cd
   $STAP_HOME/
   sh compile.sh
   ```

4. In the same terminal window, run the sample automation scenarios.
   If you are using the Bourne shell, run the following commands:

   ```
   cd
   $STAP_HOME/
   ./run.sh
   ```

   If you are using another shell, run the following commands:

   ```
   cd
   $STAP_HOME/
   sh run.sh
   ```

Configuring STAP Design Experience on Windows

To configure STAP Design Experience on Windows, follow these steps:

1. Set the STAP environment variables by following these steps
   - Update the **setenv.cmd** file with the location where the STAP DE package is extracted, by running the following command:

     ```
     set STAP_HOME=packageLoc
     set PATH=%STAP_HOME%;%PATH%
     ```

- At the command prompt, navigate to design experience folder titled **DE.zip** in the CNTK folder and set the environment variables using the following command:

```
setenv.cmd
```

2. In a new terminal window, start WireMock, which is a mock server to run the sample scenarios, by running the following command:

```
cd %STAP_HOME%\sampleWorkSpace\WireMock
startWireMock.cmd
```

3. In another terminal window, compile sample automation scenarios by running the following command:

```
cd %STAP_HOME%
compile.cmd
```

> **Note**
>
> If successful, **compile.cmd** returns with a 100% pass.

4. In the same terminal window as the previous step, run the sample automation scenarios:

```
cd %STAP_HOME%
run.cmd
```

If successful, **run.cmd** returns with a scenario summary report in the terminal window, alongside an automation report generated in the STAP UI.

Accessing STAP Microservice URLs

After STAP microservices are deployed, you can access URLs of each microservice.

To access URLs of STAP microservices, follow these steps:

1. Collect the port details by running the following command:

```
kubectl get svc -n namespace
```

2. To access each microservice, run the following command:

```
https://node ip:port
```

> **Note**
>
> If you do not have SSL configured, use `http`.

7

Upgrading STAP

Learn about upgrading your Oracle Communications Solution Test Automation Platform (STAP) cloud native environment to the latest patch set or interim patch release.

Topics in this document:

Upgrading your STAP Environment

To upgrade your STAP environment:

1. Download the new version of STAP CNTK:

   ```
   curl cntk_link cntk_zip
   unzip cntk_zip
   ```

   where:

   - *cntk_link* is the link to download the CNTK
   - *cntk_zip* is the location of the CNTK

2. Load all the images for each microservice to be upgraded from the CNTK image folder:

   From the *CNTK_home***/microservice_directory**:

   ```
   podman load -i image.tar
   ```

3. Tag the images for upload:

   ```
   podman tag image_name:image_id repoDest
   ```

4. Push the images to the repository:

   ```
   podman push repoDest
   ```

   > **Note**
   >
   > Repeat the following steps for each microservice.

5. Navigate to the *directoryName/***stap-microservice/helm_chart/** directory.

   where:

   - *directoryName* is the name of your STAP microservice directory
   - *stap-microservice* is each STAP microservice
   - *helm_chart* is the helm chart of the respective STAP microservice

   Update the **override-values.yaml** file with the new **imageName**.

6. List Helm releases in the namespace to get the chart name:

   ```
   helm ls -n namespace
   ```

7. Upgrade the Helm chart:

   ```
   helm upgrade chart_name path_to_chart_dir --values override-values.yaml -n
   namespace
   ```

8

Uninstalling STAP

Learn how to uninstall your Oracle Communications Solution Test Automation Platform (STAP) cloud native environment.

Topics in this document:

Uninstalling your STAP Environment

To uninstall your STAP environment, run the following commands:

```
helm uninstall tds_chart_name -n namespace
helm uninstall tes_chart_name  -n namespace
helm uninstall ui_chart_name  -n namespace
kubectl delete ns namespace
clean up nfs
```

This uninstalls all microservices, deletes the Kubernetes namespaces, and cleans up the NFS.

9

Backing Up And Restoring the STAP Database

Learn how to backup and restore the Oracle Communications Solution Test Automation Platform (STAP) database.

Topics in this document:

Applicability

- STAP 1.26.1.0.0 and later: Use the script-based logical backup/restore (backup.sh, restore.sh, and so on.) described in this chapter.
- STAP 1.25.x and earlier: Use the legacy PV/NFS folder backup method (tar/zip of the /data/db folder) unless otherwise instructed.
- Migration notes: Backups from 1.25.x are not directly compatible with 1.26.1. For more information, see “Migrating from STAP 1.25.x to 1.26.1”.

Prerequisite (STAP 1.26.1.0.0)

1. Ensure tdaas-cntk contains these scripts:
   - **backup.sh**
   - **config.sh**
   - **restore.sh**
   - **service-port.sh**
   - **values-update.sh**
2. Grant run permission:
   chmod +x *.sh

> **Note**
>
> You must have administrator access to the cluster/NFS/PV to access **/data** and run these scripts.

Backing Up the STAP Database (STAP 1.26.1.0.0)

To backup STAP database, perform the following steps:

1. Update **config.sh** with your environment values (paths may be relative to the scripts directory or absolute):
   - RELEASE_NAME=**"tdaas-helm"**
   - NAMESPACE=**"<your-namespace>"**

- CHART_PATH=**"../helm_chart/taas-tdaas-helm-chart"**
- VALUES_FILE=**"$CHART_PATH/override-values.yaml"**
- SERVICE_NAME=**"tdaas-service"**

2. Run the following backup script:**./backup.sh**
3. Verify the backup files are created under the mounted path (for example, **/data/backups**), for example:
   - **db_bkp_data_only_YYYYMMDD_HHMMSS.sql.gz (data-only backup)**
   - **db_bkp_full_YYYYMMDD_HHMMSS.sql.gz (schema + data + triggers + routines + events)**

Restoring the Database (STAP 1.26.1.0.0)

Restore can be performed only after a fresh deployment when the database is empty. To restore the database::

1. Copy the data-only backup file to PV/NFS (for example under **/data/backups or /data**).
2. Update **config.sh**:
   - RELEASE_NAME=**"tdaas-helm"**
   - NAMESPACE=**"<your-namespace>"**
   - CHART_PATH=**"../helm_chart/taas-tdaas-helm-chart"**
   - VALUES_FILE=**"$CHART_PATH/override-values.yaml"**
   - SERVICE_NAME=**"tdaas-service"**
3. Trigger restore::

   ```
   ./restore.sh /data/backups/db_bkp_data_only_YYYYMMDD_HHMMSS.sql.gz
   ```

4. Verify the database has been restored with the TDS APIs. For more information, see *REST API Reference for STAP*.

Migrating from STAP 1.25.x to 1.26.1.0.0

Backup from 1.25.x is NOT directly compatible with 1.26.1. Use the approach below to generate a compatible backup from the 1.25.x environment.

1. Place the following scripts under tdaas-cntk in the STAP 1.25.x environment:
   - **backup.sh**
   - **config.sh**
   - **restore.sh**
   - **service-port.sh**
   - **values-update.sh**
2. Update the Helm template in the 1.25.x environment:
   - File: **helm_chart/taas-tdaas-helm-chart/templates/tdaas-backup-job.yaml**
   - Apply the job content shown in "tdaas-backup-job.yaml (Migration Helper for STAP 1.25.x)".
3. Run the backup job on 1.25.x to produce:

- **db_bkp_data_only_YYYYMMDD_HHMMSS.sql.gz** (used for restoring into 1.26.1)
- **db_bkp_full_YYYYMMDD_HHMMSS.sql.gz** (reference or archival)

4. Copy the data-only backup to the target 1.26.1 environment PV/NFS path (for example, /data/backups).
5. Restore into 1.26.1 using:

```
./restore.sh /data/backups/db_bkp_data_only_YYYYMMDD_HHMMSS.sql.gz
```

> **Note**
>
> The NFS database folder should be exclusively allocated and used by only one TDS microservice at a single point in time.

tdaas-backup-job.yaml (Migration Helper for STAP 1.25.x)

Use this job template update in the 1.25.x Helm chart to generate backups compatible with the 1.26.1 restore approach.

```
{{- if .Values.backup.enable }}
apiVersion: batch/v1
kind: Job
metadata:
  name: db-backup-job
  labels:
    app: db-backup
spec:
  template:
    spec:
      restartPolicy: Never
      containers:
        - name: db-backup
          image: "{{ .Values.image.imageRepository }}
{{ .Values.image.imageName }}:{{ .Values.image.imageTag }}"
          command: ["/bin/sh", "-c"]
          args:
            - |
              set -e
              echo "=== DB Backup Job Starting ==="

              # Validate required env vars
              if [ -z "$DB_HOST" ] || [ -z "$DB_USER" ] || [ -z
"$MYSQL_PWD" ]; then
                echo "ERROR: Required DB env vars are missing"
                exit 1
              fi

              # Wait for MySQL
              echo "Waiting for MySQL at $DB_HOST:$DB_PORT..."
              for i in $(seq 1 10); do
                if mysqladmin ping -h "$DB_HOST" -P "$DB_PORT" -u "$DB_USER"
--protocol=TCP --silent 2>/dev/null; then
                  echo "MySQL is reachable."
                  break
```

```
                fi
                echo "Attempt $i/10 — retrying in 5s..."
                sleep 10
                if [ "$i" -eq 10 ]; then
                  echo "ERROR: MySQL not reachable"
                  exit 1
                fi
              done

              mkdir -p /data/backups
              TIMESTAMP=$(date +"%Y%m%d_%H%M%S")

              # Foreign key safe table order
              TABLE_ORDER="environment runtime_scenario
simulation_runtime_result \
              job execution connection \
              runtime_case runtime_case_result runtime_step
runtime_step_result \
              simulation_event_result simulation_event_failure_result
simulation_event_graph_result \
              event action log persistent_store runtime_scenario_result
search_query_counter search_schema user execution-id"

              # Filter only existing tables (KEEP THIS LOGIC)
              EXISTING_TABLES=""
              for t in $TABLE_ORDER; do
                if mysql -h "$DB_HOST" -P "$DB_PORT" -u "$DB_USER" -
p"$MYSQL_PWD" \
                     -e "use $DB_NAME; show tables like '$t';" | grep -q "$t";
then
                  EXISTING_TABLES="$EXISTING_TABLES $t"
                else
                  echo "WARNING: Table '$t' not found, skipping..."
                fi
              done

              echo "Tables to be dumped: $EXISTING_TABLES"

              # ------------------------
              # 1 Data-only backup
              # ------------------------
              DATA_ONLY_FILE="/data/backups/db_bkp_data_only_$
{TIMESTAMP}.sql.gz"
              echo "Starting DATA-ONLY backup..."
              mysqldump -h "$DB_HOST" -P "$DB_PORT" -u "$DB_USER" \
                --protocol=TCP --single-transaction --quick --lock-
tables=false \
                --no-create-info "$DB_NAME" $EXISTING_TABLES | gzip >
"$DATA_ONLY_FILE"

              # ------------------------
              # 2 Full backup
              # ------------------------
              FULL_FILE="/data/backups/db_bkp_full_${TIMESTAMP}.sql.gz"
              echo "Starting FULL backup..."
              mysqldump -h "$DB_HOST" -P "$DB_PORT" -u "$DB_USER" \
```

```
                --protocol=TCP --single-transaction --quick --lock-
tables=false \
                --routines --triggers --events \
                "$DB_NAME" $EXISTING_TABLES | gzip > "$FULL_FILE"

              echo "Backup completed"
              echo "Data-only: $DATA_ONLY_FILE"
              echo "Full     : $FULL_FILE"
          env:
            - name: MYSQL_PWD
              valueFrom:
                secretKeyRef:
                  name: tdaas-secrets
                  key: DB_PASSWORD
            - name: DB_HOST
              value: "{{ .Values.tdaasDB.host }}"
            - name: DB_USER
              value: "{{ .Values.tdaasDB.username }}"
            - name: DB_NAME
              value: "{{ .Values.tdaasDB.dbName | default "taas-tdaas-db" }}"
            - name: DB_PORT
              value: "{{ .Values.tdaasDB.port | default 3306 }}"
          volumeMounts:
            - mountPath: "/data"
              name: tdaas-persistent-storage
      volumes:
        - name: tdaas-persistent-storage
          persistentVolumeClaim:
            claimName: tdaas-pvc
      imagePullSecrets:
        - name: {{ .Values.image.imagePullSecret }}
  backoffLimit: 2
  activeDeadlineSeconds: 1800
{{- end }}
```

10

Troubleshooting STAP Deployment

Learn about errors you may run into when deploying Oracle Communications Solution Test Automation Platform (STAP) and how to fix them.

Topics in this document:

PVC Stuck in Pending Status

If your **PersistentVolumeClaims** (PVC) is stuck in **Pending** status, follow these steps:

- Ensure a suitable **PersistentVolume** (PV) is available.
- Verify that the PV mentions a valid **StorageClass** and matches the available PVs.
- Check if the storage provisioner is running and configured properly.

> **Note**
>
> To check PVC status, run `kubectl get pvc`.

Error Message: WebSocket Not Connected

After deployment, if you get the "WebSocket Not Connected" error message:

If you have network restrictions in cluster, check the configuration for the **override-values.yaml** for the UI Helm Chart:

- If you are working with Basic Auth deployment, ensure the hostname for Test Execution Service (TES) microservice is **localhost** with valid port numbers (if accessing outside the locally tunneled cluster, and no public load balance is used).
- If you are working with OAuth deployment, ensure the Node IPs are mentioned for TES with correct port numbers when accessing from the same network as Cluster IPs.

helmchart.tgz File Does Not Unzip

If you are unable to unzip the **helmchart.tgz** file using the `unzip` command, perform the following actions:

- Use the `tar -xvzf helmchart.tgz` command to solve this error.
- To extract the chart to a specific directory, run the following command:

  ```
  tar -xvzf helmchart.tgz -C /desired/path
  ```

Error When Applying Image Pull Secret File

If you get an error when applying the image pull secret file, follow these steps:

- Check if the file is correctly formatted.
- Ensure the namespace in the image pull secret file matches the one you created.
- Run the following command to verify if the secret was applied correctly:

  ```
  kubectl describe secret <namespace-secret>.yaml -n <namespace_name>
  ```

Deployment Stuck in ContainerCreating State

If your deployment is stuck in **ContainerCreating** state, follow these steps:

- Run `kubectl describe pod <pod_name> -n <namespace>` to check for volume mount issues or image pull failures.
- Ensure the `imagePullSecret` is correctly set up and matches the secret name.
- Verify if the required PVs and PVCs are correctly bound.

SSL Configuration Not Working

If your SSL configuration is not working, follow these steps:

- Ensure the secret for SSL certificates is correctly created using the following command:

  ```
  kubectl create secret generic cert-secret --from-file=<ssl cert file> -n
  <namespace>
  ```

- Verify if the Helm chart is correctly referencing `ssl.enabled` and `ssl.secretName`.
- Check logs for TLS handshake errors using the following command:

  ```
  kubectl logs <pod_name> -n <namespace>
  ```

Podman Push Fails

If your Podman push command fails, follow these steps:

- Check if the authentication to the container registry is configured properly.

- Verify network connectivity and registry availability.
- Ensure the repository name follows the correct format and exists in the registry.
- Make sure there is enough space in the registry to host the images.

NFS Does Not Mount

If you are facing errors when mounting your Network File Storage (NFS), follow these steps:

- Ensure the correct `nfs.server` and `nfs.path` are set in the `override-values.yaml` file.
- Check for NFS errors in pod logs by running the following command:

  ```
  kubectl describe pod <pod_name> -n <namespace>
  ```

- Verify if the NFS server is reachable from the Kubernetes cluster.