Oracle® Communications

Unified Inventory and Topology Deployment Guide

Release 7.6
F95168-02
January 2025

Oracle Communications Unified Inventory and Topology Deployment Guide, Release 7.6

F95168-02

Contents

Preface

Audience viii
Documentation Accessibility viii
Diversity and Inclusion viii

1 About Unified Inventory and Topology

Unified Inventory and Topology Architecture 1-1
About UIM 1-2
About UTIA 1-2
About Unified Operations Message Bus 1-3
About Common Authentication 1-3
Planning and Validating Your Cloud Environment 1-3
- Installing Oracle Property Graph 1-3
- Kubernetes Storage Class 1-4

2 About the Unified Inventory and Topology Toolkit

Unified Inventory and Topology Toolkit 2-1
Image Builders 2-2
- About the Manifest File 2-2

Deployment Toolkits 2-4
- Common Cloud Native Toolkit 2-4

Deploying the Services 2-5
Setting Up Prometheus and Grafana 2-6
Setting Up Elastic Stack 2-6
Setting Up OpenSearch 2-8
Adding Common OAuth Secret and ConfigMap 2-9

3 Deploying the Common Authentication Service

Building the OHS Image 3-1
Deploying OAM along with OHS for Authentication Service 3-1
Deploying OAM Using Common Cloud Native Toolkit Scripts 3-2

Using Wild Card Certificates 3-5
Configuring Ingress and Ingress Controller for OAM 3-6
Upgrading OAM 3-7
Uninstalling OAM 3-7
Specifying the Proxy Settings 3-8
Accessing the WebLogic Server Administration Console and the OAM Console 3-8
Configuring OAM 3-9
Configuring OAuth Service Settings 3-12
Creating an OAuth Identity Domain 3-12
Creating a Resource 3-13
Creating a Client 3-13
Debugging and Troubleshooting 3-14
Unable to create Domain or Admin Server is not coming up 3-14
Unable to Access OAM Console 3-15
Inventory UI is not appearing after successful login 3-16
UIM UI Not Accessible on Using SSL Port of Traditional UIM Instance 3-17
Self-signed SSL Certificates 3-19
Generating Self-signed Certificates 3-19
Generating Wild Card SSL Certificate 3-20

4 Deploying Unified Operations Message Bus

Message Bus Cloud Native Architecture 4-2
Access to Message Bus 4-2
Strimzi Operator 4-4
Create Global Resources 4-4
Private Container Repository 4-4
ImagePullPolicy 4-5
Resources 4-6
Deploying Strimzi Operator 4-6
Upgrading Strimzi Operator 4-7
Uninstalling Strimzi Operator 4-7
Validating Strimzi Operator 4-7
Restarting the Strimzi Operator 4-8
Registering the Namespaces with Strimzi Operator 4-8
Unregistering the Namespaces with Strimzi Operator 4-8
Deploying and Managing Message Bus 4-8
Deploying Message Bus 4-9
Upgrading Message Bus 4-9
Deleting Message Bus 4-9
Validating Message Bus 4-10
Restarting Message Bus 4-11

Configuring the applications.yaml File 4-11
Using Image Pull Secrets 4-12
Security Context 4-12
Cluster Size 4-13
Storage 4-13
Broker Defaults 4-14
JVM Options 4-14
Kafka Topics 4-14
Accessing Kafka Cluster 4-15
Configuring Authentication 4-17
Using GC Logs 4-20
Alternate Configuration Options 4-20
Log Level 4-20
Choosing Worker Nodes for Running Message Bus Service 4-20
Managing Message Bus Metrics 4-23
Installing and Configuring Mirror Maker 2.0 4-24
Configuring Source and Target Message Bus (Kafka cluster) Details 4-24
Installing Mirror Maker 4-25
Uninstalling Mirror Maker 4-26
Client Access 4-26
Configuring Message Bus Listeners 4-35
Debugging and Troubleshooting 4-38

5 Deploying the Unified Topology for Inventory and Automation Service

Overview of UTIA 5-1
UTIA Architecture 5-1
UIM as the Producer 5-2
Topology as the Consumer 5-2
Topology Graph Database 5-2
Topology In-Memory Database 5-3
UTIA User Interface 5-3
Creating UTIA Images 5-3
Prerequisites for Creating UTIA Images 5-3
Configuring Unified Topology Images 5-3
Creating Unified Topology Service Images 5-3
Post-build Image Management 5-5
Customizing the Images 5-5
Creating a Unified Topology Instance 5-5
Installing Unified Topology Cloud Native Artifacts and Toolkit 5-6
Setting up Environment Variables 5-6
Registering the Namespace 5-7

Creating Secrets 5-7
Installing Unified Topology Service Schema 5-11
Configuring the applications.yaml File 5-13
Configuring Unified Topology Application Properties 5-14
Max Rows 5-15
Date Format 5-15
Alarm Types 5-15
Event Status 5-15
Event Severity 5-16
Path Analysis Cost Values 5-16
Integrate Unified Topology Service with Message Bus Service 5-17
Creating a Unified Topology Instance 5-18
Accessing Unified Topology 5-18
Validating the Unified Topology Instance 5-19
Deploying the Graph Server Instance 5-19
Scheduling the Graph Server Restart CronJob 5-20
Affinity on Graph Server 5-20
Upgrading the Unified Topology Instance 5-21
Restarting the Unified Topology Instance 5-21
Alternate Configuration Options for UTIA 5-22
Setting up Secure Communication using TLS 5-22
Enabling Authentication for UTIA 5-25
Registering UTIA in Identity Provider 5-25
Common Secret and Properties 5-26
Registering Identity Provider for UTIA 5-27
Choosing Worker Nodes for Unified Topology Service 5-29
Setting up Persistent Storage 5-30
Managing Unified Topology Logs 5-31
Viewing Logs using Elastic Stack 5-31
Setting Up Elastic Stack 5-31
Viewing Logs using OpenSearch 5-33
Managing Unified Topology Metrics 5-33
Allocating Resources for Unified Topology Service Pods 5-35
Scaling Up or Scaling Down the Unified Topology Service 5-35
Enabling GC Logs for UTIA 5-35
Geo Redundancy Support 5-36
Disaster Recovery Support 5-38
Disaster Recovery across Data Centers 5-38
About Switchover and Failover 5-39
About Kafka Mirror Maker 5-40
Installation and Configuration 5-40
Setting up the Primary (active) Instance 5-40

Setting up the Secondary (standby) Instance 5-42
Switchover Sequence 5-44
Failover Sequence 5-44
Debugging and Troubleshooting 5-45
Fallout Events Resolution 5-47
Deleting and Recreating a Unified Topology Instance 5-48
UTIA Support for Offline Maps 5-49
Allowlisting Map URLs 5-49
Setting Up a Local Tile Server 5-50
Manual Changes for Setting Up a Local Tile Server 5-50

6 Data Migration and Dynamic Attribute Mapping

Planning the Topology Migration 6-1
Customizing Topology JSON files for Migration 6-4
Dynamic Data Mapping from UIM 6-11
Mapping the Dynamic Data from UIM 6-12

7 Upgrading UTIA

Prerequisites for Upgrading UTIA 7-1
Upgrading the UTIA Application 7-2
Upgrading the UTIA Schema 7-2
Upgrading the UTIA Instance 7-3

8 Checklists for Integration of Services

Integrating UIM with UTIA and Message Bus 8-5
Integrating UIM CN with Message Bus and UTIA 8-5
Integrating Traditional UIM with Message Bus and UTIA 8-6

Preface

This guide describes how to deploy and administer Oracle Communications Unified Inventory and Topology in a cloud native environment.

Audience

This document is for system administrators, database administrators, and developers who install and configure Unified Inventory and Topology.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

About Unified Inventory and Topology

Unified Inventory and Topology includes the following services:

- Unified Inventory Management (UIM)
- Unified Topology for Inventory and Automation (UTIA)
- Unified Operations Message Bus
- Common Authentication that leverages Oracle Access Manager (OAM) tool

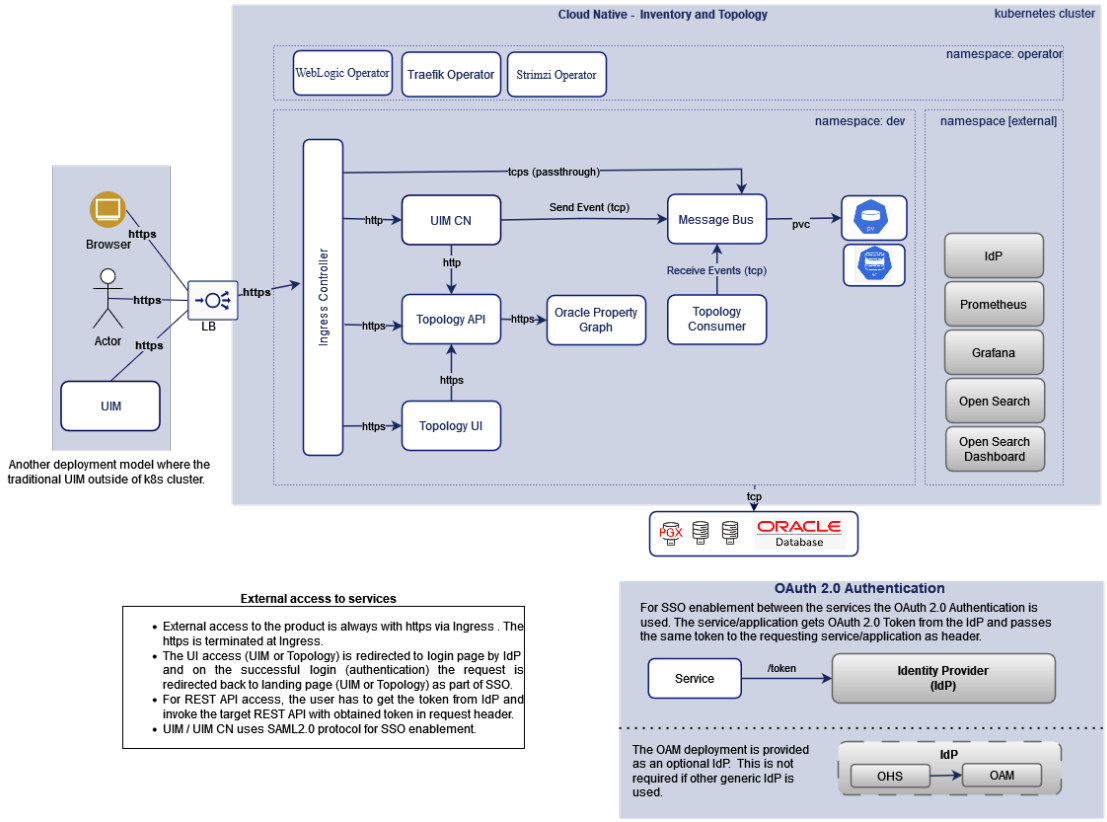
UTIA, Unified Operations Message Bus, and Common Authentication are cloud native containerized applications that are supported in a Kubernetes environment. UIM can be a traditional application or a cloud native instance.

- The embedded topology from UIM is now available as a micro-service (UTIA) based on Helidon MP.
- The communication between UIM and the UTIA service is asynchronous and this is achieved by using Message Bus service.
- OAM is an optional Identity Provider that supports OAuth2.0 protocol, used for single sign-on (SSO).

Unified Inventory and Topology Architecture

Figure 1-1 shows a high-level architecture of Unified Inventory and Topology and how the services communicate.

Figure 1-1 High-level Architecture of Unified Inventory and Topology



See the corresponding architecture diagrams of the services for more information.

About UIM

UIM is a standards-based telecommunications inventory management application that enables you to model and manage customers, services, and resources. UIM supports complex business relationships and provides full life-cycle management of services and resources. UIM provides you with a real-time, unified view of customers, services, and resource inventory, enabling you to develop and introduce new services quickly and cost-effectively. UIM supports two deployment models: traditional (on-premise) deployment and cloud native deployment in a Kubernetes cluster.

About UTIA

Unified Topology for Inventory and Automation (UTIA) enables you to view the service, network, and resource topologies in the form of topology graphs. UTIA uses Oracle Property Graph DB to manage the topology hierarchy.

UTIA has the following sub components.

- Unified Topology API
- Unified Topology PGX
- Unified Topology Consumer

- Unified Topology UI

See *UTIA User's Guide* for more information.

About Unified Operations Message Bus

Message Bus is a distributed event store and stream-processing service. Message Bus service sends and receives events and messages asynchronously to a specific destination (called as **Topic**) between the services. The Message Bus service uses Apache Kafka, which is a distributed event store and stream-processing platform, as the messaging platform. For packaging or deploying, Strimzi is used. Strimzi simplifies the process of running Apache Kafka in a Kubernetes cluster. Strimzi also provides container images and operators for running Kafka on Kubernetes.

About Common Authentication

The Common Authentication service leverages **Oracle Access Manager (OAM)** or any Identity Provider to implement the single sign-on (SSO) authentication solution with the services (UIM, Unified Topology services, and Message Bus service). This enables you to seamlessly access multiple applications without being prompted to authenticate for each application separately. The main advantage of SSO is that you are authenticated only once, which is when you log in to the first application; you are not required to authenticate again when you subsequently access different applications within the same web browser session.

OAM also supports the single logout (SLO) feature. If you access multiple applications using SSO within the same web browser session, and then if you log out of any one of the applications, you are logged out of all of the applications.

For more information about OAM, see Administering Oracle Access Management.

Planning and Validating Your Cloud Environment

To deploy the Unified Topology for Inventory and Automation services, you must set up and validate a list of prerequisite software. See **Planning and Validating Your Cloud Environment** in *UIM Cloud Native Deployment Guide* for more information.

Before starting the service deployments:

- Install property graph plugins on the PDB that are used for UTIA.
- Configure the Storage Class in Kubernetes to provision Persistent Volumes dynamically to be used for the Message Bus service.

Installing Oracle Property Graph

UTIA uses Oracle Property Graph of Oracle Database that offers a powerful graph support to explore and discover complex relationships within UTIA topology graphs.

Graph Server and Client is a software package that is required for Property Graph.

To install Property Graph:

1. Download Oracle Graph Server, **oracle-graph-plsql-<*version*>.zip,** from Oracle E-Delivery: https://www.oracle.com/database/technologies/spatialandgraph/property-graph-features/graph-server-and-client/graph-server-and-client-downloads.html

> **Note:**
>
> The versions are available at: Oracle Graph Server. See *UIM Compatibility Matrix* for the corresponding version of Oracle Graph PL/SQL Patch.

2. Extract **oracle-graph-plsql-<*version*>.zip** and open the **19c and above** folder.
3. Follow the instructions in the **readme.md** file to install Property Graph.

Kubernetes Storage Class

The Kubernetes Cluster administrator should create the Storage Class which can provision the persistent volumes dynamically.

2

About the Unified Inventory and Topology Toolkit

This chapter describes the components required for Unified Inventory and Topology.

Unified Inventory and Topology Toolkit

From Oracle Software Delivery Cloud, download the following:

- Oracle Communications Unified Inventory Management Cloud Native Toolkit
- Oracle Communications Unified Inventory Management Cloud Native Image Builder
- Oracle Communications Unified Inventory Management UTIA Image Builder
- (Optional) Oracle Communications Unified Inventory Management OHS Image Builder
- Oracle Communications Unified Inventory Management Common Toolkit

Perform the following tasks:

1. Copy the above downloaded archives into directory **workspace** and unzip the archives.
2. Export the unzipped path to the WORKSPACEDIR environment variable.
3. On Oracle Linux, where Kubernetes is hosted, download and extract the tar archive on each host. This host has a connectivity to the Kubernetes cluster.
4. Alternatively, on OKE, for an environment where Kubernetes is running, extract the contents of the tar archive (on each OKE client host). The OKE client host is the bastion host that is set up to communicate with the OKE cluster.

```
$ mkdir workspace
$ export WORKSPACEDIR=$(pwd)/workspace
//Untar UIM Builder
$ tar -xf $WORKSPACEDIR/uim-image-builder.tar.gz --directory workspace
//Untar UIMCN Toolkit
tar -xf $WORKSPACEDIR/uim-cntk.tar.gz --directory workspace
//Untar OHS Builder
tar -xf $WORKSPACEDIR/ohs-builder.tar.gz --directory workspace
//Untar UTIA Builder
$ tar -xf $WORKSPACEDIR/unified-topology-builder.tar.gz --directory
workspace
//Untar Common Toolkit
$ tar -xf $WORKSPACEDIR/common-cntk.tar.gz --directory workspace
$ export COMMON_CNTK=$WORKSPACEDIR/common-cntk
```

Assembling the Specifications

To assemble the specifications:

1. Create a directory (either in local machine or version control system where the deployment pipelines are available) to maintain the specification files needed to deploy the service. Export the directory to SPEC_PATH environment variable.
2. Copy the Strimzi Operator deployment specification file (strimzi-operator-override-values.yaml) to your $SPEC_PATH/<STRIMZI_PROJECT>.

   ```
   cp $COMMON_CNTK/samples/strimzi-operator-override-values.yaml $SPEC_PATH/
   <STRIMZI_PROJECT>/strimzi-operator-override-values.yaml
   ```

3. Copy the Micro Services deployment application specification file (applications.yaml) to your $SPEC_PATH/<PROJECT>/<INSTANCE>.

   ```
   cp $COMMON_CNTK/samples/applications.yaml $SPEC_PATH/<PROJECT>/<INSTANCE>/
   applications.yaml
   ```

4. Copy the Micro Services database specification file (database.yaml) to your $SPEC_PATH/<PROJECT>/<INSTANCE>.

   ```
   cp $COMMON_CNTK/samples/database.yaml $SPEC_PATH/<PROJECT>/<INSTANCE>/
   database.yaml
   cp $COMMON_CNTK/samples/appications-dev.yaml $SPEC_PATH/<PROJECT>/
   <INSTANCE>/applications-dev.yaml
   ```

5. Copy other specification files as required:
   - Persistent volumes and persistent volume claims files from $COMMON_CNTK/samples/nfs
   - Role and role bindings from $COMMON_CNTK/samples/rbac
   - Credential files from $COMMON_CNTK/samples/credentials

Image Builders

The following image builders are required to build the corresponding services for an end-to-end integrated environment:

- UIM Image Builder: This includes `archive uim-image-builder.tar.gz`, which is required to build UIM, UIM DB Installer Images. See **Creating the UIM Cloud Native Images** in *UIM Cloud Native Deployment Guide* for more information.
- (Optional) OHS Builder: This includes `ohs-builder.tar.gz`, required to build OHS image. See "Building the OHS Image" for more information.
- UTIA Builder: This includes `unified-topology-builder.tar.gz`, required to build Unified Topology API, Unified Topology UI, Unified Topology PGX, Unified Topology Consumer, and the Unified Topology DB Installer images.

All builder toolkits include manifest files and scripts to build the images.

About the Manifest File

A manifest file can be found in directory path $WORKSPACEDIR/<*service-builder*>/bin/<*service*>_manifest.yaml. The manifest file describes the input that goes into the *service* images. It is consumed by the image build process. The default configuration in the latest manifest file provides all necessary components for creating the *service* images easily. *A service* can be OHS, UTIA, or UIM.

You can also customize the manifest file. This enables you to:

- Specify any Linux image as the base, as long as it is a binary and is compatible with Oracle Linux.
- Upgrade the Oracle Enterprise Linux version to a newer version to uptake a quarterly CPU.
- Upgrade the JDK version to a newer JDK version to uptake a quarterly CPU.
- Choose a different **userid** and **groupid** for **oracle:oracle user:group** that the image specifies. The default is 1000:1000.

> **Note:**
>
> The `schemaVersion` and `date` parameters are maintained by Oracle. Do not modify these parameters. Version numbers provided here are only examples. The manifest file specifies the actual versions that Oracle recommends.

There are various sections in the manifest file such as:

- **Service Base Image**: The Service Base image is a necessary building block of the final *service* container images. However, it is not required by the *service* to create or manage any service instances.
  Linux parameter: The `Linux` parameter specifies the base Linux image to be used as the base Docker or Podman image. The version is the two-digit version from **/etc/redhat-release**:

  ```
  linux:
      vendor: Oracle
      version: 8-slim
      image: <container>/os/oraclelinux:8-slim
  ```

  The vendor and the version details are used for validating while an image is being built and for querying at run-time.

  > **Note:**
  >
  > To troubleshoot issues, Oracle support requires you to provide these details in the manifest file used to build the image.

- The `userGroup` parameter that specifies the default **userId** and **groupId**:

  ```
  userGroup:
    username: <username>
    userid: <userID>
    groupname: <groupname>
    groupid: <groupID>
  ```

- The `jdk` parameter that specifies the JDK vendor, version, and the staging path:

  ```
  jdk:
      vendor: Oracle
      version: <jdk_version>
  ```

```
    path: $CN_BUILDER_STAGING/downloads/java/jdk-<jdk_version>_linux-
x64_bin.tar.gz
```

- The Tomcat parameter specifies the Tomcat version and its staging path.

> **Note:**
>
> This is applicable only for the UTIA service.

```
tomcat:
  version: <tomcat_version>
  path: $CN_BUILDER_STAGING/downloads/tomcat/tomcat-<tomcat_version>.tar.gz
```

- A serviceImage parameter, where **tag** is the tag name of the *service* image.

```
serviceImage:
  tag: latest
```

> **Note:**
>
> See *UIM Compatibility Matrix* for software versions.

Deployment Toolkits

The following toolkits are required to deploy the services for an end-to-end integrated environment:

- **UIM Cloud Native toolkit**: Includes uim-cntk.tar.gz file that is required to deploy UIM in cloud native environment. See **Creating a Basic UIM Cloud Native Instance** in *UIM Cloud Native Deployment Guide*, for more information.
- **Common Cloud Native toolkit**: Includes common-cntk.tar.gz file that is required to deploy the OAM (optional), UTIA, and Message Bus services in the cloud native environment.

Common Cloud Native Toolkit

The Common cloud native toolkit (Common CNTK) includes:

- Helm charts to manage the UTIA, Common Authentication, and Message Bus services.
- Scripts to manage secrets for the services.
- Scripts to manage schemas for the services.
- Scripts to create, update, and delete the UTIA and Message Bus services.
- Scripts to create and delete the Common Authentication service.
- Sample pv and pvc yaml files to create persistent volumes.
- Sample charts to install Traefik.
- Scripts to register and un-register the namespaces with Traefik and Strimzi operator.

- The **applications.yaml** and, **database.yaml** files that provide the required configuration for the services which can be used for a production environment.
- The **applications-dev.yaml** file that contains the required configuration for the services which can be used for a development environment.
- The **strimzi-operator-override-values.yaml** file that enables you to override the configuration for deploying strimzi operator which is used for message bus service.

The **applications.yaml** and **database.yaml** files have common values that are applicable for all services in Common CNTK along with the values that are applicable for specific services.

For customized configurations to override the default values, update the values under the specific application sections in $SPEC_PATH/<PROJECT>/<INSTANCE>/applications.yaml.

While executing the scripts, the *project* and *instance* values should be provided, where *project* indicates the namespace of the Kubernetes environment where the service is deployed and *instance* is the identifier of the corresponding *service* instance, if multiples instances are created within the same namespace.

> **Note:**
>
> As multiple instances of Message Bus cannot exist in the same namespace, only one instance is created for all services within the same namespace.

While creating a basic instance for all these services, the project name is considered as **sr** and the instance name is considered as **quick**.

> **Note:**
>
> - Project and Instance names must not contain any special characters.
> - There are common values specified in the **applications.yaml** and **database.yaml** files for the services. To override the common value user can specify that value under the chart name of a service. If the value under the chart is empty, then common value is considered.

Deploying the Services

You must deploy and configure all services in the following sequence:

1. (Optional) Deploy Authentication Service (OAM along with OHS).

   > **Note:**
   >
   > Authentication service is only needed for deployment if you do not have any Identity Provider that supports SAML 2.0 and OIDC or OAuth 2.0 protocols.

2. Deploy Message Bus.
3. Deploy UIM (traditional or cloud native).

4. Configure Traditional UIM with Message Bus and UTIA, and restart UIM. See **Setting System Properties** in *UIM System Administrator's Guide*, for more information.
5. Configure OAM for UTIA client creation.
6. Deploy UTIA.

> **Note:**
>
> Ensure that each individual service is deployed successfully and verified in the above mentioned order as there are dependencies between these services. Ensure that for production instance, for High Availability, the Message Bus is set up with at least 3 replicas for kafka-cluster.

Setting Up Prometheus and Grafana

Message Bus has been tested with Prometheus and Grafana server installed and configured using the Helm charts.

- Prometheus Community is available at https://prometheus-community.github.io/helm-charts and uses the **prometheus-community/prometheus** chart.
- Grafana Community is available at https://grafana.github.io/helm-charts and uses the **grafana/grafana** chart.

Setting Up Elastic Stack

To set up Elastic Stack:

1. Install Elasticsearch and Kibana using the following commands:

```
#Install elasticsearch and kibana . It might take time to download images
from docker hub.
kubectl apply -f $COMMON_CNTK/samples/charts/elasticsearch-and-kibana/
elasticsearch_and_kibana.yaml

#Check if services are running, append namespace if deployment is other
than default like:- kubectl get services --all-namespaces
kubectl get services

Access kibana dashboard

Method 1 - kubectl get svc ( will return all the services  , append
namespace if deployment is other than default like:- kubectl get services
--all-namespaces)

Ex- elasticsearch    ClusterIP   10.96.190.99    <none>        9200/
TCP,9300/TCP   113d
    kibana           NodePort    10.100.198.88   <none>
5601:31794/TCP      113d

Kibana service nodeport at port 31794 is created
```

```
Now access kibana dashboard using url - http://<IP address of
VM>:<nodeport>/
```

2. Run the following command to create a namespace ensuring that it does not already exist.

```
kubectl get namespaces
export FLUENTD_NS=fluentd
kubectl create namespace $FLUENTD_NS
```

3. Update $COMMON_CNTK/samples/charts/fluentd/values.yaml with Elastic Search Host and Port.

```
elasticSearch:
  host: "elasticSearchHost"
  port: "elasticSearchPort"
```

For example:

```
elasticSearch:
  host: "elasticsearch.default.svc.cluster.local"
  port: "9200"
```

4. Modify the Fluentd image resources if required.

```
image: fluent/fluentd-kubernetes-daemonset:v1-debian-elasticsearch
   resources:
     limits:
     memory: 200Mi
   requests:
     cpu: 100m
     memory: 200Mi
```

5. Run the following commands to install fluentd-logging using the **$COMMON_CNTK/samples/charts/fluentd/values.yaml** file in the samples:

```
helm install fluentd-logging $COMMON_CNTK/samples/charts/fluentd -
n $FLUENTD_NS --values $COMMON_CNTK/samples/charts/fluentd/values.yaml \
--set namespace=$FLUENTD_NS \
--atomic --timeout 800s
```

6. Run the following command to upgrade fluentd-logging:

```
helm upgrade fluentd-logging $COMMON_CNTK/samples/charts/fluentd -
n $FLUENTD_NS --values $COMMON_CNTK/samples/charts/fluentd/values.yaml \
   --set namespace=$FLUENTD_NS \
   --atomic --timeout 800s
```

7. Run the following command to uninstall fluentd-logging:

```
helm delete fluentd-logging -n $FLUENTD_NS
```

8. Use 'fluentd_logging-YYYY.MM.DD' (default index configuration) index pattern in Kibana to check the logs.

Visualize logs in Kibana

To visualize logs in Kibana:

1. Navigate to Kibana dashboard (http://<IP address of VM>:<nodeport>/).
2. Create Index pattern (fluentd_looging-YYYY.MM.DD).
3. Click on Discover.

Setting Up OpenSearch

The Common CNTK has a sample that provides deployment instructions for OpenSearch on Kubernetes cluster using Helm charts. For more information, see https://opensearch.org/docs/latest/install-and-configure/install-opensearch/helm/

Create Kubernetes namespace to install OpenSearch and export it to the environment variable as follows:

```
Sample: export OPENSEARCH_NS=monitoring
```

Installing OpenSearch

Install OpenSearch as follows:

```
#Export the kubernetes namespace to be used for OpenSearch installation
export OPENSEARCH_NS=<kubernetes namespace>
export COMMON_CNTK=<path to common cntk>

#Install OpenSearch
helm install os-engine opensearch/opensearch --values=$COMMON_CNTK/samples/
charts/opensearch/os_engine_values.yaml --namespace=$OPENSEARCH_NS

#Install OpenSearch Dashboard
helm install os-board opensearch/opensearch-dashboards --values=$COMMON_CNTK/
samples/charts/opensearch/os_board_values.yaml --namespace=$OPENSEARCH_NS

#Accessing Dashboard
export NODE_PORT=$(kubectl get --namespace $OPENSEARCH_NS -o
jsonpath="{.spec.ports[0].nodePort}" services os-board-opensearch-dashboards)
export NODE_IP=$(kubectl get nodes --namespace $OPENSEARCH_NS-o
jsonpath="{.items[0].status.addresses[0].address}")
echo http://$NODE_IP:$NODE_PORT
```

Installing FluentD

Update the $COMMON_CNTK/samples/charts/fluentd/template/fluentd-config-map.yaml file with OpenSearch details such as type, host, port, scheme, user, password, and ssl_verify as follows:

```
#Export the kubernetes namespace to be used for OpenSearch installation
helm install fluentd-logging $COMMON_CNTK/samples/charts/fluentd --
values $COMMON_CNTK/samples/charts/fluentd/values.yaml --set
namespace=$OPENSEARCH_NS --atomic --timeout 800s
```

Accessing OpenSearch Dashboard

Access the OpenSearch dashboard using **nodeport** of the OpenSearch dashboard service in the namespace. Find and create index pattern with **fluentd_logging-***.

Uninstalling OpenSearch

Uninstall OpenSearch as follows:

```
helm uninstall os-board --namespace=$OPENSEARCH_NS
helm uninstall os-engine --namespace=$OPENSEARCH_NS
helm uninstall fluentd-logging --namespace=$OPENSEARCH_NS
```

Adding Common OAuth Secret and ConfigMap

To add `COMMON OAUTH` secret and `ConfigMap`:

1. Run the following command to create or update truststore by passing Identity Provider SSL certificate:

   ```
   keytool -importcert -v -alias <param> -file <path to IDP cert file> -
   keystore <truststorename>.jks -storepass <password>
   ```

   A sample is as follows:

   ```
   keytool -importcert -v -alias idpcert -file identityprovidercert.pem -
   keystore truststore.jks -storepass ****
   ```

   > **Note:**
   >
   > You must add the corresponding certificates for UIM and Identity Providers. If the Identity Provider and UIM certificates are not common, add both in the same truststore.

2. Run the following script to create the OAuth configuration as secrets and ConfigMap:

   ```
   $COMMON_CNTK/scripts/manage-app-credentials.sh -p sr -i quick -
   f $SPEC_PATH/sr/quick/applications.yaml create oauthConfig
   ```

   Enter the values as prompted:

   ```
   Provide Oauth credentials  for   'sr-quick'   ...
   Client Id: topologyClient #Provide Client ID
   Client Secret: xxxxx #Provide Client Secret
   Client Scope: <oauth-client-scope>  (if scope is not configured for oidc-
   client keep blank)
   Client Audience: <oauth-client-audience> (if audience not configured for
   oidc-client keep blank)
   Token Endpoint Uri: https://<instance>.<project>.ohs.<oam-host-
   suffix>:<port>/oauth2/rest/token #Provide oauth token endpoint URI
   Valid Issue Uri: https:// <instance>.<project>.ohs .<oam-host-
   ```

```
suffix>:<port>/oauth2 #Provide oauth valid issue URI
Introspection Endpoint Uri: https:// <instance>.<project>.ohs .<oam-host-
suffix>:<port> /oauth2/rest/token/introspect #Provide Oauth Introspection
Endpoint URI
JWKS Endpoint Uri: https://<instance>.<project>.ohs.<oam-host-
suffix>:<port>/oauth2/rest/security #Provide JWKS Endpoint URI

Provide Truststore details ...
Certificate File Path (ex. oamcert.pem): ./commoncert.pem    #provide
Certificate file path
Truststore File Path (ex. truststore.jks): ./commontrust.jks   #provide
Truststore file path
Truststore Password: xxxx  #provide Truststore password
```

Sample for IDCS is as follows:

```
Provide Oauth credentials for 'sr-quick' ...
Client Id: e6e0b2c6c3a845709bc51b561e0f008c
Client Secret: xxxx-xxxx-xxxx-xxxx
Client Scope: https://quick.sr.topology.uim.org:30443/first_scope
Client Audience: https://quick.sr.topology.uim.org:30443/
Token Endpoint Uri: https://<IDCS URL>:443/oauth2/v1/token
Valid Issue Uri: https://identity.oraclecloud.com/
Introspection Endpoint Uri: https://<IDCS URL>:443/oauth2/v1/introspect
JWKS Endpoint Uri: https://<IDCS URL>:443/admin/v1/SigningCert/jwk
Provide Truststore details ...
Certificate File Path (ex. oamcert.pem): ./identity-pint-oc9qadev-com.pem
Truststore File Path (ex. truststore.jks): ./truststore.jks
Truststore Password: xxxxx  #provide Truststore password
```

3. Verify the following:

```
$kubectl get secret -n sr
sr-quick-oauth-credentials

$kubectl get cm -n sr
sr-quick-oauth-config-cm
```

> **Note:**
>
> The oauthConfig secret is used by both messaging-bus and unified topology applications. If you are creating them in different namespaces or instances, you need to create this secret in both namespaces or instances.

3
Deploying the Common Authentication Service

This chapter describes how to optionally deploy and manage the Common Authentication service.

Building the OHS Image

To build OHS image:

1. Go to **WORKSPACEDIR** that is created in "Unified Inventory and Topology Toolkit".
2. Download **V983369-01.zip**: **Oracle Fusion Middleware 12c (12.2.1.4.0) HTTP Server for Linux x86-64, 1.9 GB** file from Oracle E-Delivery by searching for the file from **Oracle HTTP Server** 12.2.1.4.0 for (Linux x86-64) and copy them to the **$WORKSPACEDIR/ohs-builder/staging/downloads/** folder.
3. Modify `ohsBaseImage.package.path` in **$WORKSPACEDIR/ohs-builder/bin/ohs_manifest.yaml** with the filename of the downloaded OHS archive file.
4. Download `jdk-<version>_linux-x64_bin.tar.gz` and copy to the **$WORKSPACEDIR/ohs-builder/staging/downloads/java** folder.

   > **Note:**
   >
   > See *UIM Compatibility Matrix* for the latest versions of software.

5. Modify the `ohsBaseImage.jdk.path` in **$WORKSPACEDIR/ohs-builder/bin/ohs_manifest.yaml** file with the name of the downloaded JDK file.
6. Run `build-all-images.sh` in **bin** directory to build all images on OHS.

Deploying OAM along with OHS for Authentication Service

Before deploying OAM using the COMMON CNTK scripts, ensure the following:

- WebLogic Operator is deployed and configured as per UIM_CNTK. See **Setting Up Oracle WebLogic Server Kubernetes Operator** in *UIM Cloud Native Deployment Guide* for more information.
- Namespace is registered with WebLogic Operator using the UIM_CNTK script. See **Registering the Namespace** in *UIM Cloud Native Deployment Guide* for more information.
- Traefik (ingress-based) load balancer is installed as per UIM_CNTK script. See **Installing the Traefik Container Image** in *UIM Cloud Native Deployment Guide* for more information.
- Pull the Oracle Access Manager Image or latest cpu image from Oracle Container Registry as follows:
  1. Launch a browser and access the Oracle Container Registry.
  2. Click **Sign In** and enter your username and password.

3. In the **Search** field, enter **Oracle Access Manager** and press **Enter**.
4. Click **oam_cpu** for the latest CPU patch image of Oracle Access Manager.
5. In the **Terms and Conditions** box, select the language as **English**.
6. Click **Continue** and accept **Terms and Restrictions**.
7. On your Podman environment, log in to the Oracle Container Registry and enter your Oracle SSO username and password when prompted:

```
$ podman login container-registry.oracle.com
Username: <username>
Password: <password>
$ podman pull <oam-cpu-image-name>
```

8. Pull the Oracle Access Manager Image or latest CPU image from Oracle Container Registry as follows.
   For example: Use the following command to pull OAM CPU image from Oracle Container Registry:

```
docker pull container-registry.oracle.com/middleware/oam_cpu:12.2.1.4-
jdk8-ol7-221014
```

- Download **Oracle Communications Unified Inventory Management Common Toolkit** from Oracle Software Delivery Cloud.

Deploying OAM Using Common Cloud Native Toolkit Scripts

To deploy OAM using COMMON_CNTK scripts:

1. Go to the **$WORKSPACEDIR/common_cntk** folder created in Unified Inventory and Topology Toolkit and export the path to a variable COMMON_CNTK. See "Unified Inventory and Topology Toolkit" for more information.
2. Modify the parameters in the `$SPEC_PATH/sr/quick/applications.yaml` file as follows:
   - **inventory.host**: Provide the inventory host IP or address where UIM traditional application is installed. This is a mandatory parameter. For UIM cloud native instance, the value is: <uimproject>-<uiminstance>-cluster-uimcluster.<uimproject>.svc.cluster.local
   - **inventory.port**: Provide the inventory host port where the UIM on-perm is installed. This is a mandatory parameter. For UIM cloud native instance, the value is 8502.
   - **inventory.isSSL**: If traditional UIM has the SSL port used, change the value to **true**, for Cloud Native Inventory always false.
   - **imagePullSecret**: Provide the Kubernetes secret name containing the Docker secrets to pull images. This is a mandatory parameter. This secret should be accessible, which means that it must be created in the same namespace as OAM.
   - **persistentVolumeClaimName**: Provide the existing pvc name for storage of OAM domain. This is a mandatory parameter.
   - **hostSuffix**: By default it is .uim.org.
   - **loadBalancerPort**: The load balancer port exposed by Traefik or external load balancer. Enter the Secure/SSL port.

- **gcLogs**: To enable GC logs for OAM, set **enabled** to **true** and configure the number of files and size of each file. You can uncomment values inside **oam-server** to override common values for **gcLogs**.
  For example:

```
gcLogs:
  enabled: true
  fileSize: 10M
  noOfFiles: 10
```

- **tls.enabled**: Flag to enable tls or ssl. By default, it is true. If **true**, create the certificate and the key mentioned in next step. Oracle recommends not to disable SSL in production environment.

3. If SSL is enabled that is, `tls.enabled` is **true**, create the certificate as follows:

   a. Create `certs` folder in $COMMON_CNTK.

   b. Copy your signed certificate and key into `certs` folder by renaming the certificate name as `commoncert.pem` and renaming the key file name as `commonkey.pem`.

> **Note:**
>
> OAM supports wild card certificates. Your certificate can be updated with the `*.<hostSuffix>`. By default, the `hostSuffix` value is `uim.org` from **applications.yaml**. See "Using Wild Card Certificates" for more information.

   c. (Optional) Run the following command to create Single Certificate and Key for OAM, messaging-bus, UIM, and UTIA:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -
keyout $COMMON_CNTK/certs/commonkey.pem -out $COMMON_CNTK/certs/
commoncert.pem -subj "/CN=<instance>.<project>.admin.uim.org" -
extensions san -config <(echo '[req]'; echo 'distinguished_name=req';
echo '[san]';echo 'subjectAltName=@alt_names'; \echo '[alt_names]'; \
echo 'DNS.1=<instance>.<project>.admin.uim.org'; \
echo 'DNS.2=<instance>.<project>.oam.uim.org'; \
echo 'DNS.3=<instance>.<project>.ohs.uim.org'; \
echo 'DNS.4=uim.org'; \
echo 'DNS.5=<instance>.<project>.topology.uim.org'; \
echo 'DNS.6=localhost'; \
echo 'DNS.7=svc.cluster.local'; \
echo 'DNS.8=<instance>.<project>.uim.org'; \
echo 'DNS.9=admin.<instance>.<project>.uim.org'; \
echo 'DNS.10=t3.<instance>.<project>.uim.org'; \
)
```

   An example for **project:sr** and **instance: quick**:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -
keyout $COMMON_CNTK/certs/commonkey.pem -out $COMMON_CNTK/certs/
commoncert.pem -subj "/CN=quick.sr.admin.uim.org" -extensions san -
config <(echo '[req]'; echo 'distinguished_name=req'; echo '[san]';echo
'subjectAltName=@alt_names'; \echo '[alt_names]'; \
```

```
echo 'DNS.1=quick.sr.admin.uim.org'; \
echo 'DNS.2=quick.sr.oam.uim.org'; \
echo 'DNS.3=quick.sr.ohs.uim.org'; \
echo 'DNS.4=uim.org'; \
echo 'DNS.5=quick.sr.topology.uim.org'; \
echo 'DNS.6=localhost'; \
echo 'DNS.7=svc.cluster.local'; \
echo 'DNS.8=quick.sr.uim.org'; \
echo 'DNS.9=admin.quick.sr.uim.org'; \
echo 'DNS.10=t3.quick.sr.uim.org'; \
)
```

> **Note:**
>
> Ensure that **commoncert.pem** and **commonkey.pem** files are present in the **$COMMON_CNTK/certs** folder.
>
> OAM, UIM, UTIA, and Message Bus support wildcard certificates. See "Using Wild Card Certificates" for more information.

d. (Optional) To generate your own self-signed certificates, see "Self-signed SSL Certificates".

4. Create the secrets for OAM as follows:

   a. Create the mandatory secrets according to the system prompts as follows:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p sr -i quick -
f $SPEC_PATH/sr/quick/applications.yaml -a oam create
database,wlsadmin,ingressTLS
Applications specified - oam

====create database,wlsadmin,ingressTLS secret for oam Application====
Provide Database credentials for 'sr-quick-oam' ...
OAM DB Admin(sys) Username: <PDB-ADMIN-USER>
OAM DB Admin(sys) Password: <PDB-ADMIN-PWD>
OAM Schema Username: <OAM_SCHEMA_USER>
OAM Schema Password: <OAM_SCHEMA_PWD>
OAM DB Host: <DB_HOSTNAME>
OAM DB Port: <DB_PORT>
OAM DB Service Name: <SERVICE-NAME>

Provide Weblogic Admin credentials for 'sr-quick-oam' ...
Weblogic Admin Username: <WL_ADMIN_USER>
Weblogic Admin Password: <WL_ADMIN_PWD>

Provide Ingress TLS Credentials for OAM application 'sr-quick-oam' ...
Ingress TLS Certificate Path (PEM file): $COMMON_CNTK/certs/
commoncert.pem
Ingress TLS Key file Path (PEM file): $COMMON_CNTK/certs/commonkey.pem

secret/sr-quick-oam-rcu-credentials created
secret/sr-quick-oam-wls-credentials created
secret/sr-quick-oam-ingress-tls-cert-secret created
```

```
Execution status of secrets for command - create:
OAM MICROSERVICE...........Ok
```

> **Note:**
>
> The RCU Schema password guideline specifies that a valid password must be specified. The password should be alpha numeric only and can contain the following special characters: **#** , _ . The password should not start with a number or a special character.
>
> `<OAM_SCHEMA_USER>` should be less 12 characters and `<OAM_SCHEMA_PWD>` is the RCU Schema password.

b. Ensure the following secrets are created:

- Database secret : Contains the details of OAM database schema. For example, `sr-quick-oam-rcu-credentials`.
- **wlsadmin** secret: Contains the credentials for WebLogic and **oamconsole**. For example, `sr-quick-oam-wls-credentials`.
- **ingressTLS**: Contains certificate and key for OAM. For example, `sr-quick-oam-ingress-tls-cert-secret`.

c. For traditional UIM, if SSL port is used, you must create additional **configmap** to pass the inventory certificate.

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p sr -i quick -
f $SPEC_PATH/sr/quick/applications.yaml -a oam create inventorySSL
Provide Inventory SSL Credentials for OAM application 'sr-quick-oam' ...
On-prem Inventory SSL Certificate Path (PEM file): <provide inventory
certificate>
```

5. Configure Ingress and Ingress Controller for OAM. See "Configuring Ingress and Ingress Controller for OAM" for more information.
6. Create schema by running the following commands to install OAM DB and ensure that database secret and image name for database.yaml are correct:

```
$COMMON_CNTK/scripts/install-database.sh -p sr -i quick -f $SPEC_PATH/sr/
quick/database.yaml -a oam -c 1
```

7. Create OAM by running the following command to install OAM and ensure that you updated applications.yaml file:

```
$COMMON_CNTK/scripts/create-applications.sh -p sr -i quick -
f $SPEC_PATH/sr/quick/applications.yaml -a oam
```

Using Wild Card Certificates

OAM, UIM, UTIA and Message Bus supports wildcard certificates. You can generate wildCard Certificates with the hostSuffix value provided in applications.yaml. The default is **uim.org**.

To use wild card certificates:

1. To create a self-signed wild card certificate, use the following command:

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout $COMMON_CNTK/
certs/wildcardkey.pem -out $COMMON_CNTK/certs/wildcardcert.pem -subj "/
CN=*.uim.org" -extensions san -config <(echo '[req]'; echo
'distinguished_name=req';
echo '[san]';echo 'subjectAltName=@alt_names'; \echo '[alt_names]'; \
echo 'DNS.1=*.uim.org'; \
)
```

2. Change the **subDomainNameSeperator** value from period (**.**) to hyphen (**-**) so that the incoming hostnames match the wild card DNS pattern.
Update the $SPEC_PATH/project/instance/applications.yaml file as follows:

```
#Uncooment and provide the value of subDomainNameSeparator, default is "."
#Value can be changed as "-" to match wild-card pattern of ssl
certificates.
#Example hostnames for "-" quick-sr-topology.uim.org
subDomainNameSeparator: "-"
```

3. If you have configured the above settings, the following are the hostnames to access OAM application for **project:sr**, **instance:quick**, and **hostSuffix: uim.org**:

```
oam-admin hostname: quick-sr-admin.uim.org
oam-ohs hostname:  quick-sr-ohs.uim.org
oam hostname: quick-sr-oam.uim.org
oam-policy hostname: quick-sr-policy.uim.org
```

Configuring Ingress and Ingress Controller for OAM

OAM supports standard Kubernetes ingress API and provides samples for integration. The following configuration provides the OAM required annotations for Nginx.

Any Ingress Controller that conforms to the standard Kubernetes Ingress API and supports annotations needed by OAM should work. Oracle does not certify any individual Ingress controllers to confirm this **generic** compatibility.

The annotations for OAM are:

- To use Annoation Base Generic Ingress Controller, update **applications.yaml** from **$SPEC_PATH/project/instance** as follows:

```
# Valid values are TRAEFIK, GENERIC
ingressController: "GENERIC"

ingress:
  className: nginx  ##provide ingressClassName value, default value for
nginx ingressController is nginx.
  annotations:
    nginx.ingress.kubernetes.io/affinity: "cookie"
    nginx.ingress.kubernetes.io/affinity-mode: "persistent"
    nginx.ingress.kubernetes.io/session-cookie-name: "nginxingresscookie"
    nginx.ingress.kubernetes.io/proxy-body-size: "50m"
    nginx.ingress.kubernetes.io/proxy-buffer-size: 64k
```

- To enable SSL, provide the following annotations in **applications.yaml** under **oam-server tag**:

```
oam-server:
  ingress:
    annotations:
      nginx.ingress.kubernetes.io/configuration-snippet: |
        more_clear_input_headers "WL-Proxy-Client-IP" "WL-Proxy-SSL" "X-
Custom-Request-Header" ;
        more_set_input_headers "X-Forwarded-Proto: https";
        more_set_input_headers "WL-Proxy-SSL: true";
        more_set_input_headers "IS_SSL: ssl";
```

- To use TRAEFIK Ingress Controller, update **applications.yaml** from **$SPEC_PATH/project/instance** as following:

```
# Valid values are TRAEFIK, GENERIC
ingressController: "TRAEFIK"
```

Upgrading OAM

To upgrade OAM, you can use following command:

```
$COMMON_CNTK/scripts/upgrade-applications.sh -p sr -i quick -f $SPEC_PATH/sr/
quick/applications.yaml -a oam
```

> **Note:**
>
> This upgrade command will restart OAM and OHS deployments. If you want to update ingressTLS, inventorySSL secrets or want to make any changes in **applications.yaml** for OAM, you can perform this operation.
>
> This command will not make any changes to OAM domain. To update domain, you need to uninstall OAM and recreate.

Uninstalling OAM

To uninstall OAM:

1. Delete OAM as follows:

```
$COMMON_CNTK/scripts/delete-applications.sh -p sr -i quick -
f $SPEC_PATH/sr/quick/applications.yaml -a oam
```

2. Delete OAM db schema as follows:

```
$COMMON_CNTK/scripts/install-database.sh -p sr -i quick -f $SPEC_PATH/sr/
quick/database.yaml -a oam -c 2
```

3. Run the file `$COMMON_CNTK/scripts/uninstall.sh`.

> **Note:**
>
> Ensure the domain folder and its contents on the **PV_SHARED_PATH** or **Path** `sharedDomainPath` on NFS are deleted after the uninstallation. That is, delete `<project>-<instance>-oam` and `<project>-<instance>-oam-ohs` folders.

Specifying the Proxy Settings

Enter the following proxy settings:

- In the browser, go to **network no-proxy settings** and include the ***<hostSuffix>** value from **$SPEC_PATH/sr/quick/applications.yaml**. By default, it is **.uim.org** that is, ***.uim.org**.
- In `/etc/hosts` the following may changed based on the `<instance>`, `<project>`, and `hostSuffix` values in **$SPEC_PATH/sr/quick/applications.yaml**.

etc/hosts:

```
<k8s cluster ip>  <instance>.<project>.oam.<hostSuffix>
<instance>.<project>.admin.<hostSuffix>
<instance>.<project>.policy.<hostSuffix> <instance>.<project>.ohs.<hostSuffix>
for example:
<k8s cluster ip> quick.sr.oam.uim.org quick.sr.admin.uim.org
quick.sr.policy.uim.org traefik.uim.org quick.sr.ohs.uim.org
```

Accessing the WebLogic Server Administration Console and the OAM Console

You need to complete the proxy settings for accessing the WebLogic Server Administration console and the OAM console. The credentials for accessing WebLogic console or OAM console are stored in the **wlsadmin** secret.

WebLogic Console:

```
https://<instance>.<project>.admin.<hostSuffix>:<loadBalancerPort>/console
```

For example:

```
https://quick.sr.admin.uim.org:30443/console
```

OAM Console:

```
https://<instance>.<project>.admin.<hostSuffix>:<loadBalancerPort>/oamconsole
```

For example:

```
https://quick.sr.admin.uim.org:30443/console
```

Configuring OAM

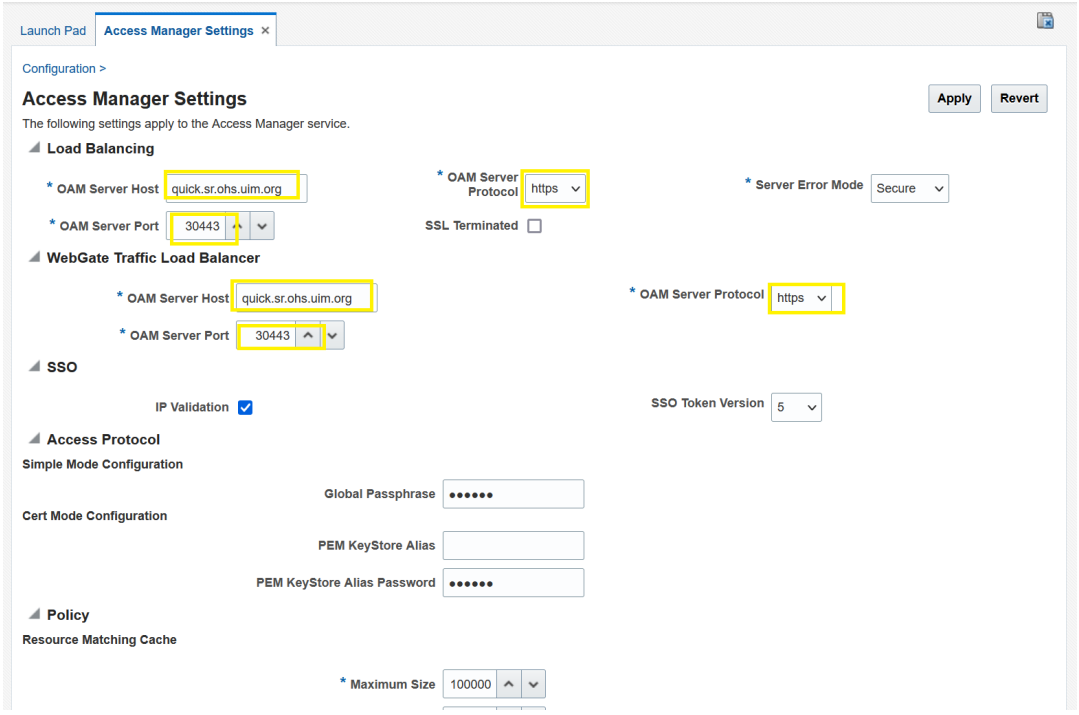
To configure OAM before using it for SSO authentication:

1. Log in to Oracle Access Management (OAM) Console:

   ```
   https://<instance>.<project>.admin.<hostSuffix>:<loadBalancerPort>/
   oamconsole
   ```

2. Click **Configuration** at the top right corner of the Console to show Configuration Launch Pad.
3. Click on **Available Services** and then click **Enable Service** for **OAuth and OpenIDConnect Service**.
4. From Configuration Launch Pad, select **Access Manager** from the **View** menu in the **Settings** section:
   a. Under **Load Balancing** and **WebGate Traffic Load Balancer**, modify `OAM Server Host` with `<instance>.<project>.ohs.<hostSuffix >`. The `hostSuffix` value is taken from **$SPEC_PATH/sr/quick/applications.yaml**. By default, it is .uim.org.
   b. Modify `OAM Server Protocol` to `https`.
   c. Modify `OAM Server Port` to `<loadBalancerPort>`. This value is from $SPEC_PATH/sr/quick/applications.yaml.
   d. Secure the load balancer port.
   e. Click **Apply** to save.

Figure 3-1 Access Manager Settings



5. From Configuration Launch Pad, select **User Identity Stores** to create an ID store for using the embedded LDAP of UIM:
   a. Click **Create** under the **IDS Profiles** section for creating an IDS profile.
   b. Specify **Name** as `UIMEmbeddedLDAP`.
   c. (Optional) Provide **Description**.
   d. Configure the Repository properties under **Repository**:
      i. Choose **Repository Options** by selecting **Create New**.
      ii. Provide **Directory Type** as `Weblogic Server Embedded LDAP`.
      iii. Provide **Host Name** as `<Inventory's AdminHost>` and **Port** as `<Inventory's AdminPort>` under **Hosts**.

> **Note:**
>
> In case of UIM Cloud Native Environment, provide AdminServer service name and port for `<Inventory's AdminHost>:<Inventory's AdminPort>` as `<uim-project>-<uim-instance>-admin:8501 (sample: sr-quick-admin:8501).`

      iv. If UIM onPrem admin server is SSL enabled, select **SSL Enabled**, for UIM Cloud Native environment not required.
      v. Provide the **Bind DN** as `cn=Admin`.
      vi. Specify **Bind Password** provided for the embedded LDAP in the WebLogic admin console. Ensure that the following steps are performed in WebLogic console where UIM is deployed. In the WebLogic Server admin console, change the credential for the embedded LDAP server as follows:

> **Note:**
>
> In case of UIM Cloud Native environment, enter your WebLogic password in the **Password** field.

         - Expand **Domain** > **Security** > **Embedded LDAP**.
         - In the **Credential** field, enter the new credential.
         - In the **Confirm Credential** field, enter the new credential again.
         - Click **Save**.
         - Reboot the WebLogic server.

      vii. Provide **Base DN** as follows:

```
ou=myrealm,dc=<inventory application domain name>
```

> **Note:**
>
> In case of UIM Cloud Native Environment, provide <inventory application domain name> as domain. On UIM CN, WebLogic domain name is set to `domain` by default.

e. Configure the user properties to configure the LDAP user object under **User** section:

   i. Provide **Base DN** as `ou=people,ou=myrealm,dc=<inventory application domain name>`.

   ii. Provide **Login ID Attribute** as `uid`.

f. Configure the Group properties to configure the LDAP group object under **Group** section:

g. Provide **Base DN** as `ou=groups,ou=myrealm,dc=<inventory application domain name>`.

h. Click **Test Connection** on the top-right corner to ensure the connection to embedded LDAP is successful.

i. Click **OK** to close the **Connection Status** dialog box.

j. Click **Create** to create IDS profile.
Entries with the profile name are displayed in the IDS Profiles and IDS Repositories table.

k. Click **Sync IDS Profiles** button on right side of **OAM ID Stores** section to see the *IDSPROFILE-UIMEmbeddedLDAP* entry displayed under OAM ID Stores table.

6. Click **Application Security** at the top right corner of the Console to show the Application Security Launch Pad.

7. Click **Agents** and then **Search** to show the **UnifiedWebgate** in the table.

8. Select **UnifiedWebgate** from the table and click **Edit** to modify the Webgate settings:

   a. Modify **Logout Redirect URL** as:

   ```
   https://<instance>.<project>.ohs.<hostSuffix>:<loadBalancerPort/oam/server/logout
   ```

   b. Modify the **Access Server** and **Host Name** under **Primary Server List** as `Other` and `<domainUID> -oam-server1 ' where domainUID is the <project>-<instance>-oam`. By default, it is `sr-quick-oam-oam-server1`.

   c. Click **Apply** to save.

9. From the Application Security Launch Pad, select **Authentication Modules** from **Plug-ins** to create 'UIM Embedded LDAP Module' authentication module.

   a. Click **Create LDAP Authentication Module** in the **Create** dropdown, under **Search Results** section.

   b. Provide **Name** as **UIM Embedded LDAP Module**.

   c. Choose **User Identity Store** as **IDSPROFILE-UIMEmbeddedLDAP** that is created above.

   d. Click **Apply** to save.

10. From the Application Security Launch Pad, select **Authentication schemas** from **Access Manager** to create 'UIM Embedded LDAP Schema' authentication schema.

a. Click **Create** under **Search Results** section.
b. Provide **Name** as **UIM Embedded LDAP Schema**.
c. Provide **Description** as **UIM Embedded LDAP Schema**.
d. Modify the **Authentication Level** as **2**.
e. Provide **Challenge Method** as **FORM**.
f. Provide **Challenge Redirect URL** as **/oam/server/**.
g. Choose **Authentication Module** as **UIM Embedded LDAP Module**.
h. Provide **Challenge URL** as **/login.jsp**.
i. Choose **Context Type** as **customwar**.
j. Provide **Context Value** as **/customConsent**.
k. Click **Apply** to save.

11. From the Application Security Launch Pad, select **Application Domains** from **Access Manager** to edit **UnifiedWebgate** application domain.
    a. Click **Search** to show the **UnifiedWebgate** in the table.
    b. Select **UnifiedWebgate** from the table and click **Edit** to modify the Application Domain settings.
    c. Select **Authentication Policies** tab and select the **Protected Resource Policy** table item.
    d. Click **Edit** button to open **Protected Resource Policy** authentication policy settings.
    e. Choose **Authentication Schema** as **UIM Embedded LDAP Schema** from the drop down.
    f. Click **Apply** to save.

Configuring OAuth Service Settings

Complete the proxy settings as mentioned in the above section.

Ensure environment variable **NO_PROXY** is set with `<hostSuffix>`.

Run the following commands from the machine on which the proxy settings are done:

```
export CREDS=`echo -n "<OAM_Domain_Username>:<password>" | base64 -w 0`
export OAMHOST=<instance>.<project>.admin.<hostSuffix> (example,
quick.sr.admin.uim.org)
export OAMPORT=<loadBalancerPort> (the value provided in $SPEC_PATH/sr/quick/
applications.yaml)
```

Creating an OAuth Identity Domain

Run the following curl statement to create the `UnifiedIdDomain` identity domain with custom-consent enabled and using `IDSPROFILE-UIMEmbeddedLDAP` as the identity provider:

```
curl -i -H "Content-Type: application/json" -H "Authorization:Basic ${CREDS}"
--cacert $COMMON_CNTK/certs/commoncert.pem --noproxy $NO_PROXY --request POST
https://${OAMHOST}:${OAMPORT}/oam/services/rest/ssa/api/v1/oauthpolicyadmin/
oauthidentitydomain -d '{"consentPageURL":"/customConsent/
```

```
customConsent.jsp","issueTLSClientCertificateBoundAccessTokens":false,"tokenSe
ttings":
[{"tokenType":"ACCESS_TOKEN","tokenExpiry":3600,"lifeCycleEnabled":false,"refr
eshTokenEnabled":true,"refreshTokenExpiry":86400,"refreshTokenLifeCycleEnabled
":false},
{"tokenType":"AUTHZ_CODE","tokenExpiry":3600,"lifeCycleEnabled":false,"refresh
TokenEnabled":true,"refreshTokenExpiry":86400,"refreshTokenLifeCycleEnabled":f
alse},
{"tokenType":"SSO_LINK_TOKEN","tokenExpiry":3600,"lifeCycleEnabled":false,"ref
reshTokenEnabled":true,"refreshTokenExpiry":86400,"refreshTokenLifeCycleEnable
d":false}],"customAttrs":"{\"allowedCustomPlugins\":\"OAuthCustomClaimsPlugin\
"}","name":"UnifiedIdDomain","description":"Unified Identity
Domain","identityProvider":"IDSPROFILE-UIMEmbeddedLDAP","errorPageURL":"/oam/
pages/servererror.jsp","keyPairRolloverDurationInHours":48}'
```

Creating a Resource

Run the following curl statement to create `UnifiedRserver` resource with default scope as `Info`:

```
curl -i -H "Content-Type: application/json" -H "Authorization:Basic ${CREDS}"
--cacert $COMMON_CNTK/certs/commoncert.pem --noproxy $NO_PROXY --request POST
https://${OAMHOST}:${OAMPORT}/oam/services/rest/ssa/api/v1/oauthpolicyadmin/
application -d '{"tokenAttributes":
[],"resServerType":"CUSTOM_RESOURCE_SERVER","resourceServerNameSpacePrefix":"U
nifiedRserver.","name":"UnifiedRserver","description":"Unified Resource
Server","audienceClaim":null,"scopes":
[{"scopeName":"Info","description":"null"},
{"scopeName":"DefaultScope","description":"DefaultScope"}],"idDomain":"Unified
IdDomain","resourceServerId":"1f50f6f4-06a9-4d1b-8347-bc5672a12e56"}'
```

Creating a Client

Run the curl statement to create `topologyClient` client.

The following is an example for creating a client with **<project>** as *sr* and **<instance>** as *quick*:

```
curl -i -H "Content-Type: application/json" -H "Authorization:Basic ${CREDS}"
--cacert $COMMON_CNTK/certs/commoncert.pem --noproxy $NO_PROXY --request POST
https://${OAMHOST}:${OAMPORT}/oam/services/rest/ssa/api/v1/oauthpolicyadmin/
client -d
'{"clientType":"CONFIDENTIAL_CLIENT","issueTLSClientCertificateBoundAccessToke
ns":false,"name":"topologyClient","grantTypes":
["PASSWORD","CLIENT_CREDENTIALS","JWT_BEARER","REFRESH_TOKEN","AUTHORIZATION_C
ODE"],"description":"null","attributes":
[{"attrName":"customeAttr1","attrValue":"CustomValue","attrType":"STATIC"}],"i
d":"topologyClient","secret":"<secret>","scopes":
["UnifiedRserver.Info"],"defaultScope":"UnifiedRserver.Info","redirectURIs":
[{"url":"https://quick.sr.topology.uim.org:30443/topology","isHttps":true},
{"url":"https://quick.sr.topology.uim.org:30443/redirect/unified-topology-
ui","isHttps":true}],"idDomain":"UnifiedIdDomain"}'
```

Add topology service specific redirect URLs under **redirectURIs** attribute in json data and update <secret>:

- For Topology-API:

```
redirect-uri: "https://<instance>.<project>.topology.<hostSuffix>:<port>/
topology"
```

- For Topology-UI:

```
redirect-uri: https://<instance>.<project>.topology.<hostSuffix>:<port>/
redirect/unified-topology-ui
```

> **Note:**
>
> If an external load balancer is used with a default port of **80** or **443**, you do not mention ports in redirect URIs. In that case, redirect URIs will be as follows:
>
> ```
> "redirectURIs":[{"url":"https://quick.sr.topology.uim.org/
> topology","isHttps":true},
> {"url":"https://quick.sr.topology.uim.org/redirect/unified-topology-
> ui","isHttps":true}]
> ```

Debugging and Troubleshooting

The following are some common issues.

Unable to create Domain or Admin Server is not coming up

To troubleshoot the issue:

1. Check if a folder with the domain name already exists at the **persistentVolumeClaim** location.
   If there is a **Domain Exists** error, the following message appears:

   ```
   The domain will be created using the script /u01/weblogic/create-domain-
   script.sh
   ERROR: The create domain job will not overwrite an existing domain. The
   domain folder /u01/oracle/user_projects/domains/accessdomain already exists
   ```

2. Ensure RCU schema creation is successful.

   ```
   kubectl -n <NAMESPACE> get pods
   ```

3. Check the logs of `<project>-<instance>-oam-dbschema` (kubectl -n <NAMESPACE>), which ends with `Repository Creation Utility - Create : Operation Completed` line.
4. Check the logs of `<project>-<instance>-oam-create-infra-domain-job-<podsuffix>`.

To resolve the issue:

1. If a folder with the same domain name already exists, delete the domain folders (<project>-<instance>-oam and <project>-<instance>-oam-ohs) and its contents.
2. Uninstall OAM. See Uninstalling OAM for more information.
3. If RCU Schema creation is not successful, then check the rcuDatabaseURL and rcuSchemaPrefix values provided.

> **Note:**
>
> Same `rcuSchemaPrefix` value cannot be used for different domains with in the same database.

4. Resolve the database issues and run the scripts again.
5. Resolve the errors appeared in the logs of `<project>-<instance>-oam-create-infra-domain-job-<podsuffix>`:
   a. If you see `mkdir: cannot create directory ... : Permission denied` error, then ensure the **PVC/sharedDomainPath** has permissions. For example: `chmod 777 /scratch/shared`.
   b. If there are no errors or exceptions in logs, ensure the <NAMESPACE> is registered with the WebLogic operator as mentioned in prerequisites for running scripts.
6. Before running the scripts again, remove the Helm releases that are partially installed as follows to get the helm releases in the namespace:

```
helm ls -n <NAMESPACE> -
```

Unable to Access OAM Console

Unable to access OAM Console using: `https://admin.<DOMAIN_NAME><hostSuffix>:<loadBalancerPort>/oamconsole`

To troubleshoot the issue:

- Ensure the OHS service is up and running the following commands:

```
kubectl -n <NAMESPACE> logs <project>-<instance>-oam-ohs-<podSuffix>
```

- Ensure the `loadBalancerPort` is correct and provide secure port if SSL is enabled.
- Ensure proxy settings are done.

To resolve the issue, identify and uninstall the failed pod as follows:

1. Check if there are any pods that are failed or in the **Error** state using:

```
kubectl -n <NAMESPACE> get pods
```

2. Check the release of the pods using the following Helm command:

```
helm ls -n <NAMESPACE>
```

3. If RCU Schema creation has failed, uninstall `<project>-<instance>-oam-dbschema` release using:

   ```
   helm -n <NAMESPACE> uninstall <project>-<instance>-oam-dbschema
   ```

4. If OAM domain creation has failed, uninstall `<project>-<instance>-oam-createdomain` release using:

   ```
   helm -n <NAMESPACE> uninstall <project>-<instance>-oam-createdomain
   ```

5. Run `$COMMON_CNTK/scripts/delete_applications.sh -p <project> -i <instance> -f $SPEC_PATH/sr/quick/applications.yaml -a oam` then ensure the `<DOMAIN_NAME>` folder and `<DOMAIN_NAME>-ohs` folder (if exists) from the PVC/`sharedDomainPath` is deleted.

Inventory UI is not appearing after successful login

To troubleshoot the issue, check if you have the credentials to view UIM and check the logs ot Topology-UI service.

The following error appears if you have recreated UIM.

Failure of Web Server bridge:

No back-end server available for connection: timed out after 10 seconds or idempotent set to OFF or method not idempotent.

To resolve the issue:

1. Restart the OHS pod.
2. Get the OHS pod name using `kubectl -n <namespace> get pods` command where the name of the pod is `<project>-<instance>-oam-ohs-<podsuffix>`.

   > **Note:**
   >
   > The pod name starts with `Pod name starts with <project>-<instance>-oam-ohs-<number>`.

3. Open the OHS pod using: `kubectl -n oamns exec -it <OHS_POD_NAME> -- bash`.
4. Run the command:

   ```
   echo '<DOMAIN_USER_PWD>' | /u01/oracle/ohssa/user_projects/domains/
   <project>-<instance>-oam-ohs/bin/restartComponent.sh ohs1
   ```

5. Exit from the pod using `exit`.

Alternatively, you can restart OHS by rolling out restart from deployments as follows:

```
    kubectl -n <namespace> get deployments
        kubectl -n <namespace> rollout restart deployment <project>-
<instance>-oam-ohs
```

UIM UI Not Accessible on Using SSL Port of Traditional UIM Instance

Check the OHS logs and if you observe SSL Handshake error message in logs. For example, `wl_ssl_open : SSL Handshake failed onserror : Success, error : 29024, status : 2`, perform the following resolution steps:

1. Identify the Inventory certificate file (**.pem**) and copy it into OHS pod.

   ```
   kubectl -n <NAMESPACE> cp <inventory-certificate-file> <NAMESPACE>/
   <OHS_POD_NAME>:/u01/oracle/ohssa/user_projects/domains/<project>-
   <instance>-oam-ohs/config/fmwconfig/components/OHS/instances/ohs1/
   keystores/
   ```

2. Enter the OHS pod using:

   ```
   kubectl -n <NAMESPACE> exec -it <OHS_POD_NAME> bash
   ```

3. Run the below commands inside the OHS pod:

   ```
   cd /u01/oracle/ohssa/user_projects/domains/<project>-<instance>-oam-ohs/
   config/fmwconfig/components/OHS/instances/ohs1/keystores

   /u01/oracle/ohssa/oracle_common/bin/orapki wallet create -wallet <wallet-
   name> -auto_login_only

   /u01/oracle/ohssa/oracle_common/bin/orapki wallet add -wallet <wallet-
   name> -trusted_cert -cert <inventory-certificate-file> -auto_login_only

   cd ..

   vim mod_wl_ohs.conf

   #edit the file for the locations mentioned as below

     <Location /Inventory>
       WLSRequest On
       WebLogicHost <inventory.host>
       WeblogicPort <inventory.port>
       #WLProxySSL ON
       WLProxySSLPassThrough ON
       SecureProxy ON
       WLSSLWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/$
   {COMPONENT_TYPE}/instances/${COMPONENT_NAME}/keystores/<wallet-name>"
       SetHandler weblogic-handler
     </Location>
     <Location /InventoryWS>
       WLSRequest On
       WebLogicHost <inventory.host>
       WeblogicPort <inventory.port>
       #WLProxySSL ON
       WLProxySSLPassThrough ON
       SecureProxy ON
       WLSSLWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/$
   {COMPONENT_TYPE}/instances/${COMPONENT_NAME}/keystores/<wallet-name>"
   ```

```
    SetHandler weblogic-handler
  </Location>
  <Location /InventoryRSOpenAPI>
    WLSRequest On
    WebLogicHost <inventory.host>
    WeblogicPort <inventory.port>
    #WLProxySSL ON
    WLProxySSLPassThrough ON
    SecureProxy ON
    WLSSLWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/$
{COMPONENT_TYPE}/instances/${COMPONENT_NAME}/keystores/<wallet-name>"
    SetHandler weblogic-handler
  </Location>
  <Location /cartridge>
    WLSRequest On
    WebLogicHost <inventory.host>
    WeblogicPort <inventory.port>
    #WLProxySSL ON
    WLProxySSLPassThrough ON
    SecureProxy ON
    WLSSLWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/$
{COMPONENT_TYPE}/instances/${COMPONENT_NAME}/keystores/<wallet-name>"
    SetHandler weblogic-handler
  </Location>
  <Location /mapviewer>
    WLSRequest On
    WebLogicHost <inventory.host>
    WeblogicPort <inventory.port>
    #WLProxySSL ON
    WLProxySSLPassThrough ON
    SecureProxy ON
    WLSSLWallet "${ORACLE_INSTANCE}/config/fmwconfig/components/$
{COMPONENT_TYPE}/instances/${COMPONENT_NAME}/keystores/<wallet-name>"
    SetHandler weblogic-handler
  </Location>
```

4. Restart the OHS server:

```
echo '<WL_ADMIN_PWD>' | /u01/oracle/ohssa/user_projects/domains/<project>-
<instance>-oam-ohs/bin/restartComponent.sh ohs1
```

5. See the **Configuring a Plug-In for One-Way SSL** section from *Using Oracle WebLogic Server Proxy Plug-Ins 12.2.1.4.0* and perform the following:

> **Note:**
>
> Perform this in UIM administrator console.

   a. Log into the Oracle WebLogic Server administration console.
   b. In the **Domain Structure** pane, expand the **Environment** node. If the server instances that are used to proxy the requests from Oracle HTTP Server are in a cluster, select **Clusters**. Otherwise, select **Servers**.

c. Select the server or cluster that you want to proxy the requests from Oracle HTTP Server.

d. In the **Configuration: General** tab, scroll down to the **Advanced** section and expand it.

e. Do one of the following:

- To enable one-way SSL, select **WebLogic Plug-In Enabled**.
- To enable two-way SSL where client certificates are used to authenticate, select **Client Cert Proxy Enabled**.
- To enable two-way SSL with client certificates, select **Both**.

f. If you have selected **Servers** in Step 2, repeat steps 3 and 4 for the other servers to which you want to proxy the requests from Oracle HTTP Servers.

g. Click **Save**.

h. For the change to take effect, you must restart the server instances.

See the **Configuring the SSL Policy/Certificate** section from *UIM System Administrator's Guide* for configuring SSL with Oracle WebLogic server.

Self-signed SSL Certificates

This section provides information on generating your self-signed SSL certificates.

Generating Self-signed Certificates

To generate self-signed certificates:

1. Create the **certs** folder under the **$COMMON_CNTK** directory.

```
mkdir $COMMON_CNTK/certs
```

2. Update the following command with the appropriate values of instance, project, and hostSuffix names and run it to generate a common self-signed certificate and key that can be used for OAM, message-bus, UIM, and UTIA. You can add or remove the DNS entries from the below command based on your requirements.

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout $COMMON_CNTK/
certs/commonkey.pem -out $COMMON_CNTK/certs/commoncert.pem -subj "/
CN=<INSTANCE>.<PROJECT>.admin.<hostSuffix>" -extensions san -config <(echo
'[req]'; echo 'distinguished_name=req'; echo '[san]';echo
'subjectAltName=@alt_names'; \echo '[alt_names]'; \
echo 'DNS.1=<INSTANCE>.<PROJECT>.admin.<hostSuffix>'; \
echo 'DNS.2=<INSTANCE>.<PROJECT>.oam.<hostSuffix>'; \
echo 'DNS.3=<INSTANCE>.<PROJECT>.ohs.<hostSuffix>'; \
echo 'DNS.4=<hostSuffix>'; \
echo 'DNS.5=<INSTANCE>.<PROJECT>.topology.<hostSuffix>'; \
echo 'DNS.6=localhost'; \
echo 'DNS.7=svc.cluster.local'; \
echo 'DNS.8=<INSTANCE>.<PROJECT>.<hostSuffix>'; \
echo 'DNS.9=admin.<INSTANCE>.<PROJECT>.<hostSuffix>'; \
echo 'DNS.10=t3.<INSTANCE>.<PROJECT>.<hostSuffix>'; \
echo 'DNS.11=<INSTANCE>.<PROJECT>.messaging.broker0.<hostSuffix>'; \
echo 'DNS.12=<INSTANCE>.<PROJECT>.messaging.broker<N>.<hostSuffix>'; \
```

```
echo 'DNS.13=<INSTANCE>.<PROJECT>.messaging.bootstrap.<hostSuffix>'; \
)
```

3. You can add or remove the DNS entries in the above sample certificate. Check the following scenarios for removing or adding the DNS entries:
   - If the Message Bus ingress listener is not enabled, you can remove the following DNS entries:

     ```
     quick.sr.messaging.broker0.uim.org
     quick.sr.messaging.broker1.uim.org
     quick.sr.messaging.bootstrap.uim.org
     ```

   - For traditional UIM, you can remove the following DNS entries and add the hostnames of traditional UIM servers:

     ```
     quick.sr.uim.org
     admin.quick.sr.uim.org
     t3.quick.sr.uim.org
     ```

   - If OAM is not used as IdP, you can remove following hostnames from the certificate:

     ```
     quick.sr.admin.uim.org
     quick.sr.oam.uim.org
     quick.sr.ohs.uim.org
     ```

Generating Wild Card SSL Certificate

To generate wild card SSL certificate:

1. Create the **certs** folder in **$COMMON_CNTK directory** as follows:

   ```
   mkdir $COMMON_CNTK/certs
   ```

2. To generate a wild card SSL certificate you can update <hostSuffix> value. The default is **uim.org** and run following command:

   ```
   openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout $COMMON_CNTK/
   certs/wildcardkey.pem -out $COMMON_CNTK/certs/wildcardcert.pem -subj "/
   CN=*.<hostSuffix>" -extensions san -config <(echo '[req]'; echo
   'distinguished_name=req';
   echo '[san]';echo 'subjectAltName=@alt_names'; \echo '[alt_names]'; \
   echo 'DNS.1=*.<hostSuffix>'; \
   )
   ```

Note:

- To use wild card certificates, you must configure **subDomainNameSeperator** field as **-**, in **applications.yaml** and **project.yaml** in the **spec** path location.
- WebLogic by default does not recognizes wild card certificates. In production environment, you must configure the custom hostname verifier as `weblogic.security.utils.SSLWLSWildcardHostnameVerifier`. See WebLogic documentation for setting up **hostNameVerifier**.
- In development environment, you can disable hostname verification.

4

Deploying Unified Operations Message Bus

This chapter describes how to deploy Unified Operations Message Bus.

Unified Operations Message Bus Overview

The Oracle Communications Unified Operations Message Bus (OCUOMB) service is a distributed event store and stream-processing platform service. The Message Bus clients send and receive events and messages from the Message Bus service that in turn sends and receives from a specific channel called Topic. This enables that the source and target clients or services are loosely coupled and asynchronous. Message Bus uses Apache Kafka in its platform to support the event store and stream-processing and for packaging. For deployment, Message Bus uses Strimzi.

Strimzi simplifies the process of running Apache Kafka in a Kubernetes cluster. Strimzi provides container images and operators for running Apache Kafka on Kubernetes. Strimzi operators are fundamental for the smooth running of Strimzi. These operators are software extensions to Kubernetes that make use of custom resources to manage applications and their components. These operators simplify the process of:

- Deploying, running, and upgrading the Kafka cluster and its components.
- Configuring and securing access to Kafka.
- Creating and managing Kafka topics.

Operators are a method of packaging, deploying, and managing a Kubernetes application. The Strimzi operators extend Kubernetes functionality and automate common and complex tasks related to a Kafka deployment. By implementing knowledge of Kafka operations in code, Kafka administration tasks are simplified and require less manual intervention. See https://strimzi.io/docs/operators/latest/overview.html for more details on the Strimzi operators. Strimzi has the following operators:

- *Cluster Operator*: Deploys and manages the Apache Kafka clusters, Kafka Connect, Kafka Mirror Maker, Kafka Bridge, Kafka Exporter, Cruise Control, and the Entity Operator.
- *Entity Operator*: Comprises the Topic Operator and User Operator
- *Topic Operator*: Manages Kafka topics

See the following webssites for more information on Strimzi and Apache Kafka:

- Strimzi: https://strimzi.io/
- Apache Kafka: https://kafka.apache.org/

The Message Bus service provides scripts and helm charts to deploy and manage the Apache Kafka cluster in Kubernetes by using the Strimzi operator and Kubernetes Custom Resources definitions. The Message Bus service does not provide any image builder toolkits to build the container images and by default, Helm charts pull the required container images from the quay.io/strimzi container repository.

Table 4-1 Container Images and Purposes

Container Image	Purpose
quay.io/strimzi/operator:<*Strimzi_Operator_version*>	Container Image with Strimzi Operator.
quay.io/strimzi/kafka:<*Strimzi_Operator_version*>-kafka-<*Kafka_version*>	Container Image with Apache Kafka and Strimzi distribution. In the following sections, the reference to the container image is named as STRIMZI_KAFKA_IMAGE_NAME

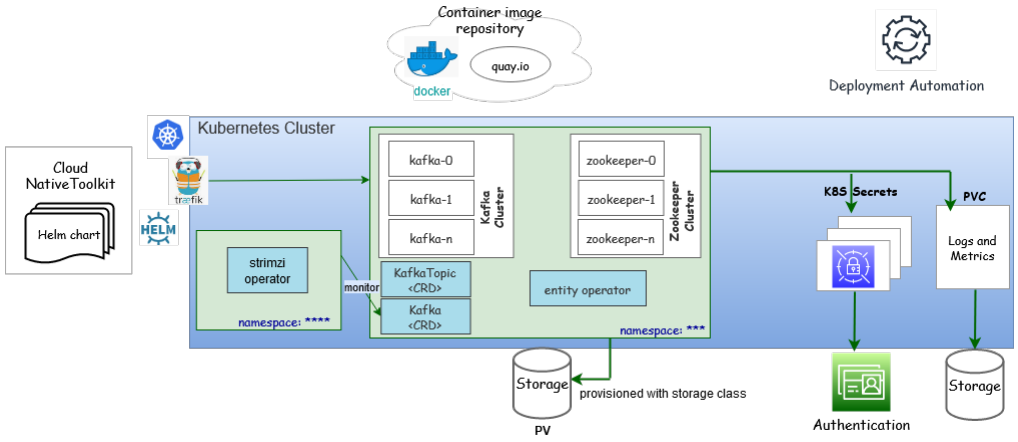
Note:

See *UIM Compatibility Matrix* for the latest versions of software.

Message Bus Cloud Native Architecture

The Message Bus service uses Apache Kafka as a distributed event store platform. To run an Apache Kafka cluster on Kubernetes, the Message Bus service uses the Strimzi operator. Strimzi is an open-source project that provides container images and operators for running Apache Kafka on Kubernetes.

Figure 4-1 Message Bus Cloud Native Architecture



Access to Message Bus

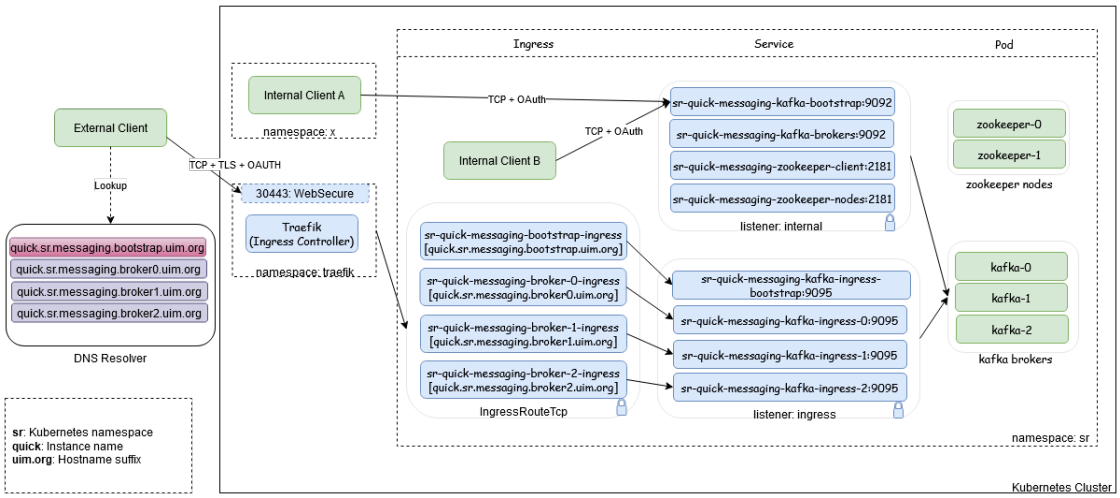
While deploying the Message Bus Service in Kubernetes namespace, the following Kubernetes service objects are created to access the Message Bus pods either internally or externally (through an ingress controller). The external access is provided through the ingress controller by IngressRouteTCP objects.

You can override the value of **subDomainNameSeparator**. The default separator is **"."**, This value can be modified as **"-"** to match the wild-card pattern of SSL certificates.

To override, uncomment and change the value in **applications.yaml** as follows:

```
#subDomainNameSeparator: "."
#Example hostnames for "-" : quick-sr-messaging-bootstrap.uim.org
```

Figure 4-2 Process of Accessing the Message Bus



The external access to Message Bus service is supported with **TCP+TLS+OAuth 2.0 Authentication** through Traefik ingress controller or a Generic ingress controller. The internal access to Message Bus Service is also supported with **TCP+TLS+OAuth 2.0 Authentication** where TLS can be configurable. Access to Message Bus service is configured through the listeners section in **applications.yaml** file.

> **Note:**
>
> - If the client is in the same Kubernetes cluster, the **internal** listener is used.
> - If the client is outside the Kubernetes cluster, the **ingress** listener is used.

The Message Bus is deployed using the scripts provided in Common CNTK. For deployment prerequisites, see "Planning and Validating Your Cloud Environment".

The following steps need to be followed to deploy a Kafka cluster in a Kubernetes namespace in a cluster:

1. Deploy the Strimzi operator to manage your Kafka cluster.

> **Note:**
>
> This is an administrative one-time activity where additional cluster roles are required.

   a. Create a namespace to deploy Strimzi Operator.

b. Deploy Strimzi Operator in the namespace. See "Deploying Strimzi Operator" for more information.

2. Deploy the Message Bus that has Kafka cluster, ZooKeeper cluster, and entity operator.
   a. Create a namespace to deploy the Kafka cluster.
   b. Register the namespace with Strimzi Operator. See "Register namespaces with Strimzi Operator" for more information.
   c. Register the namespace with Traefik. See "Registering the Namespaces with Strimzi Operator" for more information.

> **Note:**
>
> - The ingress controller (Traefik or Generic) has to be available.
> - Register the namespace with Traefik ingress controller. If you use Generic Ingress controller, ensure that **ingress.className** is set in the **applications.yaml** file.

   d. Deploy Kafka Cluster in the namespace. See "Deploy Kafka Cluster and Kafka Topic" for more information.
3. Validate the deployment with sample standalone producer and consumer clients. See the "Validating the Kafka cluster" and "Internal access - same namespace - plain" for more information.

Strimzi Operator

Export the Strimzi operator namespace environment variable to run the deployment script using the COMMON_CNTK:

```
export STRIMZI_NS=<STRIMZI_OPERATOR_NAMESPACE>
```

The configurable parameters of the Strimzi Operator charts and their default values are listed in the corresponding subsections within this document.

See the **Assembling the Specifications** section in **strimzi-operator-override-values.yaml**. To override the default values, copy the **$COMMON_CNTK/samples/strimzi-operator-override-values.yaml** file to the directory **$SPEC_PATH/<STRIMZI_PROJECT>**, where **<STRIMZI_PROJECT>** is the Kubernetes namespace where the Strimzi operator is being deployed.

Create Global Resources

Configure the **createGlobalResources** value in **strimzi-operator-override-values.yaml file** (sample). If you require more than one strimzi-cluster-operator in the same cluster, set the value to **false**. Only the latest versions of strimzi should be installed, in case of multiple strimzi-operator to avoid any risk related to backward compatibility.

Private Container Repository

The Strimzi operator pulls the Strimzi component container images from **quay.io** registry. If you want to maintain private container registry, `pull` the images from the **quay.io** registry and

`push` them into the private container registry. It is mandatory to push the images with same name and tag, the repository path can be different. For Strimzi image and tag names, see "Unified Operations Message Bus Overview" for more information.

See **About Container Image Management** section from *UIM Cloud Native Deployment Guide* for more information on private container repository management.

To use the private container registry, uncomment and modify the values in **$SPEC_PATH/<STRIMZI_PROJECT>/strimzi-operator-override-values.yaml** file. Provide the modified **strimzi-operator-override-values.yaml** file path as an `-f` option to the Strimzi operator create/upgrade command.

If the private container registry requires authentication, create the Kubernetes secret in the namespace and provide the secret name as part of **strimzi-operator-override-values.yaml** file. Create the secret with same name in the namespace where the Kafka cluster is planned to deploy.

strimzi-operator-override-values.yaml file (Sample)

```
defaultImageRegistry: <Image registry>
defaultImageRepository: <Image Repository>
image:
  imagePullSecrets: <Pull Secret>
```

The following is a sample command to create Kubernetes secret for the registry. Create the secret in the namespace where the Strimzi operator is being deployed. See https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/ for creating secret.

```
kubectl create secret docker-registry <secret-name> --docker-server=<Image
Registry> \
                                                    --docker-
username=<Username> \
                                                    --docker-
password=<Password> \
                                                    -n
<STRIMZI_OPERATOR_NAMESPACE>
```

ImagePullPolicy

The following sample of **ImagePullPolicy** for Strimzi Operator is provided. To create the policy using a different procedure, see https://kubernetes.io/docs/concepts/containers/images/#image-pull-policy

strimzi-operator-override-values.yaml file (Sample)

```
image:
  imagePullPolicy: IfNotPresent
```

Resources

These resources are used for configuring the virtual resources (limits and requests). Uncomment or add the blow resources section with new values in the **strimzi-operator-override-values.yaml** file.

```
resources:
  requests:
    memory: <Mi>
    cpu: <m>
  limits:
    memory: <Gi>
    cpu: <"1">

fullReconciliationIntervalMs: 120000
operationTimeoutMs: 300000
```

The default values are as follows:

```
resources.limits.memory: 500Mi
resources.limits.cpu: 500m
resources.requests.memory: 1Gi
resources.requests.cpu: 1
```

Along with the above resources, you can provide the following additional configurations:

```
# Full reconciliation interval in milliseconds
fullReconciliationIntervalMs: 120000
# Operation timeout in milliseconds
operationTimeoutMs: 300000
```

Deploying Strimzi Operator

Run the following script to deploy the Strimzi operator in the Kubernetes namespace:

```
$COMMON_CNTK/scripts/strimzi-operator.sh -p <STRIMZI_OPERATOR_NAMESPACE> -c
create
```

Optionally, run the following script to deploy the Strimzi operator in Kubernetes namespace with custom image registry and repository:

```
$COMMON_CNTK/scripts/strimzi-operator.sh -p <STRIMZI_OPERATOR_NAMESPACE> -c
create -f $SPEC_PATH/<STRIMZI_OPERATOR_NAMESPACE>/strimzi-operator-override-
values.yaml
```

Upgrading Strimzi Operator

Run the following script to upgrade the Strimzi Operator in Kubernetes namespace:

```
$COMMON_CNTK/scripts/strimzi-operator.sh -p <STRIMZI_OPERATOR_NAMESPACE> -c
upgrade
```

Optionally, run the following script to deploy the Strimzi operator in Kubernetes namespace with custom image registry and repository:

```
$COMMON_CNTK/scripts/strimzi-operator.sh -p <STRIMZI_OPERATOR_NAMESPACE> -c
upgrade -f $SPEC_PATH/<STRIMZI_OPERATOR_NAMESPACE>/strimzi-operator-override-
values.yaml
```

> **Note:**
>
> If you are upgrading strimzi-cluster-operator to a newer version, the old toolkit should be used for old version of strimzi (the one already deployed) and the new toolkit should be used while upgrading to the newer version, in case of **Create**, **Upgrade**, **Delete**, **Register**, and **Unregister**.

Uninstalling Strimzi Operator

Run the following script to uninstall the Strimzi Operator from Kubernetes namespace:

```
$COMMON_CNTK/scripts/strimzi-operator.sh -p <STRIMZI_OPERATOR_NAMESPACE> -c
delete
```

Validating Strimzi Operator

Validate the Strimzi operator that is installed in the provided namespace by running the following command:

```
$kubectl get pod -n <STRIMZI_OPERATOR_NAMESPACE>

NAME                                        READY   STATUS    RESTARTS   AGE
strimzi-cluster-operator-*******-***        1/1     Running   0          6m55s
```

Validate the Helm release installed for the Strimzi operator in the provided namespace by running the following command:

```
$helm list -n <STRIMZI_OPERATOR_NAMESPACE>

NAME                    NAMESPACE                           REVISION
STATUS          CHART                           APP VERSION
strimzi-operator      <STRIMZI_OPERATOR_NAMESPACE>          1
deployed        strimzi-kafka-operator-x.y.z  x.y.z
```

Restarting the Strimzi Operator

Run the following script to restart the Strimzi Operator:

```
$COMMON_CNTK/scripts/strimzi-operator.sh -p <STRIMZI_OPERATOR_NAMESPACE> -c
restart
```

Registering the Namespaces with Strimzi Operator

To create and manage the Kafka cluster in a Kubernetes namespace, this namespace must be registered with the Strimzi operator to monitor the CRDs.

Run the following script to register the namespace(s) with the Strimzi operator to monitor and create or manage the Kafka cluster and its components:

```
$COMMON_CNTK/scripts/register-namespace.sh -p <Namespace to be monitored> -t
strimzi
```

Unregistering the Namespaces with Strimzi Operator

Run the following script to unregister the namespaces from the Strimzi operator:

```
$COMMON_CNTK/scripts/unregister-namespace.sh -p <Namespace to be un-
monitored> -t strimzi
```

Deploying and Managing Message Bus

Kafka cluster consists of Kafka Brokers and Zookeeper nodes. Once the Strimzi operator is successfully installed in the Kubernetes cluster and a namespace for the Kafka cluster is registered to monitor, you can deploy and manage the Kafka cluster.

Update the **applications.yaml** file as per your requirement and verify the following configuration elements in the yaml file before deploying the Kafka cluster:

> **Note:**
>
> If **applications.yaml** is not copied from Common CNTK, copy the **$COMMON_CNTK/samples/applications.yaml** file to your local directory, for example: $SPEC_PATH/sr/quick, where the **sr** is the Kubernetes namespace and **quick** is the instance name.

- The Storage class name that is used to create persistent volumes.
- The Kafka cluster replicas, which is the number of Kafka Brokers and Zookeeper nodes.
- Virtual Resource sizing.
- The Kafka Broker default settings.
- The listeners to be exposed with authentication and TLS.

- Authentication details.
- Metrics enablement.
- Affinity settings
- Update partitions, replicas, and retention period values for the default Kafka Topics.

See "Configuring the applications.yaml File" for more details.

Deploying Message Bus

Run the following commands to deploy the Kafka cluster with Kafka Topics in a Kubernetes namespace:

```
$COMMON_CNTK/scripts/create-applications.sh \
-p <kafka cluster namespace> \
-i <kafka cluster instance name> \
-f <path to override values yaml file> \
-a messaging-bus
```

For example:

In the following command, **sr** is a namespace and **quick** an instance name:

```
$COMMON_CNTK/scripts/create-applications.sh -p sr -i quick -f $SPEC_PATH/sr/
quick/applications.yaml -a messaging-bus
```

Upgrading Message Bus

The Kafka cluster upgrade requires persistent storage enabled for rolling update. Oracle recommends you have multiple replicas so that the service is not down while upgrading.

Update the Kafka cluster configuration in the **applications.yaml** file:

```
$COMMON_CNTK/scripts/upgrade-applications.sh \
-p <kafka cluster namespace> \
-i <kafka cluster instance name> \
-f <path to override values yaml file> \
-a messaging-bus
```

For example, run the following command to upgrade the Kafka cluster and Kafka topic running in sr namespace with instance as quick:

```
$COMMON_CNTK/scripts/upgrade-applications.sh -p sr -i quick -f $SPEC_PATH/sr/
quick/applications.yaml -a messaging-bus
```

Deleting Message Bus

Run the following script to delete or uninstall the Kafka cluster and Kafka Topic from the Kubernetes namespace:

```
$COMMON_CNTK/scripts/delete-applications.sh \
-p <kafka cluster namespace> \
-i <kafka cluster instance name> \
```

```
-f <path to override values yaml file> \
-a messaging-bus
```

For example: Run the following command to delete the Kafka cluster with Kafka topic running in sr namespace with instance as quick:

```
$COMMON_CNTK/scripts/delete-applications.sh -p sr -i quick -f $SPEC_PATH/sr/
quick/applications.yaml -a messaging-bus
```

Validating Message Bus

Check the pods created for the Kafka cluster. The following sample output shows the internal listener configuration. If it has any external listener settings, the additional service objects appear:

```
$kubectl get svc -n sr

NAME                                    TYPE        CLUSTER-IP      EXTERNAL-
IP   PORT(S)                       AGE
sr-quick-messaging-kafka-bootstrap    ClusterIP   <clusterIP>
<none>        9091/TCP,9092/TCP            22m
sr-quick-messaging-kafka-brokers      ClusterIP   None
<none>        9090/TCP,9091/TCP,9092/TCP   22m
sr-quick-messaging-zookeeper-client   ClusterIP   <clusterIP>
<none>        2181/TCP                     23m
sr-quick-messaging-zookeeper-nodes    ClusterIP   None
<none>        2181/TCP,2888/TCP,3888/TCP   23m
```

Check the Service object created for the Kafka cluster. The following sample output shows the Kafka and ZooKeeper replica as **1**.

```
$kubectl get pod -n sr

 NAME                                                READY   STATUS
RESTARTS   AGE
 sr-quick-messaging-entity-operator-*****-****       3/3     Running
0          27h
 sr-quick-messaging-kafka-0                          1/1     Running
0          27h
 sr-quick-messaging-zookeeper-0                      1/1     Running
0          27h
```

Check the Helm release:

```
$helm list -n sr

NAME                    NAMESPACE       REVISION        UPDATED
STATUS          CHART                   APP VERSION
sr-quick-messaging      sr              1               *****
deployed        kafka-cluster-<x.y.z>     <x.y.z>
```

Check the persistent volume claims created:

```
$kubectl get pvc -n sr`

NAME                                     STATUS
VOLUME                                       CAPACITY   ACCESS MODES
STORAGECLASS   AGE
data-sr-quick-messaging-kafka-0       Bound    <volume>  1Gi
RWO            sc              27h
data-sr-quick-messaging-zookeeper-0   Bound    <volume>  1Gi
RWO            sc              27h
```

Run a standalone producer or consumer. See "Internal access - same namespace – plain" to run standalone producer and consumer pods in a Kafka cluster namespace.

> **Note:**
>
> As part of deploying, upgrading, and deleting the Message Bus, the Kafka topics are also created, upgraded, and deleted from the configuration provided in the input yaml file.

Restarting Message Bus

The **restart-application.sh** script with application name as **messaging-bus** restarts all the subcomponents such as Kafka, ZooKeeper, and Entity Operators of the Message Bus. Run the following command to restart:

```
$COMMON_CNTK/scripts/restart-applications.sh -p sr -i quick -f $SPEC_PATH/sr/
quick/applications.yaml -a messaging-bus
```

> **Note:**
>
> The Message Bus service restart requires to have multiple replicas so that the service is not down while upgrading and the replica count should be greater than or equal to **2**.

To validate the restart option, see "Validating Message Bus ".

Configuring the applications.yaml File

Modify the values in the **applications.yaml** file and upgrade or create the Message Bus service. The following configurations are available for the Message Bus service:

- Image Pull Secrets
- Security Context
- Cluster Size
- Storage

- Broker Defaults
- JVM Options
- Kafka Topics
- Accessing Kafka Cluster
- Authentication

Using Image Pull Secrets

You use the Image Pull Secrets sample only while using the private container repository that requires authentication. These authentication details have to be provided as Kubernetes secret object in the namespace where the Kafka cluster is planned to be deployed. This process is also followed while deploying Strimzi Operator.

> **Note:**
>
> Provide the secret name in the **kafka-cluster** section, if using different secret name than in the Strimzi Operator's namespace.

Image Pull Secrets (Sample)

```
imagePullSecret:
  imagePullSecrets:
    - name: <secret name>
```

The sample command to create secret object for registry authentication is as follows:

```
kubectl create secret docker-registry <secret-name> --docker-server=<Image
Registry> \
                                                    --docker-
username=<Username> \
                                                    --docker-
password=<Password> \
                                                     -n <Kafka-Namespace>
```

See https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/ to create the secret object.

Security Context

The **userSecurity** section that has **securityContext** is applicable only when you want to define privilege and access control settings for a pod or container. The pod security context which is configured at the pod-level is provided as a sample and is applied to all containers in given pod.

> **Note:**
>
> If a value is commented, it cannot be used, To use a different key-value, uncomment the corresponding value in **applications.yaml**.

See https://strimzi.io/blog/2022/09/09/configuring-security-context-in-pods-managed-by-strimzi/ and https://kubernetes.io/docs/tasks/configure-pod-container/security-context/ for more information.

Security-Context (Sample)

```
userSecurity:
  securityContext:
    runAsNonRoot: <true/false>
    runAsUser: <userID>
    runAsGroup: <groupID>
    fsGroup: <fsGroup>
```

Cluster Size

The Message Bus cluster consists of Kafka Brokers and Zookeeper nodes. Modify the replicas count for the Kafka Brokers and Zookeeper nodes according to the usage. For high availability of Message Bus service, make sure the number of replicas is minimum **3** for Kafka and Zookeeper, in production instance and adjust Kafka Broker configuration accordingly:

```
kafka-cluster:
  replicas:
    kafka: 3
    zookeeper: 3
```

Storage

The Message Bus uses Strimzi to deploy the Apache Kafka cluster in Kubernetes cluster. For Strimzi to work as required, an efficient data storage infrastructure is essential. Oracle recommends using a block storage as Strimzi is tested for using with block storage. For more information on data storage, see https://strimzi.io/docs/operators/latest/deploying#considerations-for-data-storage-str

The Message Bus Service stores the events (or messages) in block storage using the Kubernetes Persistent Volumes. Modify the values for class, size, and **isDeleteClaim** values in storage section under the Kafka cluster. The storage class must have dynamic persistent volume provision capability:

```
kafka-cluster:
  #storage:
    #When storage.type below is set as "persistent-claim", the storage class
name & size are mandatory to be set
    #type: persistent-claim
    #class: psrnfsn1
    #size: 1Gi
    #isDeleteClaim: false
```

For development to use ephemeral (that is, temporary container storage), do not change the values. These values must be commented for ephemeral.

Broker Defaults

The following configuration is applied when the Topics are auto created. Modify the following settings in the **kafkaConfig** section under the Kafka cluster accordingly:

```
kafka-cluster:
  kafkaConfig:
    #The default replication factor for automatically created topics
    defaultReplicationFactor: 2
    offsetsTopicReplicationFactor: 2
    transactionStateLogReplicationFactor: 2
    transactionStateLogMinIsr: 2
    minInsyncReplicas: 2
    logRetentionMinutes: 30
    numPartitions: 3
```

The values for replicationFactors and minimum in-sync replicas must be entered according to the values entered in the Kafka Cluster. These values must be less than or equal to the Kafka Cluster replica values.

For more information on the values, see the Kafka documentation at: https://kafka.apache.org/081/documentation.html#brokerconfigs

JVM Options

The Message Bus cluster consists of Kafka Brokers and Zookeeper nodes. Modify the **jvmOptions** for Kafka Brokers and Zookeeper nodes according to the usage. See https://strimzi.io/docs/operators/latest/full/configuring.html#con-common-configuration-jvm-reference for more details.

```
jvmOptions:
  kafka:
    -Xms: 1024m
    -Xmx: 1024m
    # javaSystemProperties:
    #   - name: <placeHolder>
    #     value: <value>

  zookeeper:
    -Xms: 1024m
    -Xmx: 1024m
    # javaSystemProperties:
    #   - name: <placeHolder>
    #     value: <value>
```

Kafka Topics

Add or update the Kafka Topics in the **applications.yaml** file in the **kafkaTopics** section which are required for the Message Bus service clients (producers or receivers).

For example:

```
kafka-topic:
  #List of Kafka topics
  kafkaTopics:
    - name: <topic1>
      partitions: <no_partitions>
      replicas: <no_replicas>
      config:
        retention: 7200000
        segmentBytes: 1073741824
```

The following topics are required for the UTIA integration which are defined in the **applications.yaml** file within the Common CNTK samples. These topics are created during the deployment of Message Bus service using Common CNTK:

Table 4-2 Topic, producer, and consumer details.

Topic	Producer	Consumer	Additional Details
ora-uim-topology	UIM	Unified Topology	See *UIM System Administrator's Guide* for more details.
ora-fault-topology	Assurance System	Unified Topology	See *Unified Topology* for more details
ora-retry-topology	Unified Topology	Unified Topology	See *Unified Topology* for more details
ora-dlt-topology	Unified Topology	Unified Topology	See *Unified Topology* for more details

> **Note:**
>
> Do not use the default topics (ora-uim-topology, ora-fault-topology, ora-retry-topology and ora-dlt-topology) for a standalone testing. Use only the **ora-test-topic** to test the deployment of Message Bus service.

Accessing Kafka Cluster

There are various listener type configurations available to access the Message Bus service internally and externally. The Authentication configuration is applied across all listener types. As part of Kafka cluster deployment, the Kubernetes service objects are created to provide access to Kafka cluster pods. This service objects are created based on the listener type configuration in the **applications.yaml** file for message-bus section. You can access the Message Bus service in any of the following ways:

- Accessing within the same cluster (Internal access)
- Accessing from outside of the cluster (External access)

> **Note:**
>
> When a Message Bus service is deployed, it autogenerates the certificates of TLS for server and client. You must use the custom certificates so that the certificates are retained when the service is terminated and created again. See "Using Wild Card Certificates" for more information.

Accessing the Message Bus service from within the same cluster (Internal access)

The **internal** listener configuration in the **applications.yaml** file is used when the client services are in the same Kubernetes cluster, which can be in the same namespace or a different namespace. This configuration is enabled by default.

```
kafka-cluster:
  listeners:
    #Plain is for internal access within the same k8s cluster.
    internal:
      # Enable the tls to true if encryption/decryption is needed for
internal access
      #tls: false
```

See "Configuring Message Bus Listeners" for more information.

Accessing the Message Bus service from outside of the cluster (External access)

The **ingress** listener configuration in the **applications.yaml** file is used when the client services are outside of the Kubernetes cluster. This access is achieved using the ingress controller.

```
# To expose the kafka-cluster to external kafka clients via ingress
controller uncomment the following and modify accordingly.

# Valid values are TRAEFIK, GENERIC
ingressController: <INGRESS_CONTROLLER>

#ingress:
#  #specify className field for ingressClassName of generic ingress
controller.
#  #In case of nginx the default values is nginx
#  className: "nginx"

#provide loadbalancer port
# if TLS is enabled in global section, then loadbalancerport will be used as
external port for Generic or Traefik.
loadbalancerport: <loadBalancer-port>


kafka-cluster:
  listeners:
    #To expose the kafka-cluster to external kafka clients via ingress
controller (traefik or generic) uncomment the following and modify
accordingly.
    #ingress:
```

```
    #  #The secure port of either ingress controller or external load-
balancer. If TLS is Disabled in global, then below ingressSslPort will be
used as external port.
    #  ingressSslPort: <LOADBALANCER_PORT>
    #  #If using Generic Ingress controller, below given annotations are
mandatory for Message-Bus external access.
    #  #These annotations are required for nginx ingress controller.
    #  annotations:
    #    nginx.ingress.kubernetes.io/ingress.allow-http: "false"
    #    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
    #    ingress.kubernetes.io/ssl-passthrough: "true"
    #    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
```

See "Configuring Message Bus Listeners" for more information.

Accessing the Message Bus service using a nodeport listener

The **nodeport** listener configuration in **applications.yaml** file configuration is also used when the client services are outside of the Kubernetes cluster. The access is directly with the Kubernetes work node's port.

> **Note:**
>
> Oracle does not recommend this listener for production. It must be used only for debugging where ingress controller is not deployed.

```
kafka-cluster:
  listeners:
    #To expose the kafka-cluster to external kafka clients without ingress
controller, uncomment the following section and modify accordingly
    #nodeport:
      #default is true. can be turned off if needed
      #tls: true
      #if need to expose on a static nodeport, pease uncomment the below
section and provide values
      #nodePort: 32100
```

See "Configuring Message Bus Listeners" for more information.

Configuring Authentication

Kafka 2.0.0 or later supports an extensible OAuth 2.0 compatible token-based mechanism available, called **SASL OAUTHBEARER**. Strimzi has developed extensions that provide integration with OAuth 2.0 compliant authorization servers. That means, in principle, you can use any OAuth 2.0 compliant authorization server to enable centrally managed users for authentication with Kafka.

The Message Bus service uses a Strimzi operator to deploy Kafka brokers and in-turn use OAuth 2.0 token-based authentication while establishing a session to a Kafka broker. With this authentication, Message Bus clients (or Kafka clients) and Kafka brokers communicate with a central OAuth 2.0 compliant authorization server. These Kafka clients use the authorization server to obtain access tokens and are configured with access tokens issued by the server. Kafka brokers communicate with authorization server to validate the tokens presented by the

clients, thus confirming their identities. You can perform the validation of access token using a fast local JWT validation or a token validation using an **introspection endpoint**.

To configure OAuth 2.0 support for Kafka Brokers in the Message Bus service, you need to update **applications.yaml** file and create or upgrade the service.

Prerequisites

- The Authorization server **(OAuth 2.0 compliant)** is up and running. See "Deploying OAM along with OHS for Authentication Service" in Authentication Service
- Configure the client for Kafka broker in the authorization server. See "Creating a Client" section in Authentication Service
- Configure the clients for Kafka producer or consumer application in the authorization server. See "Creating a Client" section in Authentication Service
- Kafka cluster is configured with **oauth** type Authentication. See the following sections.

Enable Authentication on Kafka Cluster:

This procedure describes how to configure Kafka brokers so that the broker listeners are enabled to use OAuth 2.0 authentication by using an authorization server.

> **Note:**
>
> Oracle recommends to use OAuth 2.0 over an encrypted interface through a listener with **tls**. Plain listeners are not recommended.

To enable authentication on the Kafka cluster:

1. In **applications.yaml**, un-comment or add the following configurations:
   a. Set the **authentication.enabled** flag to **true** and update the **loadbalancerhost**, **loadbalancerport** and **ohsHostname** in **$SPEC_PATH/sr/quick/applications.yaml** file.
   b. To use fast local JWT validation, set **useFastLocalJWTvalidation** value to **true** under **kafka-cluster.listeners.authentication**. If not set, the introspection endpoint is used for validation.

```
# The enabled flag is to enable or disable authentication
authentication:
  enabled: true

#Uncomment the below host aliases section and provide hostname to
ipaddresss mappings
#This will add entries to POD's /etc/hosts file for hostname resolution
when DNS and other options are not applicable.
#For more details see https://kubernetes.io/docs/tasks/network/
customize-hosts-file-for-pods/

#hostAliases:
#- ip: <ip-address>
  #hostnames:
  #- <hostname-1> # Ex. quick.sr.ohs.uim.org
```

```
#Sample sub-section for using fast local jwt validation
kafka-cluster:
  listeners:
    authentication:
      useFastLocalJWTvalidation: true
```

2. The Message Bus service uses other configuration values from Kubernetes Secret (**<namespace>-<instance>-oauth-credentials**) and Config Map (**<namespace>-<instance>-oauth-config-cm**) objects from the same namespace. This Secret and Configuration Map Kubernetes objects have to be created before deploying the Message Bus service for authentication. See "Adding Common OAuth Secret and ConfigMap" for creating the secret. The configuration values used are:
   - clientID: The client ID to identify the client.
   - clientSecret: The client secret used for authentication.
   - validIssuerUri: The URI of the token issuer used for authentication.
   - introspectionEndpointUri: The URI of the token introspection endpoint.
   - jwksEndpointUri: The endpoint with public keys of authentication server that has to be used for fast local JWT validation.
   - tlsTrustedCertificate: The trusted certificates for TLS connection to the authorization server.

The following optional values are supported for authentication. See Strimzi documentation https://strimzi.io/docs/operators/in-development/configuring.html#type-KafkaListenerAuthenticationOAuth-reference for details on each value. Add the following optional values as required, under the **kafka-cluster.listeners.authentication** section in applications.yaml file:

```
# Additional optional authentication values
kafka-cluster:
  listeners:
    authentication:
      oauthConfig:
        #Enable or disable audience checking
        checkAudience:
        #Enable or disable issuer checking.  By default issuer is checked
using the value configured by validIssuerUri
        checkIssuer:
        #The audience to use when making requests to the authorization
server's token endpoint
        clientAudience:
        #The scope to use when making requests to the authorization server's
token endpoint
        clientScope:
        #The connect timeout in seconds when connecting to authorization
server
        connectTimeoutSeconds:
        #Enable or disable TLS hostname verification. Default value is false.
        disableTlsHostnameVerification:
        #The read timeout in seconds when connecting to authorization server.
        readTimeoutSeconds:
        #URI of the User Info Endpoint to use as a fallback to obtaining the
user id
        userInfoEndpointUri:
```

```
#Name of the claim from the JWT authentication token
userNameClaim:
```

Using GC Logs

By default, GC logs are disabled, you can enable it and view the logs on `stdout` by using `kubectl logs <kafka-cluster-pod-name>`.

To Enable GC logs, update **$SPEC_PATH/<project>/<instance>/applications.yaml** file as follows:

1. Under `gcLogs` make `enabled` as `true`.
2. Uncomment the `gcLogs` option under `kafka-cluster` to override common values.

```
gcLogs:
  enabled: true
```

> **Note:**
>
> You do not have to configure **fileSize** and **noOfFiles** as the logs are printed on the **stdout**.

Alternate Configuration Options

There are various alternate options for configuring the Message Bus.

Log Level

Kafka uses Apache log4j. By default, it is enabled with `INFO`. Update this for debugging:

```
logging:
  kafka:
    logLevel: INFO
  zookeeper:
    logLevel: INFO
```

Choosing Worker Nodes for Running Message Bus Service

Update the Message Bus service configuration section in the **applications.yaml** file to node affinity or pod affinity and anti-affinity to constrain which nodes your pod can be scheduled. Alternatively, co-locate the pods in same node (or separate) and run either create or upgrade script.

Node Affinity

Node affinity is conceptually similar to **nodeSelector**, that enables you to constrain which nodes your pod can be scheduled, based on the node labels.

There are two types of node affinities:

- Schedule a pod using required node affinity: The scheduler cannot schedule the pod unless the rule is met.
- Schedule a pod using preferred node affinity: The scheduler tries to find a node that meets the rule. If a matching node is not available, the scheduler continues to schedule the pod.

Preferred node affinity

The sample configuration for enabling preferred node affinity is as follows:

```
kafka-cluster:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: name
            operator: In
            values:
            - south_zone
```

Kubernetes pod is scheduled on the node with label name as *south_zone*. If node with label name: *south_zone* is not available, pod will still be scheduled on another node.

Pod Affinity and Anti-Affinity

The Pod Affinity or anti-affinity allows you to constrain which node your pod is eligible to be scheduled, based on the labels on other pods.

Similar to node affinity, there are two types of pod affinity and anti-affinity:

- `requiredDuringSchedulingIgnoredDuringExecution`
- `preferredDuringSchedulingIgnoredDuringExecution`

Pod Affinity

Assign a Kubernetes pod to a node based on the labels on other pods using the Pod Affinity in a Kubernetes cluster. Modify the Kafka cluster override values yaml file.

The sample configuration for enabling the **required** pod affinity is as follows:

```
kafka-cluster:
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: app.kubernetes.io/name
                operator: In
                values:
                  - kafka
          topologyKey: "kubernetes.io/hostname"
```

Kubernetes pod is scheduled on the node which contains a pod with label http://app.kubernetes.io/name: kafka.

Modify the Kafka cluster override values yaml file. The sample configuration for enabling the **preferred** pod affinity is as follows:

```
kafka-cluster:
  affinity:
    podAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 1
          podAffinityTerm:
            labelSelector:
              matchExpressions:
                - key: app.kubernetes.io/name
                  operator: In
                  values:
                  - kafka
            topologyKey: "kubernetes.io/hostname"
```

The Kubernetes pod is scheduled on the node which contains a pod with label http:// app.kubernetes.io/name: kafka. If the node is not available, pod will still be scheduled on another node.

Pod anti-affinity

Assign a Kubernetes pod to a node based on the labels on other pods using pod anti affinity in a Kubernetes cluster.

Modify the Kafka cluster override values yaml file. The sample configuration with **required** pod anti-affinity is as follows:

```
kafka-cluster:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: app.kubernetes.io/name
                operator: In
                values:
                  - kafka
          topologyKey: "kubernetes.io/hostname"
```

Kubernetes pod is scheduled on the node which does not contain a pod with label http:// app.kubernetes.io/name: kafka.

Modify the Kafka cluster's override values yaml file. The sample configuration with **preferred** pod anti-affinity is follows:

```
kafka-cluster:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 1
          podAffinityTerm:
            labelSelector:
              matchExpressions:
                - key: app.kubernetes.io/name
```

```
            operator: In
            values:
            - kafka
        topologyKey: "kubernetes.io/hostname"
```

Kubernetes pod is scheduled on the node which does not contains a pod with label http://app.kubernetes.io/name: kafka. If node is not available, pod will still be scheduled on another node.

Managing Message Bus Metrics

Metrics in Message Bus are configured by enabling the JMX Exporter and Kafka Exporter. JMX Exporter can be enabled to get JVM metrics of Kafka cluster and Kafka Exporter can be enabled on a Kafka cluster to extract additional Prometheus metrics data from Kafka brokers, which is related to offsets, consumer groups, consumer lag, and topics.

See https://strimzi.io/docs/operators/latest/overview.html#metrics-overview_str for more information on metrics from Strimzi.

Enable metrics

Enable Kafka Exporter and JMX Exporter in the **$SPEC_PATH/sr/quick/applications.yaml** file and upgrade or create the Message Bus service. The sample content is as follows:

```
kafka-cluster:
  metrics:
    kafkaExporter:
      enable: true
    jmxExporter:
      enable: true
```

The above configuration exposes the Prometheus metrics for Kafka Brokers, Topics, and Consumer Groups components on metrics end-point on the pods. You can view these details on Prometheus UI by configuring the Scrape job. You can view this information in the form of graphs using the Grafana dashboard.

See https://github.com/danielqsj/kafka_exporter#metrics to see the exposed metrics.

Prometheus and Grafana setup

See Setting Up Prometheus and Grafana for more information.

Adding scrape Job in Prometheus

Add the following Scrape job in Prometheus Server. This can be added by editing the config map used by the Prometheus server:

```
- job_name: Message_bus
  kubernetes_sd_configs:
  - role: pod
    namespaces:
      names:
      - 'sr'
  relabel_configs:
  - separator: ";"
    regex: __meta_kubernetes_pod_label_(strimzi_io_.+)
```

```
    replacement: $1
    action: labelmap
  - source_labels: [__meta_kubernetes_namespace]
    separator: ";"
    regex: (.*)
    target_label: namespace
    replacement: $1
    action: replace
  - source_labels: [__meta_kubernetes_pod_name]
    separator: ";"
    regex: (.*)
    target_label: kubernetes_pod_name
    replacement: $1
    action: replace
  - source_labels: [__meta_kubernetes_pod_node_name]
    separator: ";"
    regex: (.*)
    target_label: node_name
    replacement: $1
    action: replace
  - source_labels: [__meta_kubernetes_pod_host_ip]
    separator: ";"
    regex: (.*)
    target_label: node_ip
    replacement: $1
    action: replace
```

Sample Grafana dashboards

Add the Prometheus data source and import the sample Grafana dashboards from Strimzi github.

The sample Grafana dashboard for Kafka and JMX Exporters can be downloaded from the following links:

- JMX Exporter metrics: https://github.com/strimzi/strimzi-kafka-operator/blob/main/examples/metrics/grafana-dashboards/strimzi-kafka.json
- Kafka Exporter metrics: https://github.com/strimzi/strimzi-kafka-operator/blob/main/examples/metrics/grafana-dashboards/strimzi-kafka-exporter.json

Installing and Configuring Mirror Maker 2.0

This section describes the installation and configuration of Mirror Maker 2.0.

Configuring Source and Target Message Bus (Kafka cluster) Details

Update the *$COMMON_CNTK/samples/messaging-bus/kafka-mirror-maker/values.yaml* with source and target Kafka cluster details as follows:

```
sourceCluster:
    #Source Kafka cluster
    name: sr1-quick1-messaging
    #Bootstarp server for connection to the source Kafka cluster
    bootstrapServers: sr1-quick1-messaging-kafka-bootstrap:9092
  targetCluster:
```

```
#Target Kafka cluster
name: sr2-quick2-messaging
#Bootstarp server for connection to the target Kafka cluster
bootstrapServers: sr2-quick2-messaging-kafka-bootstrap:9092
```

In the above command:

- **sourceCluster.name** is the helm release for source Kafka cluster (sr1-quick1-messaging)
- **sourceCluster.bootstrapServers** is the bootstrap server of source Kafka cluster (sr1-quick1-messaging-kafka-bootstrap:9092)
- **targetCluster.name** is the helm release for target Kafka cluster (sr2-quick2-messaging)
- **targetCluster.bootstrapServers** is the bootstrap server of target Kafka cluster (sr2-quick2-messaging-kafka-bootstrap:9092)

> **Note:**
>
> To enable geo replication between the Kafka clusters from different namespaces, we can use the hostname pattern as **servicename.namespace.svc.cluster.local** while updating
>
> `$COMMON_CNTK/samples/messaging/kafka-mirror-maker/values.yaml`
>
> If the **sr1-quick1-messaging-kafka-bootstrap** service is hosted in Strimzi namespace on 9092 port and the client application in another namespace, then the bootstrap-server URL should be used as **sr1-quick1-messaging-kafka-bootstrap.strimzi.svc.cluster.local**
>
> If the target cluster is in another Kubernetes cluster, you must to use external listener for referring to the boostrap server.
>
> While using `Nodeport`, the worker node IP of the target cluster is to be used as the target cluster bootstrap address along with the exposed nodeport.
>
> While using `Ingress`, the hostname of the target cluster is to be used as target cluster bootstrap address.

Installing Mirror Maker

Run the following command to install Mirror Maker in specific namespace:

```
helm install mirror-maker $COMMON_CNTK/samples/messaging/kafka-mirror-maker/ -
n <namespace> --values $COMMON_CNTK/samples/messaging/kafka-mirror-maker/
values.yaml
```

Validate that Mirror Maker is installed by running the following command:

```
kubectl get pods -n <namespace>
replication-mirror-maker-mirrormaker2-5c6d7dd7d7-r89cj             1/1
Running   0          67m
kubectl get svc -n <namespace>
```

```
replication-mirror-maker-mirrormaker2-api          ClusterIP   <clusterIP>
<none>        8083/TCP                              67m
```

Uninstalling Mirror Maker

Run the following command to uninstall Mirror Maker from specific namespace:

```
helm uninstall mirror-maker -n <namespace>
```

Delete topic mm2-offset-syncs.messaging-test.internal from the source cluster (dev1-messaging)

```
$kubectl -n <SourceKafkaClusterNamespace> run kafka-topic -ti --
image=<STRIMZI_KAFKA_IMAGE_NAME> --rm=true --restart=Never -- bin/kafka-
topics.sh --bootstrap-server <instance>-messaging-kafka-bootstrap:9092 --
delete --topic mm2-offset-syncs.messaging-test.internal
```

Delete topics heartbeats, mirrormaker2-cluster-status, mirrormaker2-cluster-offsets, mirrormaker2-cluster-configs from the target cluster (dev2-messaging)

```
$kubectl -n <TargetKafkaClusterNamespace> run kafka-topic -ti --
image=<STRIMZI_KAFKA_IMAGE_NAME> --rm=true --restart=Never -- bin/kafka-
topics.sh --bootstrap-server <namespace>-<instance>-messaging-kafka-
bootstrap:9092 --delete --topic heartbeats

$kubectl -n <TargetKafkaClusterNamespace> run kafka-topic -ti --
image=<STRIMZI_KAFKA_IMAGE_NAME> --rm=true --restart=Never -- bin/kafka-
topics.sh --bootstrap-server <namespace>-<instance>-messaging-kafka-
bootstrap:9092 --delete --topic mirrormaker2-cluster-status

$kubectl -n <TargetKafkaClusterNamespace> run kafka-topic -ti --
image=<STRIMZI_KAFKA_IMAGE_NAME> --rm=true --restart=Never -- bin/kafka-
topics.sh --bootstrap-server <namespace>-<instance>-messaging-kafka-
bootstrap:9092 --delete --topic mirrormaker2-cluster-offsets

$kubectl -n <TargetKafkaClusterNamespace> run kafka-topic -ti --
image=<STRIMZI_KAFKA_IMAGE_NAME> --rm=true --restar
```

Client Access

Accessing Message Bus in events producer and consumers clients.

Internal Access in the Same namespace for Plain

When the message producer or consumer applications are in same namespace as the Message Bus service then they can access the Kafka cluster using the Bootstrap Kubernetes service object name and port.

Run the following command to test the standalone **producer**. Here the project namespace is **sr** and instance is **quick**.

```
$kubectl -n sr run kafka-producer-plain -ti \
--image=<STRIMZI_KAFKA_IMAGE_NAME> \
```

```
--rm=true --restart=Never \
-- bin/kafka-console-producer.sh \
--bootstrap-server sr-quick-messaging-kafka-bootstrap:9092 \
--topic ora-test-topic
```

Type a few lines of text and each ENTER sends a message to Kafka broker. Type **CTRL-C** to quit.

Run the following command to test the standalone **consumer**. Here the project namespace is **sr** and instance is **quick**.

```
$kubectl -n sr run kafka-consumer-plain -ti \
--image=<STRIMZI_KAFKA_IMAGE_NAME> \
--rm=true --restart=Never \
-- bin/kafka-console-consumer.sh \
--bootstrap-server sr-quick-messaging-kafka-bootstrap:9092 \
--group ora-uim-consumer-test --isolation-level read_committed \
--topic  ora-test-topic --from-beginning
```

You get responses after the validation is successful.

Internal Access in a Different namespace for Plain

When the massage producer or consumer applications are in different namespace than the Message Bus service then they can access the Kafka cluster using the bootstrap service name and port but need to suffix ***<namespace>.svc.cluster.local*** to the service name.

See "Internal access - same namespace - plain" section on running the standalone console test producer and consumer pods for testing. Replace the bootstrap-server url with **sr-quick-messaging-kafka-bootstrap.sr.svc.cluster.local**, where the namespace is **sr** and instance is **quick**.

Internal Access in the Same namespace for Authentication

When the message producer or consumer applications are in same namespace as the Message Bus service then they can access the Kafka cluster using the bootstrap Kubernetes service object name and port.

Create a test client pod definition.

1. Copy the following YAML content into the bastion host (or worker node) as **mb-test-client-deployment.yaml** file.
2. Update the **hostAliases** section according to your OAuth service environment.
3. Update the **STRIMZI_KAFKA_IMAGE_NAME**.
4. Update the **OAUTH Endpoint**, **Client Id** and **Secret**.
5. Update the **OAUTH Endpoint**, **Client Id**, **Client Secret**, **Scope**, **Audience**, and anything else that are applicable to your client configuration

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name:  mb-test-auth-client-deployment
  labels:
    app:  mb-test-auth-client
spec:
```

```
  replicas: 1
  selector:
    matchLabels:
      app:  mb-test-auth-client
  template:
    metadata:
      labels:
        app:  mb-test-auth-client
    spec:
#     <Uncomment below and replace with your bootstrap and brokers DNS
names>
      #hostAliases:
      #- ip: <LOADBALANCER_IP>
      #  hostnames:
      #  - "<OHS_HOSTNAME>"
      containers:
      - name:  mb-test-client
        image: <STRIMZI_KAFKA_IMAGE_NAME>
        command:
        - "tail"
        - "-f"
        - "/dev/null"
        imagePullPolicy: IfNotPresent
        env:
        - name: OAUTH_TOKEN_ENDPOINT_URI
          value: <Update the OAUTH_TOKEN_ENDPOINT_URI>
        - name: OAUTH_CLIENT_ID
          value: <Update the OAUTH_CLIENT_ID>
        - name: OAUTH_CLIENT_SECRET
          value: <Update the OAUTH_CLIENT_SECRET>
       # - name: OAUTH_SCOPE
       #   value: <Uncomment and update OAUTH_SCOPE>
       # - name: OAUTH_AUDIENCE
       #   value: <Uncomment and update OAUTH_AUDIENCE>
        ports:
        - containerPort: 9090
          name: http
          protocol: TCP
```

Create the authentication properties in a file (mb_test_client.properties).

```
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLogin
Module required;
security.protocol=SASL_PLAINTEXT
sasl.mechanism=OAUTHBEARER
sasl.login.callback.handler.class=io.strimzi.kafka.oauth.client.JaasClientOaut
hLoginCallbackHandler
```

Run the test client container and provide authentication properties

```
#Apply the test client pod definition in the namespace (say "sr").
$kubectl apply -f mb-test-client-deployment.yaml -n sr

#Get the newly created pod name
$kubectl get pod -n sr | grep mb-test-auth-client-deployment
```

```
#Sample Output
#mb-test-auth-client-deployment-******-****         1/1     Running
0          98s

#Copy the mb_authentication.properties file into the pod
$kubectl -n sr cp mb_test_client.properties mb-test-auth-client-deployment-
******-****:/home/kafka/mb_test_client.properties
```

Test for message bus producer client:

- Start an interactive shell process in the test client pod
- Export the environment variables needed for the authentication
- Run the console producer command.
- Enter some string messages

```
#Get the newly created pod name
kubectl get pod -n sr | grep mb-test-auth-client-deployment

#Exec into the newly created pod
kubectl exec -it mb-test-auth-client-deployment-******-**** -n sr -- bash

#Run the following test console producer
bin/kafka-console-producer.sh \
--producer.config /home/kafka/mb_test_client.properties \
--bootstrap-server sr-quick-messaging-kafka-bootstrap:9092 \
--topic ora-test-topic
```

Test for message bus consumer client:

- Start an interactive shell process in the test client pod
- Export the environment variables needed for the authentication
- Run the console consumer command.
- You will see the previous string messages of producer

```
#Get the newly created pod name
kubectl get pod -n sr | grep mb-test-auth-client-deployment

#Sample Output
#mb-test-auth-client-deployment-******-****         1/1     Running
0          98s

#Exec into the newly created pod
kubectl exec -it mb-test-auth-client-deployment-******-**** -n sr -- bash

#Run the following test console consumer
bin/kafka-console-consumer.sh \
--consumer.config /home/kafka/mb_test_client.properties \
--bootstrap-server sr-quick-messaging-kafka-bootstrap:9092 \
--topic ora-test-topic \
--from-beginning
```

External ingress access - SSL and Authentication

The external access to Message Bus is provided through Ingress controller (Traefik or Generic) with TLS enabled. The following must be performed in clients for testing:

- Export and import the Message Bus service (that is sr-quick-messaging-cluster-ca-cert, where sr is namespace and quick is instance) certificate into clients.
- Export and import the certificate of OAuth service into the clients.

> **Note:**
>
> This is optional and is required only if OAuth is enabled for SSL.

- Update the bootstrap and brokers DNS names with load balancer IP in the **etc/hosts** file of clients (that is, event producer or consumer applications).
- Update the DNS name of OAuth service with load balancer IP in **/etc/hosts** file of clients.

> **Note:**
>
> This is optional and is required only if the OAuth service requires DNS name to access.

- Run the producer or consumer script with SSL and Authentication details

In the following section, the external ingress access test is provided with Strimzi Kafka container. If you want to test the client code without Kubernetes cluster then you can download the Apache Kafka and perform the same.

Add Message Bus service and OAuth service certifications to trust store. See **Import/export of TLS certificates** section.

```
#Run the below command to export and import the Message Bus service
certificate into the trust store (mb-cert-keystore.jks) file.
$COMMON_CNTK/scripts/export-cluster-cert.sh -p sr -i quick -l . -k ./mb-test-
client-cert-keystore.jks -a mb-cert

#Get the OAuth (OAM) service certificate and import into trust store (mb-test-
client-cert-keystore.jks) file (Optional, needed if OAuth is SSL)
keytool -importcert -alias oauth-server -file <Path to OAuth Server
certificate, the .pem file> -keystore ./mb-test-client-cert-keystore.jk --
trustcacerts -noprompt
```

Create the following authentication properties in a file (**mb_test_client.properties**).

```
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLogin
Module required;
security.protocol=SASL_SSL
sasl.mechanism=OAUTHBEARER
sasl.login.callback.handler.class=io.strimzi.kafka.oauth.client.JaasClientOaut
hLoginCallbackHandler
ssl.endpoint.identification.algorithm=
```

Create a test client pod definition.

1. Copy the following YAML content into the bastion host (or worker node) as "**mb-test-client-deployment.yaml**" file.
2. Update the Strimzi Kafka image.
3. Update the hostAliases section according to your OAuth and Message Bus service setup. This will add entries to /etc/hosts file.
4. Update the **OAuth Endpoint**, **Client Id**, **Client Secret**, and **Trust Store Password** in **env** section.

> **Note:**
>
> You can override the value of **subDomainNameSeparator**. The default is **.**. This value can be changed as **"-"** to match the wild card pattern of SSL certificates.
>
> To override, uncomment and change this value in applications.yaml. See "Using Wild Card Certificates" for more information.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name:  mb-test-client-deployment
  labels:
    app: mb-test-client
spec:
  replicas: 1
  selector:
    matchLabels:
      app:  mb-test-client
  template:
    metadata:
      labels:
        app:  mb-test-client
    spec:
#     <Uncomment below and replace with your bootstrap and brokers DNS names>
#     hostAliases:
#     - ip: <Replace with your LOADBALANCER_IP>
#       hostnames:
#       - "<INSTANCE.PROJECT.messaging.broker0.uim.org>"
#       - "<INSTANCE.PROJECT.messaging.brokerN.uim.org>"
#       - "<INSTANCE.PROJECT.messaging.bootstrap.uim.org>"
#       - "<Replace with OHS_HOSTNAME>"
      containers:
      - name:  mb-test-client
        image: quay.io/strimzi/kafka:0.34.0-kafka-3.4.0
        command:
        - "tail"
        - "-f"
        - "/dev/null"
        imagePullPolicy: IfNotPresent
        env:
        - name: OAUTH_TOKEN_ENDPOINT_URI
```

```
          value: <Replace with your OAUTH_TOKEN_ENDPOINT_URI>
        - name: OAUTH_CLIENT_ID
          value: <Replace with your OAUTH_CLIENT_ID>
        - name: OAUTH_CLIENT_SECRET
          value: <Replace with your OAUTH_CLIENT_SECRET>
        #- name: OAUTH_SCOPE
        #  value: <Uncomment and replace with your OAUTH_SCOPE>
        #- name: OAUTH_AUDIENCE
        #  value: <Uncomment and replace with yours OAUTH_AUDIENCE>
        - name: KAFKA_OPTS
          value: " \
                 -Djavax.net.ssl.trustStore=/home/kafka/mb-test-client-cert-
keystore.jks \
                 -Djavax.net.ssl.trustStorePassword=<Replace with your store
password> \
                 -Djavax.net.ssl.trustStoreType=JKS"
      ports:
        - containerPort: 9090
          name: http
          protocol: TCP
```

Run the test client container and apply readiness for authentication and SSL.

```
#Apply the test client pod definition in the namespace (say "sr").
$kubectl apply -f mb-test-client-deployment.yaml -n sr

#Get the newly created pod name
$kubectl get pod -n sr | grep mb-test-client-deployment

#Sample Output
#mb-test-client-deployment-******-****         1/1     Running   0
98s

#Copy the certificate store into the newly created pod.  Replace the pod name
below
kubectl -n sr cp mb-test-client-cert-keystore.jks <Replace with mb-test-
client-deployment pod name>:/home/kafka/mb-test-client-cert-keystore.jks

#Copy the mb_test_client.properties file into the POD
kubectl -n sr cp mb_test_client.properties <Replace with mb-test-client-
deployment pod name>:/home/kafka/mb_test_client.properties
```

Start a shell session inside container for console **test producer**.

```
#Get the newly created pod name
$kubectl get pod -n sr | grep mb-test-auth-client-deployment

#Sample Output
#mb-test-auth-client-deployment-******-****         1/1     Running
0          98s

#Exec into the newly created pod
kubectl exec -it<Replace with mb-test-client-deployment pod name> -n sr --
bash
```

```
#Run the following producer command
bin/kafka-console-producer.sh \
--producer.config /home/kafka/mb_test_client.properties \
--bootstrap-server quick.sr.messaging.bootstrap.uim.org:30443 \
--topic ora-test-topic
```

Start start a shell session inside container for console **test consumer**:

```
#Exec into the newly created pod
kubectl exec -it <Replace with mb-test-client-deployment pod name> -n sr --
bash

#Run the following producer command.  Replace the bootstrap-server url
accordingly to your environment
bin/kafka-console-consumer.sh \
--consumer.config /home/kafka/mb_test_client.properties \
--bootstrap-server sthatipa.sr.messaging.bootstrap.uim.org:30443 \
--consumer-property group.id=test-client-service \
--topic ora-test-topic --from-beginning
```

Clean-up the newly created test pod:

```
kubectl delete -f mb-test-client-deployment.yaml -n sr
```

External node port access

The nodeport listener type allows the external access from outside of the Kubernetes cluster using the load balancer or Kubernetes worker node ip address and nodePort(port of worker node).

The Bootstrap URL is constructed with worker node IP Address and node port of bootstrap service.

Get the host port of the external bootstrap service using the following command:

```
$kubectl get service sr-quick-messaging-kafka-nodeport-bootstrap -
o=jsonpath='{.spec.ports[0].nodePort}{"\n"}' -n sr

Output: 32100
```

Get the IP Address of the Kubernetes worker node. Replace the <NODE_NAME> in the following with your node name:

```
$kubectl get node <NODE_NAME> -o=jsonpath='{range .status.addresses[*]}{.type}
{"\t"}{.address}{"\n"}' -n sr

Output:
InternalIP  100.xx.xx.142
Hostname    *********
```

Update the Kafka cluster Bootstrap URL as **100.xx.xx.142:32100** in the events producer and consumer applications.

To access with plain, see "Internal access - same namespace - plain" section. Replace the bootstrap URL with above constructed one.

To access with Authentication, see "Internal access - same namespace - authentication" section. Replace the bootstrap URL with above constructed one.

To access with SSL and Authentication, see "External ingress access - SSL & Authentication" section. Replace the bootstrap URL with above constructed one.

Import/export of TLS certificates

To enable TLS encrypted access, the ca-certs of Kafka cluster is needed to be extracted and imported into key store and the location of that key store is used as the producer or consumer properties in events application.

Export the ca-certs of the Kafka cluster using the following command:

```
$COMMON_CNTK/scripts/export-cluster-cert.sh -p <Namespace of kafka cluster> \
-i <instance name of kafka cluster> \
-l <directory to export clustercerts temporarily> \
-k <keystore-location> \
-a <alias for cert>
```

For example:

```
$COMMON_CNTK/scripts/export-cluster-cert.sh -p sr -i quick -l . -k ./mb-cert-keystore.jks -a mb-sr-quick-cert
```

The export-cluster-cert.sh script creates JKS type truststore by default in the provided key store location. If any other truststore type is created, specify that as producer or consumer property while running the clients. These exported artifacts can be used in Kafka client applications.

> **Note:**
>
> If custom certificates were used during cluster creation, then these can be directly provided through a keystore than extracting the generated certs.

Using custom certificates

Custom certificates can be used while creating the Kafka cluster:

Prerequisites:

- Certificates and keys are to be in PEM format.
- Key should not be encrypted. Encrypted keys are not supported since they need user interaction for entering the passphrase during access.

Creating a custom certificate

To create a custom certificate, see Self-signed SSL Certificates.

Create Kubernetes secret

Run the following command by replacing the placeholders:

```
kubectl create secret generic <secret-name> --from-file=<key-file-name> --
from-file=<certificate-file-name>
 For example:
kubectl create secret generic myCustomCertSecret --from-file=commonkey.pem --
from-file=commoncert.pem
```

Update Kafka Cluster configuration

Update the customCerts configuration section in Kafka cluster's override values yaml file:

```
kafka-cluster:
  ## to enable custom or owned certs for tls please create a kubernetes
secret with the cert and key if not already present, uncomment the below
section and add respective values.
    ## please be advised that encrypted keys are not supported since they
require user interaction for the passphrase
    customCerts:
      # Secret in which cert and key are present
      secretName: <secret-name created above>
      certName: <certificate file used in the secret created above>
      keyName: <key-file used in the secret created above>
```

Configuring Message Bus Listeners

Message Bus has three listeners (internal, ingress and nodeport) to access the service. These are described the in following sections.

Message Bus Internal Listener

The following is the configuration for **internal** listener type which can be commented or uncommented.

```
 kafka-cluster:
  listeners:
  # plain is for internal access within the same k8s cluster.
    internal:
```

From same namespace in cluster

This is an internal access method that is used by the message producer or consumer clients (or applications) when they are deployed in same namespace as the Message Bus service. This is enabled by default with **internal** listener type. To access the Message Bus, the producer or consumer applications must get the Bootstrap service URL of the Kafka cluster.

To get the Bootstrap service URL of the Kafka cluster run the following command:

```
kubectl get svc -n sr | grep sr-quick-messaging-kafka-bootstrap
```

```
sr-quick-messaging-kafka-bootstrap       ClusterIP   <clusterIP>
<none>        9091/TCP,9092/TCP
```

> **Note:**
>
> The project namesapce is sr and instance is quick.

Use the **sr-quick-messaging-kafka-bootstrap:9092** URL in the producer and consumer client configuration in the applications.

From another namespace in cluster

This is an internal access method which is used by the producer or consumer client applications when they are deployed in different namespace than the message-bus service. This is enabled by default with internal listener type. To access the Message Bus, the producer or consumer client applications have to get the Bootstrap service URL of the Kafka cluster and convert the URL pattern as **serviceName.namespace.svc.cluster.local**.

If the *sr-quick-messaging-kafka-bootstrap* service is hosted in **sr** namespace on 9092 port and the client applications from different namespace can access the Kafka cluster with Bootstrap URL as ***sr-quick-messaging-kafka-bootstrap.sr.svc.cluster.local:9092***

Message Bus Ingress Listener

This is an external access method which is used by message producer or consumer applications when they are deployed outside of the Kubernetes cluster. This is disabled by default and must be enabled in the **applications.yaml**. This external access is provided through the Traefik Ingress Controller and Generic Ingress Controller to the Kafka cluster. To enable this external access, the ingress listener type configuration must be enabled in the Kafka cluster configuration yaml file.

Ingress listener type

Un-comment the ingress lister type section in applications.yaml file to expose the Message Bus Service outside of Kubernetes cluster. Ingress controller (Traefik or Generic) should be deployed in order for this ingress listener type to work and Message Bus namespace must be registered with Traefik operator. In case of Generic Ingress, set **ingress.className** according to your Generic Ingress Controller.

In case of Generic Ingress controller (nginx), annotations given under the **kafka-cluster.listeners.ingress.annotations** tag in applications.yaml are mandatory.

```
# To expose the kafka-cluster to external kafka clients via ingress
controller uncomment the following and modify accordingly. # Valid values are
TRAEFIK, GENERIC
ingressController: "TRAEFIK"

#ingress:
#  #specify className field for ingressClassName of generic ingress
controller.
#  #In case of nginx the default values is nginx
#  className: "nginx"

#provide loadbalancer port
# if TLS is enabled in global section, then loadbalancerport will be used as
external port for Generic or Traefik
```

```
loadbalancerport: <loadBalancer-port>

kafka-cluster
  listeners:
    ingress:
      # if TLS is Disabled in global, then ingressSslPort will be used as
external port.
      ingressSslPort: <LoadBalancer_SSL_Port>
      # If using Generic Ingress controller, below given annotations are
mandatory for Message-Bus external access.
      # These annotations are required for nginx ingress controller in
Message-Bus.
      annotations:
        nginx.ingress.kubernetes.io/ingress.allow-http: "false"
        nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
        ingress.kubernetes.io/ssl-passthrough: "true"
        nginx.ingress.kubernetes.io/ssl-passthrough: "true"
```

In external producer or consumer messaging clients (or applications), the following must be done to access the Kafka cluster through Ingress controller.

- The Bootstrap server and advertised broker host names must be configured in DNS at client side.
- Import the TLS certificate and trust stores from the Kafka cluster into client configurations.
- Add required additional properties in Kafka producer or consumer client configuration.

DNS settings in client applications host

The Bootstrap server host name and advertised broker host names must be configured in /etc/hosts file in producer and consumer client applications with the Traefik or Load Balancer IP Address. Hostnames are pre-configured when deployed with ingress listener type enabled with the following pattern:

```
bootstrap-server:  <kafka-cluster-instance-name>.<kafka-cluser-project-
name>.messaging.bootstrap.uim.org
broker-0:          <kafka-cluster-instance-name>.<kafka-cluser-project-
name>.messaging.broker0.uim.org
broker-1:          <kafka-cluster-instance-name>.kafka-cluser-project-
name>.messaging.broker1.uim.org

For example if a instance is quick and namesapce is sr then the hostnames
will be as follows:
bootstrap-server:  quick.sr.messaging.bootstrap.uim.org
broker-0:          quick.sr.messaging.broker0.uim.org
broker-1:          quick.sr.messaging.broker1.uim.org
```

> **Note:**
>
> You can override the value of **subDomainNameSeparator**. The default value is **"."**, This value can be changed to **"-"** to match the wild card pattern of SSL certificates.
>
> To override the value, uncomment and change it in **applications.yaml** as follows:
>
> ```
> #subDomainNameSeparator: "."
>
> #Example hostnames for "-" : quick-sr-messaging-bootstrap.uim.org
> ```

Importing certificates into client applications

See the "Import/export of TLS certificates" section in “Client Access” section for exporting the ca-certs of Kafka cluster to producer or consumer applications.

Message Bus NodePort Listener

This is another external access method which is used by events producer or consumer client applications when they are deployed out-side of the Kubernetes cluster and wants to access the message-bus service without ingress controller.

Node port

The following configuration in the application yaml file allows exposing the nodeport listener type to access the Message Bus externally with tls and OAuth 2.0 Authentication.

```
Kafka-cluster:
  listeners:
    #To expose the kafka-cluster to external kafka clients without ingress
controller, uncomment the following section and modify accordingly.
    nodeport:
      tls: true
      # if need to expose on a static nodeport, please uncomment the below
nodePort key and provide values.
      nodePort: 32100
      authentication: true
```

When the tls is enabled the certificates of the Kafka cluster must be imported in the events producer and consumer clients to access the Kafka cluster.

See the "Import/export of TLS certificates" section in “Client Access” section for exporting the auto-generated ca-certs of Kafka cluster.

Debugging and Troubleshooting

NotEnoughReplicasException

When you get the **org.apache.kafka.common.errors.NotEnoughReplicasException**: Messages are rejected since there are fewer in-sync replicas than required. The reason could be that the topics replicas is not meeting the default minInsyncReplicas value configured in the Message Bus service.

Asynchronous auto-commit of offsets failed

When you get the following error in the logs (for example: UTIA Consumer). To resolve this make sure that **max.polling.interval.ms** is always greater than the last poll or else reduce the **max.poll.records**.

```
[Consumer clientId=consumer-ora-uim-topology-service-2, groupId=ora-uim-
topology-service] Asynchronous auto-commit of offsets failed: Offset commit
cannot be completed since the consumer is not part of an active group for
auto partition assignment; it is likely that the consumer was kicked out of
the group.. Will continue to join group.
```

Add these additional properties in the YAML file under the **mp.messaging.connector.helidon-kafka** section with override values.

```
mp.messaging:
  connector:
    helidon-kafka:
      # The following are default global configuration values which effects
for all the consumer groups.
      max.polling.interval.ms: 300000
      max.poll.records: 500

  # The following are channel specific configuration values
  incoming:
    # The toInventoryChannel effects only for ora-uim-topology-service
consumer group
    # uncomment and update the specific values
    #toInventoryChannel:
      #max.polling.interval.ms: 300000
      #max.poll.records: 500

    # The toFaultChannel effects only for ora-uim-topology-retry-service
consumer group
    # Uncomment and update the specific values
    #toRetryChannel:
      #max.polling.interval.ms: 300000
      #max.poll.records: 200

    # The toDltChannel effects only for ora-uim-topology-dlt-service consumer
group
    # uncomment and update the specific values
    #toDltChannel:
      #max.polling.interval.ms: 300000
      #max.poll.records: 100
```

Performance Tuning: Consumer Configurations

The following are some consumer configuration properties in message consumers which are related to performance. See https://kafka.apache.org/documentation/#consumerconfigs for all available consumer config properties.

- max.poll.records (default=500) defines the maximum number of messages that a consumer can poll at once.

- max.partition.fetch.bytes (default=1048576) defines the maximum number of bytes that the server returns in a poll for a single partition.
- max.poll.interval.ms (default=300000) defines the time a consumer must process all messages from a poll and fetch a new poll afterward. If this interval is exceeded, the consumer leaves the consumer group.
- http://heartbeat.interval.ms (default=3000) defines the frequency with which a consumer sends heartbeats.
- http://session.timeout.ms (default=10000) defines the time a consumer must send a heartbeat. If no heartbeat was received in that timeout, the member is considered dead and leaves the group.

Managing Consumer Groups

For more list of options available on the consumer groups see the apache kafka managing consumer groups section. The following sub-sections list some significant operations. See "Message Bus Client Access" for more information.

List consumer groups

```
#Exec into running message bus test client pod
kubectl exec -it mb-test-auth-client-deployment-******-**** -n sr -- bash

#Run the below command to list all the consumer groups
bin/kafka-consumer-groups.sh \
--command-config /home/kafka/mb_test_client.properties \
--bootstrap-server <Your Bootstrap Server URL> \
--list
```

Describe consumer group

```
#Exec into running Kafka admin client pod
kubectl exec -it mb-test-auth-client-deployment-******-**** -n sr -- bash

#Run the below command to describe specific consumer group to check topics,
partitions, offsets
#Replace the command-config, bootstrap, group values accordingly
bin/kafka-consumer-groups.sh \
--command-config /home/kafka/mb_test_client.properties \
--bootstrap-server <Your Bootstrap Server URL> \
--group test-client-service \
--describe
```

Reset offset of a consumer group

```
#Exec into running Kafka admin client pod
kubectl exec -it mb-test-auth-client-deployment-******-**** -n sr -- bash

#Run the below command to reset offset for consumer group for topic to
latest.  See Apache Kafka documentation for other available options.
#Replace the command-config, bootstrap, group and topic values accordingly
bin/kafka-consumer-groups.sh \
--command-config /home/kafka/mb_test_client.properties \
--bootstrap-server <Your Bootstrap Server URL> \
--group test-client-service \
```

```
--reset-offsets \
--topic ora-test-topic \
--to-latest \
--execute
```

Topics

For more detailed list of operations available on the topics see the "Apache Kafka Operations".The following sub-sections list some significant operations. See "Message Bus Client Access" for more information.

Create

Create a topic with three partitions and two replications.

```
#Exec into running Kafka admin client pod
kubectl exec -it mb-test-auth-client-deployment-******-**** -n sr -- bash


#Run the below command to create a topic
bin/kafka-topics.sh \
 --command-config /home/kafka/mb_test_client.properties \
 --bootstrap-server <Your Bootstrap Server URL> \
 --create \
 --topic replicated-2 \
 --replication-factor 2 \
 --partitions 3
```

List

To list all topics:

```
#Exec into running Kafka admin client pod
kubectl exec -it mb-test-auth-client-deployment-******-**** -n sr -- bash


#Run the below command to list all the topic
bin/kafka-topics.sh \
 --command-config /home/kafka/mb_test_client.properties \
 --bootstrap-server <Your Bootstrap Server URL> \
 --list
```

Describe

Describes the topic and its partition count, replicas factory along with leaders for the partition.

```
#Exec into running Kafka admin client pod
kubectl exec -it mb-test-auth-client-deployment-******-**** -n sr -- bash

#Run the below command to describe the topic
bin/kafka-topics.sh \
 --command-config /home/kafka/mb_test_client.properties \
 --bootstrap-server <Your Bootstrap Server URL> \
 --topic replicated-2 \
 --describe
```

```
#Sample output
Topic: replicated-2      TopicId: vyalpPOmR0CtYt7Sc-gbxA
PartitionCount: 3               ReplicationFactor: 2          Configs:
min.insync.replicas=1,message.format.version=3.0-IV1

Topic: replicated-2     Partition: 0     Leader: 1      Replicas: 1,0
Isr: 1,0
Topic: replicated-2     Partition: 1     Leader: 0      Replicas: 0,1
Isr: 0,1
Topic: replicated-2     Partition: 2     Leader: 1      Replicas: 1,0
Isr: 1,0
```

Alter

You can alter a topic and increase the partitions to 2.

```
#Exec into running Kafka admin client pod
kubectl exec -it mb-test-auth-client-deployment-******-**** -n sr -- bash

#Run the below command to alter the topic  bin/kafka-topics.sh \
 --command-config /home/kafka/mb_test_client.properties \
 --bootstrap-server <Your Bootstrap Server URL> \
 --alter \
 --topic <Your Topic Name> \
 --partitions 1
```

Reassignment

The partition reassignment tool can also be used to selectively move replicas of a partition to a specific set of brokers. In the following example the partitions for topic (**replicated-2**) are reassigned to different brokers.

See the "Message Bus Client Access"section for more information on running the message bus test pod with required configuration such as Authentication and SSL.

Create a file called custom-reassignment.json file a terminal

```
{"version":"1", "partitions":
[{"topic":"replicated-2","partition":"0","replicas":"[0,1]"},
{"topic":"replicated-2","partition":1,"replicas":"[1,2]"},
{"topic":"replicated-2","partition":"2","replicas":"[0,2]"}]}
```

Run the following commands for reassignment:

```
#Copy the custom-reassignment.json file into the newly created pod under /
home/kafka directory
$kubectl cp custom-reassignment.json mb-test-auth-client-deployment-******-
****:/home/kafka/custom-reassignment.json -n kafka


#Exec into running test pod
kubectl exec -it mb-test-auth-client-deployment-******-**** -n sr -- bash
#Cd directory to /home/kafka

#Validate the topic ("replicated-2"
```

```
/opt/kafka/bin/kafka-topics.sh \
--command-config /home/kafka/mb_test_client.properties \
--bootstrap-server <Your Bootstrap Server URL> \
--topic replicated-2 --describe
Topic: replicated-2     TopicId: vyalpPOmR0CtYt7Sc-gbxA PartitionCount:
3       ReplicationFactor: 2    Configs:
min.insync.replicas=1,message.format.version=3.0-IV1
        Topic: replicated-2     Partition: 0    Leader: 1       Replicas:
1,0   Isr: 1,0
        Topic: replicated-2     Partition: 1    Leader: 1       Replicas:
0,1   Isr: 1,0
        Topic: replicated-2     Partition: 2    Leader: 1       Replicas:
1,0   Isr: 1,0

#Run reassign-partitions script to reassign the partitions according to the
json file
$/opt/kafka/bin/kafka-reassign-partitions.sh --bootstrap-server dev-messaging-
kafka-bootstrap:9092 --reassignment-json-file custom-reassignment.json --
execute

Current partition replica assignment

{"version":1,"partitions":[{"topic":"replicated-2","partition":0,"replicas":
[1,0],"log_dirs":["any","any"]},
{"topic":"replicated-2","partition":1,"replicas":[0,1],"log_dirs":
["any","any"]},{"topic":"replicated-2","partition":2,"replicas":
[1,0],"log_dirs":["any","any"]}]}

Save this to use as the --reassignment-json-file option during rollback
Successfully started partition reassignments for
replicated-2-0,replicated-2-1,replicated-2-2

#Verfify the reassignment status
$/opt/kafka/bin/kafka-reassign-partitions.sh --bootstrap-server dev-messaging-
kafka-bootstrap:9092 --reassignment-json-file custom-reassignment.json --
verify

Status of partition reassignment:
Reassignment of partition replicated-2-0 is complete.
Reassignment of partition replicated-2-1 is complete.
Reassignment of partition replicated-2-2 is complete.

Clearing broker-level throttles on brokers 0,1,2
Clearing topic-level throttles on topic replicated-2

# Validate the partition assignments
$/opt/kafka/bin//kafka-topics.sh \
  --command-config /home/kafka/mb_test_client.properties \
  --bootstrap-server <Your Bootstrap Server URL> \
  --topic replicated-2 --describe
Topic: replicated-2     TopicId: vyalpPOmR0CtYt7Sc-gbxA PartitionCount:
3       ReplicationFactor: 2    Configs:
min.insync.replicas=1,message.format.version=3.0-IV1
        Topic: replicated-2     Partition: 0    Leader: 1       Replicas:
0,1   Isr: 1,0
        Topic: replicated-2     Partition: 1    Leader: 1       Replicas:
```

```
1,2   Isr: 1,2
        Topic: replicated-2     Partition: 2    Leader: 0       Replicas:
0,2   Isr: 0,2
```

5

Deploying the Unified Topology for Inventory and Automation Service

This chapter describes how to deploy and manage UTIA service.

Overview of UTIA

Oracle Communications Unified Topology for Inventory and Automation (UTIA) represents the spatial relationships among your inventory entities for the inventory and network topology.

- UTIA provides a graphical representation of topology where you can see your inventory and its relationships at the level of detail that meets your needs.

  See UTIA Help for more information about the topology visualization.

Use UTIA to view and analyze the network and service data in the form of topology diagrams. UTIA collects this data from UIM.

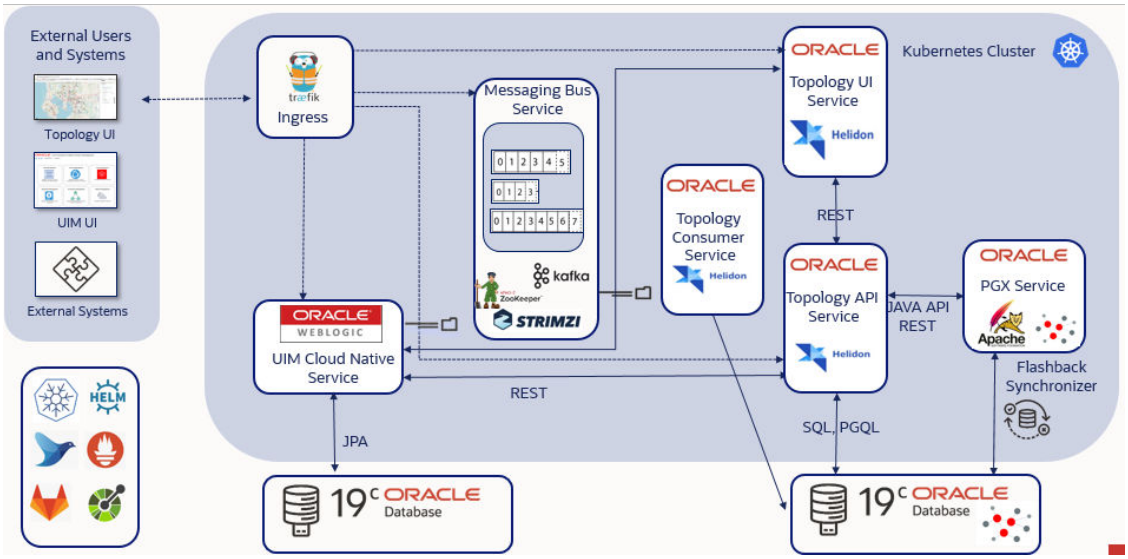
You use UTIA for the following:

- Viewing the networks and services, along with the corresponding resources, in the form of topological diagrams and graphical maps.
- Planning the network capacity.
- Tracking networks.
- Viewing alarm information.

UTIA Architecture

Figure 5-1 shows a high-level architecture of the UTIA service.

Figure 5-1 UTIA Architecture



UIM as the Producer

UIM communicates with the Topology Service using REST APIs and Kafka Message Bus. UIM is the Producer for Create, Update and Delete operations from UIM that impact Topology. UIM uses REST APIs to communicate directly with the UTIA Service while building the messages and can also continue processing when the Topology Service is unavailable.

Topology as the Consumer

The UTIA service is a consumer for inventory system and assurance system messages. UTIA processes multiple message events including TopologyNodeCreate, TopologyNodeUpdate, TopologyNodeDelete, TopologyEdgeCreate, TopologyEdgeUpdate, TopologyEdgeDelete, TopologyFaultEventCreate, TopologyFaultEventUpdate, TopologyPerformanceEventCreate, TopologyPerformanceEventUpdate.

The service information is updated using the TopologyProfileCreate, TopologyProfileUpdate, and TopologyProfileDelete events.

Topology Graph Database

The UTIA Service communicates to the Oracle Databases using the Oracle Property Graph feature with PGQL and standard SQL. It can communicate directly to the database or with the In-Memory Graph for high performance operations. This converged database feature of Oracle Database makes it possible to utilize the optimal processing method with a single database. The Graph Database is isolated and a separate Pluggable Database (PDB) from the UIM Database but runs on the same 19c version for simplified licensing.

Topology In-Memory Database

The UTIA Service also uses the Oracle Labs Parallel Graph AnalytiX (PGX) In-Memory database. The PGX server is used for Path Analysis and is configured for periodic updates.

UTIA User Interface

UTIA provides a graphical representation of topology where you can see your inventory and its relationships at the level of detail that meets your needs. UTIA is built using Oracle Redwood Design System.

Creating UTIA Images

You must install the prerequisite software and tools for creating UTIA images.

Prerequisites for Creating UTIA Images

You require the following prerequisites for creating UTIA images:

- Podman on the build machine if Linux version is greater than or equal to 8.
- Docker on the build machine if Linux version is lesser than 8
- Unified Topology Builder Toolkit (ref about the deliverables)
- Install Maven and update path variable with Maven Home.

  ```
  Set PATH variable export PATH=$PATH:$MAVEN_HOME/bin
  ```

- Java, installed with JAVA_HOME set in the environment.

  ```
  Set PATH variable export PATH=$PATH:$JAVA_HOME/bin
  ```

- Bash, to enable the `<tab>` command complete feature.

See *UIM Compatibility Matrix* for details about the required and supported versions of these prerequisite software.

Configuring Unified Topology Images

The dependency manifest file describes the input that goes into the Unified Topology images. It is consumed by the image build process. The default configuration in the latest manifest file provides the necessary components for creating the Unified Topology images easily. See "About the Manifest File" for more information.

Creating Unified Topology Service Images

To create the Unified Topology service images:

> **Note:**
>
> See *UIM Compatibility Matrix* for the latest versions of software.

1. Go to WORKSPACEDIR.
2. Download graph server war file from Oracle E-Delivery (https://www.oracle.com/database/technologies/spatialandgraph/property-graph-features/graph-server-and-client/graph-server-and-client-downloads.html → Oracle Graph Server <*version*>→ Oracle Graph Webapps <*version*> for (Linux x86-64)) and copy graph server war file to directory $WORKSPACEDIR/unified-topology-builder/staging/downloads/graph. Ensure only one copy of PGX.war exists in …/downloads/graph path.

   > **Note:**
   >
   > The log level is set to debug by default in graph server war file. If required, update the log level to **error/info in graph-server-webapp-23.3.0.war/WEB-INF/classes/logback.xml** before building images.

3. Download tomcat-9.0.62.tar.gz and copy to $WORKSPACEDIR/unified-topology-builder/staging/downloads/tomcat.
4. Download **jdk-17.0.7_linux-x64_bin.tar.gz** and copy to $WORKSPACEDIR/unified-topology-builder/staging/downloads/java.
5. Export proxies in environment variables, fill the details on proxy settings:

   ```
   export ip_addr=`ip -f inet addr show eth0|egrep inet|awk '{print $2}'|awk -F/ '{print $1}'`
   export http_proxy=
   export https_proxy=$http_proxy
   export no_proxy=localhost,$ip_addr
   export HTTP_PROXY=
   export HTTPS_PROXY=$HTTP_PROXY
   export NO_PROXY=localhost,$ip_addr
   ```

6. Update $WORKSPACEDIR/unified-topology-builder/bin/gradle.properties with required proxies.

   ```
   systemProp.http.proxyHost=
   systemProp.http.proxyPort=
   systemProp.https.proxyHost=
   systemProp.https.proxyPort=
   systemProp.http.nonProxyHosts=localhost|127.0.0.1
   systemProp.https.nonProxyHosts=localhost|127.0.0.1
   ```

7. Uncomment the proxy block and provide $WORKSPACEDIR/unified-topology-builder/bin/m2/settings.xml with required proxies.

   ```
   <proxies>
       <proxy>
           <id>oracle-http-proxy</id>
           <host>xxxxx</host>
           <protocol>http</protocol>
           <nonProxyHosts>localhost|127.0.0.1|xxxxx</nonProxyHosts>
           <port>xxxxx</port>
           <active>true</active>
       </proxy>
   </proxies>
   ```

8. Copy UI custom icons to directory older $WORKSPACEDIR/unified-topology-builder/staging/downloads/unified-topology-ui/images if you have any customizations for service topology icon. For making customizations, see "Customizing the Images".
9. Update the image tag in $WORKSPACEDIR/unified-topology-builder/bin/unified_topology_manifest.yaml
10. Run `build-all-images` script to create unified topology service images:

```
$WORKSPACEDIR/unified-topology-builder/bin/build-all-images.sh
```

> **Note:**
>
> You can include the above procedure into your CI pipeline as long as the required components are already downloaded to the staging area.

Post-build Image Management

The Unified Topology image builder creates images with names and tags based on the settings in the manifest file. By default, this results in the following images:

- uim-7.5.1.2.0-unified-topology-base-1.0.0.2.0:latest
- uim-7.5.1.2.0-unified-topology-api-1.0.0.2.0:latest
- uim-7.5.1.2.0-unified-pgx-1.0.0.2.0:latest
- uim-7.5.1.2.0-unified-topology-ui-1.0.0.2.0:latest
- uim-7.5.1.2.0-unified-topology-dbinstaller-1.0.0.2.0:latest
- uim-7.5.1.2.0-unified-topology-consumer-1.0.0.2.0:latest

Customizing the Images

Service topology can be customized using a JSON configuration file. See **Customizing UTIA Service Topology Configurations from UIM** in *UIM System Administrator's Guide* for more information. As a part of customization, if custom icons are to be used to represent nodes in service topology, they must be placed in the **$WORKSPACEDIR/unified-topology-builder/staging/downloads/unified-topology-ui/images/** folder and **unified-topology-ui image** must be rebuilt.

Creating a Unified Topology Instance

This section describes how to create a Unified Topology service instance in your cloud native environment using the operational scripts and the configuration provided in the common cloud native toolkit.

Before you can create a Unified Topology instance, you must validate cloud native environment. See "Planning and Validating Your Cloud Environment" for details on prerequisites.

In this section, while creating a basic instance, the project name is considered as **sr** and instance name is considered as **quick**.

> **Note:**
>
> Project and Instance names cannot contain any special characters.

Installing Unified Topology Cloud Native Artifacts and Toolkit

Build container images for the following using the Unified Topology cloud native Image Builder:

- Unified Topology Core application
- Unified PGX application
- Unified Topology User Interface application
- Unified Topology database installer

See "Deployment Toolkits" to download the Common cloud native toolkit archive file. Set the variable for the installation directory by running the following command,
where $WORKSPACEDIR is the installation directory of the COMMON cloud native toolkit:

```
export COMMON_CNTK=$WORKSPACEDIR/common-cntk
```

Setting up Environment Variables

Unified Topology Service relies on access to certain environment variables to run seamlessly. Ensure the following variables are set in your environment:

- Path to your common cloud native toolkit
- Traefik namespace

To set the environment variables:

1. Set the COMMON_CNTK variable to the path of directory where common cloud native toolkit is extracted as follows:

   ```
   $ export COMMON_CNTK=$WORKSPACEDIR/common-cntk
   ```

2. Set the `TRAEFIK_NS` variable for Traefik namespace as follows:

   ```
   $ export TRAEFIK_NS=Treafik Namespace
   ```

3. Set the TRAEFIK_CHART_VERSION variable for Traefik helm chart version. Refer UIM Compatibility Matrix for appropriate version. The following is a sample for Traefik chart version 15.1.0.

   ```
   $ export TRAEFIK_CHART_VERSION=15.1.0
   ```

4. Set SPEC_PATH variable to the location where application and database yamls are copied. See "Assembling the Specifications" to copy specification files if not already copied.

   ```
   $ export SPEC_PATH=$WORKSPACEDIR/utia_spec_dir
   ```

Registering the Namespace

After you set the environment variables, register the namespace.

To register the namespace, run the following command:

```
$COMMON_CNTK/scripts/register-namespace.sh -p sr -t targets
# For example, $COMMON_CNTK/scripts/register-namespace.sh -p sr -t traefik
# Where the targets are separated by a comma without extra spaces
```

> **Note:**
>
> `traefik` is the name of the target for registration of the namespace `sr`. The script uses TRAEFIK_NS to find these targets. Do not provide the Traefik target if you are not using Traefik.
>
> For Generic Ingress Controller, you do not have to register the namespace. To select the ingress controller, provide the **ingressClassName** value under the **ingress.className** field in the **applications.yaml file**. For more information about ingressClassName, see https://kubernetes.io/docs/concepts/services-networking/ingress/

Creating Secrets

You must store sensitive data and credential information in the form of Kubernetes Secrets that the scripts and Helm charts in the toolkit consume. Managing secrets is out of the scope of the toolkit and must be implemented while adhering to your organization's corporate policies. Additionally, Unified Topology service does not establish password policies.

> **Note:**
>
> The passwords and other input data that you provide must adhere to the policies specified by the appropriate component.

As a prerequisite to use the toolkit for either installing the Unified Topology database or creating a Unified Topology instance, you must create secrets to access the following:

- UTIA Database
- UIM Instance Credentials
- Secret for UTIA API
- Secret for UTIA UI
- OAM Authentication server details
- Truststore secret for OAM server

The toolkit provides sample scripts to perform this. These scripts should be used for manual and faster creation of an instance. It does not support any automated process for creating instances. The scripts also illustrate both the naming of the secret and the layout of the data

within the secret that Unified Topology requires. You must create secrets before running the install-database.sh or create-applications.sh scripts.

Creating Secrets for Unified Topology Database Credentials

The database secret specifies the connectivity details and the credentials for connecting to the Unified Topology PDB (Unified Topology schema). This is consumed by the Unified Topology DB installer and Unified Topology runtime.

1. Run the following script to create the required secrets:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p sr -i quick -
f $SPEC_PATH/sr/quick/applications.yaml -a unified-topology create database
```

2. Enter the corresponding values as prompted:
   - TOPOLOGY DB Admin(sys) Username: Provide Topology Database admin username
   - TOPOLOGY DB Admin(sys) Password: Provide Topology Database admin password
   - TOPOLOGY Schema Username: Provide username for Unified Topology schema to be created
   - TOPOLOGY Schema Password: Provide Unified Topology schema password
   - TOPOLOGY DB Host: Provide Unified Topology Database Hostname
   - TOPOLOGY DB Port: Provide Unified Topology Database Port
   - TOPOLOGY DB Service Name: Provide Unified Topology Service Name
   - PGX Client Username: Provide username for PGX Client User to be created
   - PGX Client Password: Provide PGX Client Password
3. Verify that the following secret is created:

```
sr-quick-unified-topology-db-credentials
```

Creating Secrets for UIM Credentials

The UIM secret specifies the credentials for connecting to the UIM application. This is consumed by Unified Topology runtime.

1. Run the following scripts to create the UIM secret:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p sr -i quick -
f $SPEC_PATH/sr/quick/applications.yaml -a unified-topology create uim
```

2. Enter the credentials and the corresponding values as prompted. The credentials should be as shown in the following example:

```
Provide UIM credentials ...(Format should be http: //<host>:<port>)
UIM URL: Provide UIM Application URL, sample https://quick.sr.uim.org:30443
UIM Username: Provide UIM username
UIM Password: Provide UIM password
Is provided UIM a Cloud Native Environment ? (Select number from menu)
1) Yes
2) No
#? 1
Provide UIM Cluster Service name (Format <uim-project>-<uim-instance>-
```

```
cluster-uimcluster.<project>.svc.cluster.local)
UIM Cluster Service name: sr-quick-cluster-uimcluster.sr.svc.cluster.local
#Provide UIM Cluster Service name.
```

> **Note:**
>
> - If the OAM IDP provider is used for authentication, provide the UIM front-end hostname URL in the format: `https://<instance>.<project>.ohs.<oam-host-suffix>:<loadbalancerport>`.
> - Provide the default UIM URL, if the SAML protocol is configured for authentication with any IDP (such as IDCS, Keycloak, and so on.) For example: `https://<instance>.<project>.<hostSuffix>:<loadbalancerport>`.

3. Verify that the following secret is created:

```
sr-quick-unified-topology-uim-credentials
```

Creating Secrets for Authentication on Unified Topology API

The **appUsers** secret specifies authentication configuration for Unified Topology API.

1. Update $SPEC_PATH/$PROJECT/$INSTANCE/credentials/topology-user-credentials.yaml with authentication configuration:

```
security:
  enabled: true #set enabled flag to true to enable authentication
  providers:
  - oidc:
      identity-uri: #idp-identity-uri
      base-scopes: #idp base scopes along with openid scope if supported
      client-id: topologyClient #Provide name of the client-id created
      client-secret:  xxxx #Provide Client Secret
      token-endpoint-auth: CLIENT_SECRET_POST
      cookie-name: "OIDC_SESSION"
      cookie-same-site: "Lax"
      header-use: true
      audience: #idp-audience
      redirect: true
      redirect-uri: "/topology"
```

2. Run the following script to create the appUsers secret:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p sr -i quick -f
$SPEC_PATH/sr/quick/applications.yaml -a unified-topology create appUsers
```

3. Enter the values appropriately against prompts.
4. Provide App User Credentials for `sr-quick`.
5. Enter the app credentials file: $SPEC_PATH/$PROJECT/$INSTANCE/credentials/topology-user-credentials.yaml.

6. Verify that the following secret is created:

```
sr-quick-unified-topology-user-credentials
```

Creating Secrets for Authentication on Unified Topology UI

The appUIUsers secret specifies authentication configuration for Unified Topology UI application.

1. Update $SPEC_PATH/$PROJECT/$INSTANCE/credentials/topology-ui-user-credentials.yaml with authentication configuration.

> **Note:**
>
> Uncomment and set the value of the property session timeout same as the value of **tokenExpiry**, if **tokenExpiry** is set with different value than default while creating an identity domain during OAM setup. **tokenExpiry** is set while creating an identity domain during OAM setup. See "Deploying the Common Authentication Service" for more information.

topology-ui-user-credentials.yaml

```
security:
  enabled: true #set enabled flag to true to enable authentication
  providers:
  - oidc:
      identity-uri: #idp-identity-uri
      base-scopes: #idp base scopes along with openid scope if supported
      client-id: topologyClient #Provide name of the client-id created
      client-secret:  xxxx #Provide Client Secret
      token-endpoint-auth: CLIENT_SECRET_POST
      cookie-name: "OIDC_SESSION"
      cookie-same-site: "Lax"
      header-use: true
      audience: #idp-audience
      redirect: true
      redirect-uri: "/redirect/unified-topology-ui"
      logout-enabled: true
      #The following values are needed when logout is enabled for OIDC
      logout-uri: "/oidc/logout"
      post-logout-uri: apps/unified-topology-ui
      #provide server logout else it it going to userlogout and dispalying
error page of OAM
      logout-endpoint-uri: "https://<oam-instance>.<oam-project>.ohs.<oam-
host-suffix>:<port>/oam/server/logout" #Provide oidc logout, update
<loadbalancerport> value and ohshostname.
      cookie-encryption-password: "lpmaster"

#uncomment and set the value of the property sessiontimeout same as the
value of tokenExpiry, if tokenExpiry is set with different value than
default while creating an identity domain during OAM setup.
#sessiontimeout: 3600
```

2. Run the following script to create the appUIUsers secret:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p sr -i quick -
f $SPEC_PATH/sr/quick/applications.yaml -a unified-topology create
appUIUsers
```

3. Enter the Topology UI User Credentials for 'sr-quick'.
4. Enter the app credentials file: $SPEC_PATH/$PROJECT/$INSTANCE/credentials/topology-ui-user-credentials.yaml #Provide path to the **topology-ui-user-credentials.yaml**.
5. Verify that the following secret is created:

```
sr-quick-unified-topology-ui-user-credentials
```

Creating Secrets for Authentication Server Details

The OAuth secret specifies details of the authentication server. It is used by Unified Topology to connect to Message Bus Bootstrap service. See "Adding Common OAuth Secret and ConfigMap" for more information.

If authentication is enabled on UTIA, ensure that you create an **oauthConfig** secret with the appropriate OIDC details of your identity provider. To create an **oauthConfig** secret, see "Adding Common OAuth Secret and ConfigMap".

Creating Secrets for SSL enabled on traditional UIM truststore

The **inventorySSL** secret stores the truststore file of the SSL enabled on traditional UIM, it is required only if Authentication is not enabled on topology and to integrate topology with UIM traditional instance.

1. Create truststore file using UIM certificates and to enable SSL on UIM. See *UIM System Administrator's Guide* for more information.
2. Once you have the certificate of traditional UIM run following command to create truststore:

```
keytool -importcert -v -alias uimonprem -file ./cert.pem -keystore ./
uimtruststore.jks -storepass *******
```

3. After creating **uimtruststore.jks** run following command to create **inventorySSL** secret and pass the truststore created above:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p sr -i quick -
f $SPEC_PATH/sr/quick/applications.yaml -a unified-topology create
inventorySSL
```

   The system prompts for the trustsotre file location and passpharase for truststore. Provide appropriate values.

Installing Unified Topology Service Schema

To install the Unified Topology schema:

1. Update values under unified-topology-dbinstaller in $SPEC_PATH/sr/quick/database.yaml file with values required for unified topology schema creation.

> **Note:**
>
> - The YAML formatting is case-sensitive. Use a YAML editor to ensure that you do not make any syntax errors while editing. Follow the indentation guidelines for YAML.
> - Before changing the default values provided in the specification file, verify that they align with the values used during PDB creation. For example, the default tablespace name should match the value used when PDB is created.

2. Edit the database.yaml file and update the DB installer image to point to the location of your image as follows:

```
unified-topology-dbinstaller:
  dbinstaller:
    image:  DB_installer_image_in_your_repo
    tag: DB_installer image tag in your repo
```

3. If your environment requires a password to download the container images from your repository, create a Kubernetes secret with the Docker pull credentials. See "Kubernetes documentation" for details. Refer the secret name in the database.yaml. Provide image pull secret and image pull policy details.

```
unified-topology-dbinstaller:
  imagePullPolicy: Never
# The image pull access credentials for the "docker login" into Docker
repository, as a Kubernetes secret.
# Uncomment and set if required.
#  imagePullSecret: ""
```

4. Run the following script to start the Unified Topology DB installer, which instantiates a Kubernetes pod resource. The pod resource lives until the DB installation operation completes.

```
$COMMON_CNTK/scripts/install-database.sh -p sr -i quick -f $SPEC_PATH/sr/
quick/database.yaml -a unified-topology -c 1
```

5. You can run the script with -h to see the available options.
6. Check the console to see if the DB installer is installed successfully.
7. If the installation has failed, run the following command to review the error message in the log:

```
kubectl logs -n sr sr-quick-unified-topology-dbinstaller
```

8. Clear the failed pod by running the following command:

```
helm uninstall sr-quick-unified-topology-dbinstaller -n sr
```

9. Run the install-database script again to install the Unified Topology DB installer.

Configuring the applications.yaml File

The **applications.yaml** file is a Helm override values file to override default values of unified topology chart. Update values under chart unified-topology in $SPEC_PATH/<PROJECT>/<INSTANCE>/applications.yaml to override the default values.

The **applications.yaml** provides a section for values that are common for all microservices. Provide Values under that common section and it is reflected for all services.

> **Note:**
>
> There are common values specified in **applications.yaml** and **database.yaml** for the microservices. To override the common value, specify the value for the common value under chart name of microservice. If value under the chart is empty, then common value is considered.

To configure the project specification:

1. Edit the **applications.yaml** to provide the image in your repository (name and tag) by running the following command:

```
vi $SPEC_PATH/<PROJECT>/<INSTANCE>/applications.yaml

** edit the topologyAPiName, pgxName, uiName to reflect the Unified
Topology image names and location in your docker repository
** edit the topologyAPiTag, pgxTag, uiTag to reflect the Unified Topology
image names and location in your docker repository

unified-topology:
  image:
    topologyApiName: uim-7.6.0.0.0-unified-topology-api-1.1.0.0.0
    pgxName: uim-7.6.0.0.0-unified-pgx-1.1.0.0.0
    uiName: uim-7.6.0.0.0-unified-topology-ui-1.1.0.0.0
    topologyConsumerName: uim-7.6.0.0.0-unified-topology-consumer-1.1.0.0.0
    topologyApiTag: latest
    pgxTag: latest
    uiTag: latest
    topologyConsumerTag: latest
```

2. If your environment requires a password to download the container images from your repository, create a Kubernetes secret with the Docker pull credentials. See the "Kubernetes documentation" for details. See the secret name in the **applications.yaml** for more information.

```
# The image pull access credentials for the "docker login" into Docker
repository, as a Kubernetes secret.
# uncomment and set if required.

unified-topology:
#  imagePullSecret:
#    imagePullSecrets:
#      - name: regcred
```

3. Set Pull Policy for unified topology images in **applications.yaml**. Set pullPolicy to **Always** in case image is updated.

```
unified-topology:
  image:
    pullPolicy: Never
```

4. Update **loadbalancerport** in **applications.yaml**. If there is no external loadbalancer configured for the instance change the value of loadbalancerport to the **ingressController NodePort** . If SSL is enabled on Unified Topology, provide SSL NodePort and if SSL is disabled, provide non-SSL NodePort.
If you use Oracle Cloud Infrastructure LBaaS, or any other external load balancer, if TLS is enabled set `loadbalancerport` to 443 else set `loadbalancerport` to 80 and update the value for `loadbalancerhost` appropriately.

```
#provide loadbalancer port
loadbalancerport: 30305
```

5. To enable authentication, set the flag **authentication.enabled** to **true**. If OAM is used for authentication, provide loadbalancer ip-address and ohs-hostname under **hostAliases**. The hostAliases are added under the pods within the **/etc/hosts** file. If you use any other IDP and it is not under the public dns server, you can provide the **hostAliases**.

```
# The enabled flag is to enable or disable authentication
authentication:
  enabled: true

hostAliases:
- ip: <ip-address>
  hostnames:
  - <oam-instance>.<oam-project>.ohs.<hostSuffix> (ex quick.sr.ohs.uim.org)
```

6. If Authentication is not enabled on UTIA and want to integrate UTIA with traditonal SSL enabled UIM, you have to create **inventorySSL** secret and enable the **inventorySSL** flag in **applications.yaml** as shown below:

```
# make it true if using on prem inventory with ssl port enabled and
authentication is not enabled on topology
# always false for Cloud Native inventory
# not required in production environment
isInventorySSL: true
```

Configuring Unified Topology Application Properties

Sample configuration files topology-static-config.yaml.sample, topology-dynamic-config.yaml.sample are provided as follows:

- The sample files for Topology API service are added in **$COMMON_CNTK/charts/unified-topology-app/charts/unified-topology/config/topology-api**.
- The sample files for Topology Consumer service are added in **$COMMON_CNTK/charts/unified-topology-app/charts/unified-topology/config/topology-consumer**.

To override configuration properties, copy the sample static property file to **topology-static-config.yaml** and sample dynamic property file to **topology-dynamic-config.yaml**. Provide key value to override the default value provided out-of-the-box for any specific system

configuration property. The properties defined in property files are fed into the container using Kubernetes configuration maps. Any changes to these properties require the instance to be upgraded. Pods are restarted after configuration changes to **topology-static-config.yaml**.

Max Rows

Modify the following setting to limit the number of records returned in LIMIT queries:

```
topology:
    query:
      maxrows: 5000
```

Date Format

Any modifications to the date format used by all dates must be consistently applied to all consumers of the APIs.

```
topology:
    api:
      dateformat: yyyy-MM-dd'T'HH:mm:ss.SSS'Z'
```

Alarm Types

The out of the box alarm types utilize industry standard values. If you want to display a different value, modify the value accordingly:

For example: To modify the COMMUNICATIONS_ALARM change the value to COMMUNICATIONS_ALARM: Communications

```
alarm-types:
      COMMUNICATIONS_ALARM: COMMUNICATIONS_ALARM
      PROCESSING_ERROR_ALARM: PROCESSING_ERROR_ALARM
      ENVIRONMENTAL_ALARM: ENVIRONMENTAL_ALARM
      QUALITY_OF_SERVICE_ALARM: QUALITY_OF_SERVICE_ALARM
      EQUIPMENT_ALARM: EQUIPMENT_ALARM
      INTEGRITY_VIOLATION: INTEGRITY_VIOLATION
      OPERATIONAL_VIOLATION: OPERATIONAL_VIOLATION
      PHYSICAL_VIOLATION: PHYSICAL_VIOLATION
      SECURITY_SERVICE: SECURITY_SERVICE
      MECHANISM_VIOLATION: MECHANISM_VIOLATION
      TIME_DOMAIN_VIOLATION: TIME_DOMAIN_VIOLATION
```

Event Status

UTIA supports 3 types of events: 'Raised' for new events, 'Updated' for existing events with updated information and 'Cleared' for events that have been Closed.

To modify the 'CLEARED' event change the value to CLEARED: closed

```
    event-status:
      CLEARED: CLEARED
      RAISED: RAISED
      UPDATED: UPDATED
```

Event Severity

UTIA supports various types of event severity on a Device. The severity from most severe to least severe is CRITICAL(1), MAJOR(5), WARNING(10), INTERMEDIATE(15), MINOR(20), CLEARED(25) and None(999).

Internally, a numeric value is used to identify the severity hierarchy. The top three most severe events are tracked in UTIA.

To modify the 'INTERMEDIATE' severity change the value to INTERMEDIATE: moderate

```
severity:
      CLEARED: CLEARED
      INDETERMINATE: INDETERMINATE
      CRITICAL: CRITICAL
      MAJOR: MAJOR
      MINOR: MINOR
      WARNING: WARNING
```

Path Analysis Cost Values

UTIA supports 3 different types of numeric cost values for each edge/connectivity maintained in topology. The cost type label is configured based on your business requirements and data available.

You select the cost parameter to evaluate while using path analysis. The cost values are maintained externally using the REST APIs.

To modify 'costValue3' from Distance to Packet Loss change the value to costValue3: PacketLoss after updating the data values.

```
pathAnalysis:
  costType:
    costValue1: Jitter
    costValue2: Latency
    costValue3: Distance
```

Path Analysis Alarms

Alarms can be used by path analysis to exclude devices in the returned paths. The default setting is to exclude devices with any alarm.

To allow Minor and Greater alarms modify the setting to:

excludeAlarmTypes: Critical and Greater, Major and Greater

All Paths Limit

To improve the response time, modify the max number of paths returned when using 'All' Paths.

Topology Consumer

Reduce the Poll size for Retry and dlt Topic

Uncomment or add the configuration values in **topology-config.yaml** and upgrade the Topology Consumer service.

Maximum Poll Interval and Records

Edit **max.poll.interval.ms** to increase or decrease the delay between invocations of **poll()** when using consumer group management and **max.poll.records** to increase or decrease the maximum number of records returned in a single call to **poll()**.

```
mp.messaging:
      incoming:
        toInventoryChannel:
    #      max.poll.interval.ms: 300000
    #      max.poll.records: 500
        toFaultChannel:
    #      max.poll.interval.ms: 300000
    #      max.poll.records: 500
        toRetryChannel:
    #      max.poll.interval.ms: 300000
    #      max.poll.records: 200
        toDltChannel:
    #      max.poll.interval.ms: 300000
    #      max.poll.records: 100
```

Partition assignment strategy

The **PartitionAssignor** is the class that decides which partitions are assigned to which consumer. While creating a new Kafka consumer, you can configure the strategy that can be used to assign the partitions amongst the consumers. You can set it using the configuration **partition.assignment.strategy**. The partition re-balance (moving partition ownership from one consumer to another) happens, in case of:

- Addition of new Consumer to the Consumer group.
- Removal of Consumer from the Consumer group.
- Addition of New partition to the existing topic.

To change the partition assignment strategy, update the *topology-config.yaml* for topology consumer and redeploy the POD. The below example configuration shows the **CooperativeStickyAssignor** strategy. For list of supported partition assignment strategies, see **partition.assignment.strategy** in Apache Kafka documentation.

```
mp.messaging
  connector:
    helidon-kafka:
      partition.assignment.strategy:
org.apache.kafka.clients.consumer.CooperativeStickyAssignor
```

Integrate Unified Topology Service with Message Bus Service

To integrate Unified Topology API service with Message Bus service:

1. In the file **$SPEC_PATH/sr/quick/applications.yaml**, uncomment the section **messagingBusConfig**.
2. Provide namespace and instance name on which the Messaging Bus service is deployed.

3. Security protocol is SASL_PLAINTEXT if authentication is enabled on Message bus service. If authentication is not enabled on the Message Bus service, the security protocol is PLAINTEXT.

A sample configuration when authentication is enabled and Messaging Bus is deployed on instance 'quick' and namespace 'sr' is as follows:

applications.yaml

```
authentication:
  enabled: true

messagingBusConfig:
  namespace: sr
  instance: quick
```

Creating a Unified Topology Instance

To create a Unified Topology instance in your environment using the scripts that are provided with the toolkit:

1. Run the following command to create a UTIA instance:

   ```
   $COMMON_CNTK/scripts/create-applications.sh -p sr -i quick -
   f $SPEC_PATH/sr/quick/applications.yaml -a unified-topology
   ```

   The create-applications script uses the helm chart located in $COMMON_CNTK/charts/unified-topology-app to create and deploy a unified-topology service.
2. If the scripts fail, see the Troubleshooting Issues section at the end of this topic, before you make additional attempts.

Accessing Unified Topology

Proxy Settings

To set the proxy settings:

1. In the browser's network no-proxy settings include ***<hostSuffix>**. For example, `*uim.org`.
2. In `/etc/hosts` include **etc/hosts**

   ```
   <k8s cluster ip or loadbalancerIP>
   <instance>.<project>.topology.<hostSuffix>

   for example: <k8s cluster ip or external loadbalancer ip>
   quick.sr.topology.uim.org
   ```

Exercise Unified Topology service endpoints

If TLS is enabled on Unified Topology, exercise endpoints using Hostname <topology-instance>.<topology-project>.topology.uim.org.

Unified Topology UI endpoint format: https://<topology-instance>.<topology-project>.topology.<hostSuffix>:<port>/apps/unified-topology-ui

Unified Topology API endpoint format: https://<topology-instance>.<topology-project>.topology.<hostSuffix>:<port>/topology/v2/vertex

- Unified Topology UI endpoint: https://quick.sr.topology.uim.org:30443/apps/unified-topology-ui
- Unified Topology API endpoint: https://quick.sr.topology.uim.org:30443/topology/v2/vertex

If TLS is not enabled on Unified Topology, exercise endpoints:

Unified Topology UI endpoint format: http://<topology-instance>.<topology-project>.topology.<hostSuffix>:<port>/apps/unified-topology-ui

Unified Topology API endpoint format: http://<topology-instance>.<topology-project>.topology.<hostSuffix>:<port>/topology/v2/vertex

Validating the Unified Topology Instance

To validate the UTIA instance:

1. Run the following to check the status of unified-topology instance deployed.

   ```
   $COMMON_CNTK/scripts/application-status.sh -p sr -i quick -f $SPEC_PATH/sr/quick/applications.yaml -a unified-topology
   ```

   The application-status script returns the status of unified topology service deployments and pods status.

2. Run the following endpoint to monitor health of unified-topology:

   ```
   https://<loadbalancerhost>:<loadbalancerport>/unified-topology/health
   ```

3. Run the following Unified Topology service endpoints to add entry in /etc/hosts <k8s cluster ip or external loadbalancer ip> quick.sr.topology.uim.org:
   - Unified Topology UI endpoint: `https://quick.sr.topology.uim.org:30443/apps/unified-topology-ui`
   - Unified Topology API endpoint: `https://quick.sr.topology.uim.org:30443/topology/v2/vertex`

Deploying the Graph Server Instance

Graph Server or Pgx Server instance is needed for Path Analysis. By default, replicaCount of pgx(graph) server pods is set to '0'. For path analysis to function , set the replicaCount of pgx pods to '2' and upgrade instance. See "Upgrading the UTIA Instance" for more information.

A **cron** job must be scheduled to periodically reload the active unified-topology-pgx pod.

```
pgx:
  pgxName: "unified-pgx"
  replicaCount: 2
  java:
    user_mem_args: "-Xms8000m -Xmx8000m -XX:+HeapDumpOnOutOfMemoryError -
XX:HeapDumpPath=/logMount/$(APP_PREFIX)/unified-topology/unified-pgx/"
    gc_mem_args: "-XX:+UseG1GC"
    options:
  resources:
```

```
    limits:
      cpu: "4"
      memory: 16Gi
    requests:
      cpu: 3500m
      memory: 16Gi
```

Scheduling the Graph Server Restart CronJob

Once the instance is created succesfully, cronjob needs to schedule for unified-topology-pgx pod restarts. For a scheduled period of time, one of the unified-topology-pgx pod is restarted and all incoming requests are routed to other unfified-topology-pgx pod seamlessly.

Update the script $COMMON_CNTK/samples/cronjob-scripts/pgx-restart.sh to include required environment variables - KUBECONFIG, pgx_ns, pgx_instance. For a basic instance, pgx_ns is sr and pgx_instance is quick.

```
export KUBECONFIG=<kube config path>
export pgx_ns=<unified-topology project name>
export pgx_instance=<unified-topology instance name>
pgx_pods=`kubectl get pods -n $pgx_ns --sort-by=.status.startTime -o name |
awk -F "/" '{print $2}' | grep $pgx_instance-unified-pgx`
pgx_pod_arr=( $pgx_pods )
echo "Deleting pod - ${pgx_pod_arr[0]}"
kubectl delete pod ${pgx_pod_arr[0]} -n $pgx_ns --grace-period=0
```

The following crontab is scheduled for every day midnight. Scheduled time may vary depending on the volume of data.

Variable $COMMON_CNTK should be set in environment where cronjob runs or replace $COMMON_CNTK with complete path.

```
crontab -e 0 0 * * * $COMMON_CNTK/samples/cronjob-scripts/pgx-restart.sh
> $COMMON_CNTK/samples/cronjob-scripts/pgx-restart.log
```

Affinity on Graph Server

If multiple PGX pods are scheduled on the same worker node, the memory consumption by these PGX pods becomes very high. To address this, include the following affinity rule in **applications.yaml**, under the unified-topology chart to avoid scheduling of multiple PGX pods on the same worker node.

The following **podantiaffinity** rule uses the **app= <topology-project>-<topology-instance>-unified-pgx** label. Update the label with the corresponding project and instance names for UTIA service. For example: **sr-quick-unified-pgx**.

```
unified-topology:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: app
                operator: In
```

```
            values:
            - <topology-project>-<topology-instance>-unified-pgx
      topologyKey: "kubernetes.io/hostname"
```

Upgrading the Unified Topology Instance

Upgrading Unified Topology is required when there are updates made to **applications.yaml** and **topology-static-config.yaml** and **topology-dynamic-config.yaml** configuration files.

Run the following command to upgrade unified topology service.

```
$COMMON_CNTK/scripts/upgrade-applications.sh -p sr -i quick -f $COMMON_CNTK/
samples/applications.yaml -a unified-topology
```

After script execution is done, validate the unified topology service by running application-status script.

Restarting the Unified Topology Instance

To restart the Unified Topology instance:

1. Run the following command to restart unified topology service

   ```
   $COMMON_CNTK/scripts/restart-applications.sh -p sr -i quick -
   f $SPEC_PATH/sr/quick/applications.yaml -a unified-topology -r all
   ```

2. After running the script, validate the unified topology service by running application-status script.
3. To restart **unified-topology-api/unified-topology-ui/unified-pgx**, run the above command by passing **-r** with service name as follows:
4. To restart Unified Topology API

   ```
   $COMMON_CNTK/scripts/restart-applications.sh -p sr -i quick -
   f $SPEC_PATH/sr/quick/applications.yaml -a unified-topology -r unified-
   topology-api
   ```

5. To restart Unified Topology PGX

   ```
   $COMMON_CNTK/scripts/restart-applications.sh -p sr -i quick -
   f $SPEC_PATH/sr/quick/applications.yaml -a unified-topology -r unified-pgx
   ```

6. To restart Unified Topology UI:

   ```
   $COMMON_CNTK/scripts/restart-applications.sh -p sr -i quick -
   f $SPEC_PATH/sr/quick/applications.yaml -a unified-topology -r unified-
   topology-ui
   ```

7. To restart Unified Topology Consumer

   ```
   $COMMON_CNTK/scripts/restart-applications.sh -p sr -i quick -
   f $SPEC_PATH/sr/quick/applications.yaml -a unified-topology -r unified-
   topology-consumer
   ```

Alternate Configuration Options for UTIA

You can configure UTIA using the following alternate options.

Setting up Secure Communication using TLS

When Unified Topology service is involved in secure communication with other systems, either as the server or as the client, you should additionally configure SSL/TLS.The procedures for setting up TLS use self-signed certificates for demonstration purposes. However, replace the steps as necessary to use signed certificates.

To setup secure communication using TLS:

1. Generate keystore by passing `commoncert.pem` and `commonkey.pem` generated while OAM setup for inputs. Provide `-name "param"`.

```
openssl pkcs12 -export -in $COMMON_CNTK/certs/commoncert.pem -
inkey $COMMON_CNTK/certs/commonkey.pem -out $COMMON_CNTK/certs/
keyStore.p12 -name "topology"
```

2. Edit the $SPEC_PATH/sr/quick/applications.yaml and set tls enabled to true. Provide tls strategy to be used either terminate or reencrypt. Tls strategy should be RENCRYPT If authetication is enabled using OHS service enabled with SSL.

```
tls:
  # The enabled flag is to enable or disable the TLS support for the
unified topology m-s end points
  enabled: true
  # valid values are TERMINATE, REENCRYPT
  strategy: "REENCRYPT"
```

> **Note:**
>
> TLS terminate strategy requires ingressTLS secret and TLS reencrypt requires both ingressTLS and appkeystore secrets to be created.

3. Create IngressTLS secret to pass the generated certificate and key pem files.

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p sr -i quick -
f $SPEC_PATH/sr/quick/applications.yaml -a unified-topology create
ingressTLS
```

4. The script prompts for the following detail:
   a. Ingress TLS Certificate Path (PEM file): <path_to_cert.pem>
   b. Ingress TLS Key file Path (PEM file): <path_to_key.pem>

5. Create appkeystore secret to pass the generated keystore file.

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p sr -i quick -
f $SPEC_PATH/sr/quick/applications.yaml -a unified-topology create
appKeystore
```

6. The script prompts for the following detail:
   - App TLS Keystore Passphrase: <export password value passed while creating keyStrore.p12 key>
   - App TLS Keystore Key Alias: <-name "param" passed while creating keyStore.p12 key>
   - App TLS Keystore PrivateKey Path: <path to keyStore.p12>
7. Verify that the following secrets are created successfully.

```
sr-quick-unified-topology-ingress-tls-cert-secret
sr-quick-unified-topology-keystore
```

8. Create Unified Topology Instance as usual. Access Topology endpoints using hostname <topology-instance>.<topology-instance>.topology.uim.org
9. Add entry in /etc/hosts <k8s cluster ip or external loadbalancer ip> quick.sr.topology.uim.org
10. Unified Topology UI endpoint: `https://quick.sr.topology.uim.org:30443/apps/unified-topology-ui`
11. Unified Topology API endpoint: `https://quick.sr.topology.uim.org:30443/topology/v2/vertex`

> **Note:**
>
> UTIA supports wild card certificates. You can generate wildCard Certificates with the **hostSuffix** value provided in **applications.yaml**. The default is **uim.org**.
>
> You must change the **subDomainNameSeperator** value from period(**.**) to hyphen(**-**) so that the incoming hostnames match the wild card DNS pattern.
>
> Make the following updates to the **$SPEC_PATH/project/instance/applications.yaml file**.
>
> ```
> #Uncooment and provide the value of subDomainNameSeparator, default
> is "."
> #Value can be changed as "-" to match wild-card pattern of ssl
> certificates.
> #Example hostnames for "-" quick-sr-topology.uim.org
> subDomainNameSeparator: "-"
> ```

Setting up Secure Outgoing Communication using TLS

As part of the secret created under section **Creating Secrets for Authentication Server details**,, a truststore is created by adding OAM server certificate. This enables secure communication between OAM and UTIA applications.

Similarly, to enable secure outgoing communication between the server and UTIA, perform the steps mentioned in the section **Creating Secrets for Authentication Server details**.

1. Add server certificates to the truststore.
2. Recreate the secret using **Creating Secrets for Authentication Server details**.
3. Upgrade the UTIA instance to take the latest truststore from secret. To upgrade UTIA, see **Upgrade Unified Topology Instance** section.

For example: To enable SSL outgoing communication from UTIA to UIM on premise application, Add UIM certificates to the truststore and recreate the secret and upgrade UTIA application.

> **Note:**
>
> Follow the standard procedure for certificate creation. If UIM Inventory is accessed using IP address/Hostname of the machine, UIM certificate should contain IP address/Hostname of the machine as subject alternative name in the certificate. Sample command for certificate creation along with subject alternative names (Both the cloud native value and subject alternative names has hostname entry):
>
> ```
> openssl req -x509 -newkey rsa:2048 -days 365 -keyout key.pem -out
> cert.pem -nodes -subj "/CN=<hostname> /ST=TL /L=HYD /O=ORACLE /
> OU=CAGBU" -extensions san -config <(echo '[req]'; echo
> 'distinguished_name=req'; echo '[san]';echo
> 'subjectAltName=@alt_names';echo '[alt_names]';echo
> 'DNS.1=<hostname>';echo 'DNS.2=localhost';echo
> 'DNS.3=svc.cluster.local';)
> ```

Using Annoation-Based Generic Ingress Controller

UTIA supports standard Kubernetes ingress API and has samples for integration. In the following configuration, the required annotations for UTIA for nginx, are provided.

Any Ingress Controller, which conforms to the standard Kubernetes ingress API and supports annotations required by UTIA should work, although Oracle does not certify individual Ingress controllers to confirm this **generic** compatibility.

To use annotation-based generic ingress controller:

1. Update applications.yaml to provide the following annotations that enable stickiness through cookies:

```
# Valid values are TRAEFIK, GENERIC
ingressController: "GENERIC"

ingress:
  className: nginx  ##provide ingressClassName value, default value for
nginx ingressController is nginx.
  # This annotation is required for nginx ingress controller.
  annotations:
    nginx.ingress.kubernetes.io/affinity: "cookie"
    nginx.ingress.kubernetes.io/affinity-mode: "persistent"
```

```
    nginx.ingress.kubernetes.io/session-cookie-name: "nginxingresscookie"
    nginx.ingress.kubernetes.io/proxy-body-size: "50m"
```

2. To enable SSL REENCRYPT strategy for UTIA, add the applications-specific annotation under **unified-topology** tag in **applications.yaml** as follows:

```
unified-topology:
   #uncomment and provide applications specific annotations if required,
these will get added to list of annotations specified in common section.
   ingress:
     annotations:
     #following annotation is required if ssl reencrypt strategy is
enabled with nginx
     nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
```

Enabling Authentication for UTIA

This section provides you with information on enabling authentication for UTIA.

The samples, for using IDCS as Identity Provider, are packaged with UTIA. To use any Identity Provider of your choice, you must follow the corresponding configuration instructions.

Registering UTIA in Identity Provider

You can register UTIA as a Confidential application in Identity Provider. To do so:

1. Access the IDCS console and log in as administrator.
2. Navigate to the **Domains** and select the domain (*Default domain*) to add Helidon application as Confidential application.
3. Click **Add application** to register Helidon application as Confidential application.
   a. Choose **Confidential Application** and click **Launch workflow**.
   b. Enter the name as **Unified Topology Application** and description as **Unified Topology Application**.
   c. Select **Enforce grants as authorization** checkbox under **Authentication and authorization** section.
   d. Click **Next** at the bottom of the page.
   e. Choose **Configure this application as a resource server now** radio button under **Resource server configuration**.
   f. Enter **Primary Audience** as **https://<topology-hostname>:<loadbalancer-port>/**.
   g. Select **Add secondary audience** and enter IDCS URL as **Secondary audience**.
   h. Select **Add scopes** and add **utiaScope** as **allowed scope**.
   i. Select **Configure this application as a client now** radio button under the **Client configuration** section.
   j. Select **Resource owner**, **Client credentials**, and **Authorization code** check boxes.
   k. Select **Allow HTTP URLs** check box only if your UTIA application is not SSL enabled.
   l. Enter the following **Redirect URLs**:
      - https://<unified-topology-hostname>:<loadbalancer-port>/topology

- https://<unified-topology-hostname>:<loadbalancer-port>/redirect/unified-topology-ui/

m. Enter **Post-logout redirect URL** as **https://<topology-hostname>:<loadbalancer-port>/apps/unified-topology-ui** (provide your Helidon application's home page URL).

n. Enter **Logout URL** as **https://<topology-hostname>:<loadbalancer-port>/oidc/logout** (provide your Helidon application's logout URL).

o. (Optional) Select **Bypass consent** button for skipping the consent page after IDCS login.

p. Select **Anywhere** radio button for **Client IP address**.

q. Click **Next** and click **Finish**.

4. Click **Activate** to create application (Unified Topology Application).
5. Click **Activate application** from the pop-up window.
6. Click **Users** on the left side pane to assign users.
   a. Click **Assign users** to add domain users to the registered application.
   b. Choose the desired users from the pop-up window and click **Assign**.
7. (Optional) Click **Groups** on the left-side pane to assign groups.
   a. Click **Assign groups** to add domain groups to the registered application.
   b. Choose the desired groups from the pop-up window and click **Assign**.

Common Secret and Properties

You create a secret and config map with OAuth client details, which will be required for Message Bus and UTIA.

Getting Client Credentials

Access the IDCS console and log in as Administrator. To get client credentials:

1. Navigate to **Domains** and select the domain (*Default domain*) to add Helidon application as Confidential application.
2. Click on the **Unified Topology Application** name from the table.
3. Scroll to view the **Client secret** under the **General Information** section.
4. Click **Show secret** link to open a pop-up window showing the client secret.
5. Copy the link and store it to use in the Helidon application configuration.

Creating the OAuth Secrets and ConfigMap

To create **OauthConfig** secret with OIDC, see "Adding Common OAuth Secret and ConfigMap".

The sample for IDCS is as follows:

```
Client Id: e6e0b2cxxxxxxxxxxxxxx
Client Secret: xxxx-xxxx-xxxx-xxxx
Client Scope: https://quick.sr.topology.uim.org:30443/utiaScope
Client Audience: https://quick.sr.topology.uim.org:30443/
Token Endpoint Uri:https://idcs-
df3**********f64b21.identity.pint.oc9qadev.com:443/oauth2/v1/token
```

```
Valid Issue Uri: https://identity.oraclecloud.com/
Introspection Endpoint Uri: https://idcs-
df3**********f64b21.identity.pint.oc9qadev.com:443/oauth2/v1/introspect
JWKS Endpoint Uri: <oauth-jwks-endpoint-uri> : https://idcs-
df3**********f64b21.identity.pint.oc9qadev.com:443/admin/v1/SigningCert/jwk
Certificate File Path: ./identity-pint-oc9qadev-com.pem
Truststore File Path): ./truststore.jks (Note: trustore file should contain
IDP and UIM certificates)
Truststore Password: xxxxxx
```

Registering Identity Provider for UTIA

You register a OIDC Identity Provider for UTIA.

Creating Secrets for UTIA UI Authentication

The **appUIUsers** secret stores authentication configuration for UTIA UI.

To create secrets for UTIA UI authentication:

1. Update **$COMMON_CNTK/samples/credentials/topology-ui-user-credentials.yaml** with authentication configuration.

```
security:
  enabled: true
  config.require-encryption: false
  providers:
  - oidc:
      identity-uri: "<oauth-identity-uri>" (Ex. https://idcs-
df3*************f64b21.identity.pint.oc9qadev.com:443)
      client-id: "<oauth-client-id>" (Ex. e6e0b2cxxxxxxxxxxxxxxxx)
      client-secret: "<oauth-client-secret>" (Ex. xxxx-xxxx-xxxx-xxxx)
      redirect: true
      redirect-uri: "/redirect/unified-topology-ui"
      token-endpoint-auth: CLIENT_SECRET_POST
      audience: "<oauth-client-audience>" (Ex. "")
      base-scopes: "<oauth-client-scope>" (Ex. utiaScope )
      header-use: true
      cookie-name: "OIDC_SESSION"
      cookie-same-site: "LAX"
      logout-enabled: true
      logout-uri: "/oidc/logout"
      logout-endpoint-uri: "<oauth-logout-uri>"
      post-logout-uri: /apps/unified-topology-ui
```

2. Run the following script to create the **appUIUsers** secret:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p sr -i quick -
f $COMMON_CNTK/samples/applications.yaml -a unified-topology create
appUIUsers
```

3. Enter the corresponding values as prompted by the system.
4. Provide **Topology UI User Credentials** for **sr-quick**.
5. Enter the app credentials file **$COMMON_CNTK/samples/credentials/topology-ui-user-credentials.yaml** (provide path to the topology-ui-user-credentials.yaml).

6. Verify if the following secret is created:

```
sr-quick-unified-topology-ui-user-credentials
```

The following **topology-ui-user-credentials.yaml** is a sample file for IDCS:

```
security:
      enabled: true
      config.require-encryption: false
      providers:
      - oidc:
          identity-uri: "<oauth-identity-uri>" (Ex. <a target="_blank"
href="https://idcs-
df3*************f64b21.identity.pint.oc9qadev.com:443">https://idcs-
df3*************f64b21.identity.pint.oc9qadev.com:443</a>)
          client-id: "<oauth-client-id>" (Ex. e6e0b2cxxxxxxxxxxxxxxxx)
          client-secret: "<oauth-client-secret>" (Ex. xxxx-xxxx-xxxx-xxxx)
          redirect: true
          redirect-uri: "/redirect/unified-topology-ui"
          token-endpoint-auth: CLIENT_SECRET_POST
          audience: "<oauth-client-audience>" (Ex. "")
          base-scopes: "<oauth-client-scope>" (Ex. utiaScope )
          header-use: true
          cookie-name: "OIDC_SESSION"
          cookie-same-site: "LAX"
          logout-enabled: true
          logout-uri: "/oidc/logout"
          logout-endpoint-uri: "<a target="_blank" href="https://idcs-
df3*****f64b21.identity.oraclecloud.com/oauth2/v1/userlogout"">https://idcs-
df3*****f64b21.identity.oraclecloud.com/oauth2/v1/userlogout"</a>
          post-logout-uri: /apps/unified-topology-ui
```

Creating Secrets for Authentication on UTIA API

The **appUsers** secret stores the authentication configuration for UTIA API application.

To create secrets for authentication on UTIA API:

1. Update **$COMMON_CNTK/samples/credentials/topology-user-credentials.yaml** with authentication configuration as follows:

```
security:
  enabled: true
  config.require-encryption: false
  providers:
  - oidc:
      identity-uri: "<oauth-identity-uri>" (Ex. https://idcs-
df3*****************f64b21.identity.pint.oc9qadev.com:443)
      client-id: "<oauth-client-id>" (Ex. e6e0b2cxxxxxxxxxxxxxxxxxxxx)
      client-secret: "<oauth-client-secret>" (Ex. xxxx-xxxx-xxxx-xxxx)
      redirect: true
      audience: "<oauth-client-audience>" (Ex. "")
      base-scopes: "<oauth-client-scope>" (Ex. utiaScope )
      redirect-uri: "/topology"
      header-use: true
```

```
        cookie-name: "OIDC_SESSION"
        cookie-same-site: "Lax"
```

2. Run the following script to create the **appUIUsers** secret:

   ```
   $COMMON_CNTK/scripts/manage-app-credentials.sh -p sr -i quick -
   f $COMMON_CNTK/sr/quick/applications.yaml -a unified-topology create
   appUsers
   ```

3. Enter the corresponding values as prompted by the system.
4. Provide **App User Credentials** for **sr-quick**.
5. Enter the app credentials file **$COMMON_CNTK/samples/credentials/topology-user-credentials.yaml** (provide path to the topology-user-credentials.yaml).
6. Verify if the following secret is created:

   ```
   sr-quick-unified-topology-user-credentials
   ```

The following **topology-user-credentials.yaml** is a sample file for IDCS:

```
  security:
      enabled: true
      config.require-encryption: false
      providers:
      - oidc:
          identity-uri: "<oauth-identity-uri>" (Ex. https://idcs-
df3****************f64b21.identity.pint.oc9qadev.com:443)
          client-id: "<oauth-client-id>" (Ex. e6e0b2cxxxxxxxxxxxxxxxxxxxx)
          client-secret: "<oauth-client-secret>" (Ex. xxxx-xxxx-xxxx-xxxx)
          redirect: true
          audience: "<oauth-client-audience>" (Ex. "")
          base-scopes: "<oauth-client-scope>" (Ex. utiaScope )
          redirect-uri: "/topology"
          header-use: true
          cookie-name: "OIDC_SESSION"
          cookie-same-site: "Lax"
```

Choosing Worker Nodes for Unified Topology Service

By default, Unified Topology has its pods scheduled on all worker nodes in the Kubernetes cluster in which it is installed. However, in some situations, you may want to choose a subset of nodes where pods are scheduled.

For example:

Limitation on the deployment of Unified Topology on specific worker nodes per each team for reasons such as capacity management, chargeback, budgetary reasons, and so on.

To choose a subset of nodes where pods are scheduled, you can use the configuration in the **applications.yaml** file.

Sample node affinity configuration(requiredDuringSchedulingIgnoredDuringExecution) for unified topology service:

applications.yaml

```
unified-topology:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: name
            operator: In
            values:
            - south_zone
```

Kubernetes pod is scheduled on the node with label name as *south_zone*. If node with label name: *south_zone* is not available, pod will not be scheduled.

Sample node affinity configuration (preferredDuringSchedulingIgnoredDuringExecution:) for unified topology service:

applications.yaml

```
unified-topology:
  affinity:
    nodeAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: name
            operator: In
            values:
            - south_zone
```

Kubernetes pod is scheduled on the node with label name as *south_zone*. If node with label name: *south_zone* is not available, pod will still be scheduled on another node.

Setting up Persistent Storage

Follow the instructions mentioned in *UIM Cloud Native Deployment guide* for configuring Kubernetes persistent volumes.

To create persistent storage:

1. Update **applications.yaml** to enable storage volume for unified topology service and provide the persistent volume name.

   ```
   storageVolume:
     enabled: true
     pvc: sr-nfs-pvc #Specify the storage-volume name
   ```

2. Update database.yaml to enable storage volume for unified topology dbinstaller and provide the persistent volume name.

   ```
   storageVolume:
     enabled: true
   ```

```
type: pvc
pvc: sr-nfs-pvc #Specify the storage-volume name
```

After the instance is created, you must see the directories **unified-topology** and **unified-topology-dbinstaller** in your PV mount point, if you have enabled logs.

Managing Unified Topology Logs

To customize and enable logging, update the logging configuration files for the application.

1. Customize unified-topology-api service logs:
   - For service level logs update file $COMMON_CNTK/charts/unified-topology-app/charts/unified-topology/config/topology-api/logging-config.xml
   - For Helidon-specific logs update file $COMMON_CNTK/charts/unified-topology-app/charts/unified-topology/config/topology-api/logging.properties. By default console handler is used, you can provide filehandler as well uncomment below lines and provide <project> and <instance> names for location to save logs

     ```
     handlers=io.helidon.common.HelidonConsoleHandler,java.util.logging.FileHandler
     java.util.logging.FileHandler.formatter=java.util.logging.SimpleFormatter
     java.util.logging.FileHandler.pattern=/logMount/sr-quick/unified-topology/unified-topology-api/logs/TopologyJULMS-%g-%u.log
     ```

2. Customize unified-topology-pgx service logs:
   Update file $COMMON_CNTK/charts/unified-topology-app/charts/unified-topology/config/pgx/logging-config.xml
3. Customize unified-topology-ui service logs:
   Update file $COMMON_CNTK/charts/unified-topology-app/charts/unified-topology/config/topology-ui/logging.properties
4. Update the logging configuration files and upgrade the unified-topology m-s application:

   ```
   $COMMON_CNTK/scripts/upgrade-applications.sh -p sr -i quick -f $SPEC_PATH/applications.yaml -a unified-topology
   ```

Viewing Logs using Elastic Stack

You can view and analyze the Unified Topology service logs using Elastic Stack.

The logs are generated as follows:

- Fluentd collects the text logs that are generated during Unified Topology deployment and sends them to Elasticsearch.
- Elasticsearch collects all types of logs and converts them into a common format so that Kibana can read and display the data.
- Kibana reads the data and presents it in a simplified view.

See "Setting Up Elastic Stack" for more information.

Setting Up Elastic Stack

To set up Elastic Stack:

1. Install Elasticsearch and Kibana using the following commands:

```
#Install elasticsearch and kibana . It might take time to download images
from docker hub.
kubectl apply -f $COMMON_CNTK/samples/charts/elasticsearch-and-kibana/
elasticsearch_and_kibana.yaml

#Check if services are running, append namespace if deployment is other
than default like:- kubectl get services --all-namespaces
kubectl get services

Access kibana dashboard

Method 1 - kubectl get svc ( will return all the services  , append
namespace if deployment is other than default like:- kubectl get services
--all-namespaces)

Ex- elasticsearch    ClusterIP   10.96.190.99    <none>        9200/
TCP,9300/TCP   113d
    kibana           NodePort    10.100.198.88   <none>
5601:31794/TCP      113d

Kibana service nodeport at port 31794 is created

Now access kibana dashboard using url - http://<IP address of
VM>:<nodeport>/
```

2. Run the following command to create a namespace ensuring that it does not already exist.

```
kubectl get namespaces
export FLUENTD_NS=fluentd
kubectl create namespace $FLUENTD_NS
```

3. Update $COMMON_CNTK/samples/charts/fluentd/values.yaml with Elastic Search Host and Port.

```
elasticSearch:
  host: "elasticSearchHost"
  port: "elasticSearchPort"
```

   For example:

```
elasticSearch:
  host: "elasticsearch.default.svc.cluster.local"
  port: "9200"
```

4. Modify the Fluentd image resources if required.

```
image: fluent/fluentd-kubernetes-daemonset:v1-debian-elasticsearch
   resources:
     limits:
     memory: 200Mi
   requests:
```

```
        cpu: 100m
        memory: 200Mi
```

5. Run the following commands to install fluentd-logging using the **$COMMON_CNTK/samples/charts/fluentd/values.yaml** file in the samples:

```
helm install fluentd-logging $COMMON_CNTK/samples/charts/fluentd -
n $FLUENTD_NS --values $COMMON_CNTK/samples/charts/fluentd/values.yaml \
--set namespace=$FLUENTD_NS \
--atomic --timeout 800s
```

6. Run the following command to upgrade fluentd-logging:

```
helm upgrade fluentd-logging $COMMON_CNTK/samples/charts/fluentd -
n $FLUENTD_NS --values $COMMON_CNTK/samples/charts/fluentd/values.yaml \
   --set namespace=$FLUENTD_NS \
   --atomic --timeout 800s
```

7. Run the following command to uninstall fluentd-logging:

```
helm delete fluentd-logging -n $FLUENTD_NS
```

8. Use 'fluentd_logging-YYYY.MM.DD' (default index configuration) index pattern in Kibana to check the logs.

Visualize logs in Kibana

To visualize logs in Kibana:

1. Navigate to Kibana dashboard (http://<IP address of VM>:<nodeport>/).
2. Create Index pattern (fluentd_looging-YYYY.MM.DD).
3. Click on Discover.

Viewing Logs using OpenSearch

You can view and analyze the Application logs using OpenSearch.

The logs are generated as follows:

1. Fluentd collects the application logs that are generated during cloud native deployments and sends them to OpenSearch.
2. OpenSearch collects all types of logs and converts them into a common format so that OpenSearch Dashboard can read and display the data.
3. OpenSearch Dashboard reads the data and presents it in a simplified view.

See "Setting Up OpenSearch" for more information.

Managing Unified Topology Metrics

Run the following endpoint to monitor metrics of unified topology:

```
https://<loadbalancerhost>:<loadbalancerport>/sr/quick/unified-topology/
metrics
```

Prometheus and Grafana setup

See "Setting Up Prometheus and Grafana" for more information.

Adding scrape Job in Prometheus

Add the following Scrape job in Prometheus Server. This can be added by editing the **config** map used by the Prometheus server:

```
- job_name: 'topologyApiSecuredMetrics'
      oauth2:
        client_id: <client-id>
        client_secret: <client-secret>
        scopes:
          - <Scope>
        token_url: <OAUTH-TOKEN-URL>
        tls_config:
          insecure_skip_verify: true
      kubernetes_sd_configs:
      - role: pod
      relabel_configs:
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
        action: keep
        regex: true
      - source_labels: [__meta_kubernetes_pod_label_app]
        action: keep
        regex: (<project>-<instance>-unified-topology-api)
      - source_labels: [__meta_kubernetes_pod_container_port_number]
        action: keep
        regex: (8080)
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
        action: replace
        target_label: __metrics_path__
        regex: (.+)
      - source_labels: [__address__,
__meta_kubernetes_pod_annotation_prometheus_io_port]
        action: replace
        regex: ([^:]+)(?::\d+)?;(\d+)
        replacement: $1:$2
        target_label: __address__
      - action: labelmap
        regex: __meta_kubernetes_pod_label_(.+)
      - source_labels: [__meta_kubernetes_namespace]
        action: replace
        target_label: kubernetes_namespace
      - source_labels: [__meta_kubernetes_pod_name]
        action: replace
        target_label: kubernetes_pod_name
```

> **Note:**
>
> If Authentication is not enabled on Unified Topology, remove **oauth** section from above mentioned job.

Allocating Resources for Unified Topology Service Pods

To increase performance of the service, **applications.yaml** has configuration to provide JVM memory settings and pod resources for Unified Topology Service.

There are separate configurations provided for topology-api, topology-consumer, pgx and topology-ui services. Provide required values under the service name under unified-topology application.

```
unified-topology:
  topologyApi:
    apiName: "unified-topology-api"
    replicaCount: 3
    java:
      user_mem_args: "-Xms2000m -Xmx2000m -XX:+HeapDumpOnOutOfMemoryError -
XX:HeapDumpPath=/logMount/$(APP_PREFIX)/unified-topology/unified-topology-
api/"
      gc_mem_args: "-XX:+UseG1GC"
      options:
    resources:
      limits:
        cpu: "2"
        memory: 3Gi
      requests:
        cpu: 2000m
        memory: 3Gi
```

Scaling Up or Scaling Down the Unified Topology Service

Provide replica count in **applications.yaml** to scale up or scale down the unified topology pods. Replica count can be configured for topology-api, topology-consumer, pgx and topology-ui pods individually by updating **applications.yaml**.

Update **applications.yaml** to increase replica count to 3 for topology-api deployment.

```
unified-topology:
  topologyApi:
    replicaCount: 3
```

Apply the change in replica count to the running Helm release by running the upgrade-applications script.

```
$COMMON_CNTK/scripts/upgrade-applications.sh -p sr -i quick -f $SPEC_PATH/sr/
quick/applications.yaml -a unified-topology
```

Enabling GC Logs for UTIA

By default, GC logs are disabled, you can enable them and view the logs at the corresponding folders inside location **/logMount/sr-quick/unified-topology**.

To Enable GC logs, update **$SPEC_PATH/sr/quick/applications.yaml** file as follows:

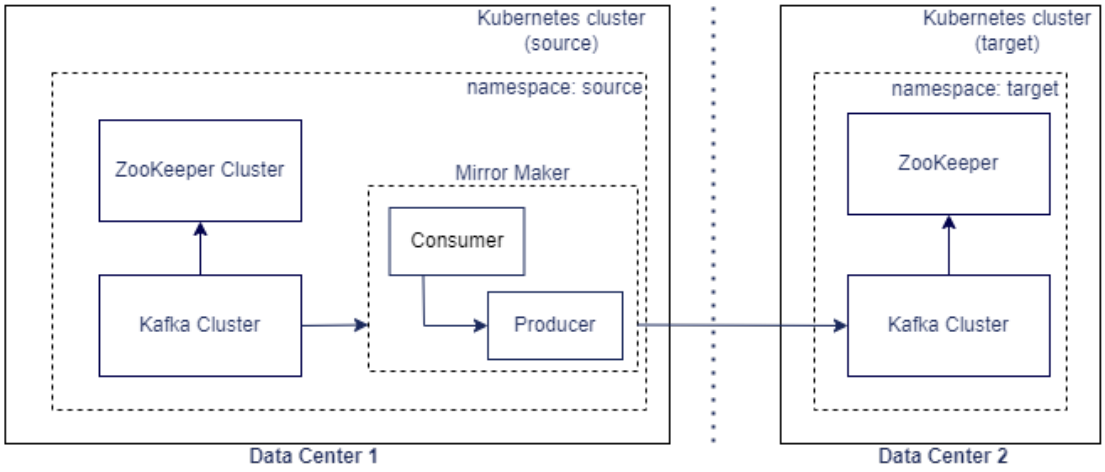
1. Under `gcLogs` make `enabled` as `true` you can uncomment `gcLogs` options under unified-topology to override the common values.
2. To configure the maximum size of each file and limit for number of files you need to set `fileSize` and `noOfFiles` inside `gcLogs` as follows:

```
gcLogs:
  enabled: true
  fileSize: 10M
  noOfFiles: 10
```

Geo Redundancy Support

The Geo Redundancy of Message Bus (which uses Kafka) is achieved with Mirror Maker tool. Apache Kafka Mirror Maker replicates data across two Kafka clusters, within or across data centers. See https://strimzi.io/blog/2020/03/30/introducing-mirrormaker2/ for more details.

The following diagram shows an example of how mirror maker replicates the topics from source Kafka cluster to target Kafka cluster.



The prerequisites are as follows:

- The Strimzi operator should be up and running
- The source Message Bus service should be up and running
- The target Message Bus service should be up and running

Strimzi Operator

Validate that the Strimzi operator is installed by running the following command:

```
$kubectl get pod -n <STRIMZI_NAMESPACE>

NAME                                        READY   STATUS    RESTARTS   AGE
strimzi-cluster-operator-566948f58c-sfj7c   1/1          Running
0          6m55s
```

Validate installed helm release for Strimzi operator by running the following command:

```
$helm list -n <STRIMZI_NAMESPACE>

NAME                    NAMESPACE          REVISION        STATUS
CHART                         APP VERSION
strimzi-operator        STRIMZI_NAMESPACE  1               deployed
strimzi-kafka-operator-0.X.0   0.X.0
```

Source Message Bus

The source Message Bus should be up and running (the Kafka cluster from which the topics should be replicated).

Validate the Kafka cluster is installed by running the following command:

```
$kubectl get pod -n sr1

 NAME                                                  READY   STATUS
RESTARTS   AGE
 sr1-quick1-messaging-entity-operator-5f9c688c7-2jcjg   3/3     Running
0          27h
 sr1-quick1-messaging-kafka-0                           1/1     Running
0          27h
 sr1-quick1-messaging-zookeeper-0                       1/1     Running
0          27h
```

Validate the persistent volume claims created for the Kafka cluster by running the following command:

```
$kubectl get pvc -n sr1

NAME                                    STATUS
VOLUME                                    CAPACITY   ACCESS MODES
STORAGECLASS   AGE
data-sr1-quick1-messaging-kafka-0       Bound    <volume>  1Gi
RWO           sc             27h
data-sr1-quick1-messaging-zookeeper-0   Bound    <volume>  1Gi
RWO           sc
```

Target Message Bus

The target Message Bus should be up and running (the Kafka cluster to which the topics should be replicated).

Validate the Kafka cluster is installed by running the following command:

```
$kubectl get pod -n sr2

 NAME                                                  READY   STATUS
RESTARTS   AGE
 sr2-quick2-messaging-entity-operator-5f9c688c7-2jcjg   3/3     Running
0          27h
 sr2-quick2-messaging-kafka-0                           1/1     Running
0          27h
```

```
 sr2-quick2-messaging-zookeeper-0                      1/1     Running
0          27h
```

Validate the persistent volume claims created for the Kafka cluster by running the following command:

```
$kubectl get pvc -n <kafka target namespace>`

NAME                                      STATUS
VOLUME                                      CAPACITY   ACCESS MODES
STORAGECLASS   AGE
data-sr2-quick2-messaging-kafka-0       Bound    <volume>  1Gi
RWO           sc              27h
data-sr2-quick2-messaging-zookeeper-0   Bound    <volume>  1Gi
RWO           sc              27h
```

Installing and configuring Mirror Maker 2.0

A sample is provided at **$COMMON_CNTK/samples/messaging/kafka-mirror-maker**.

Disaster Recovery Support

A minimum of two pods is required for a service to be highly available. They should be on different worker nodes (Kubernetes can schedule the pods on different nodes using pod anti-affinity). If one node goes down, it takes out the corresponding pod, leaving the other pod(s) to handle the requests until the downed pod can be rescheduled. When a worker node goes down, the PODs running on that worker node will be rescheduled on other available worker nodes.

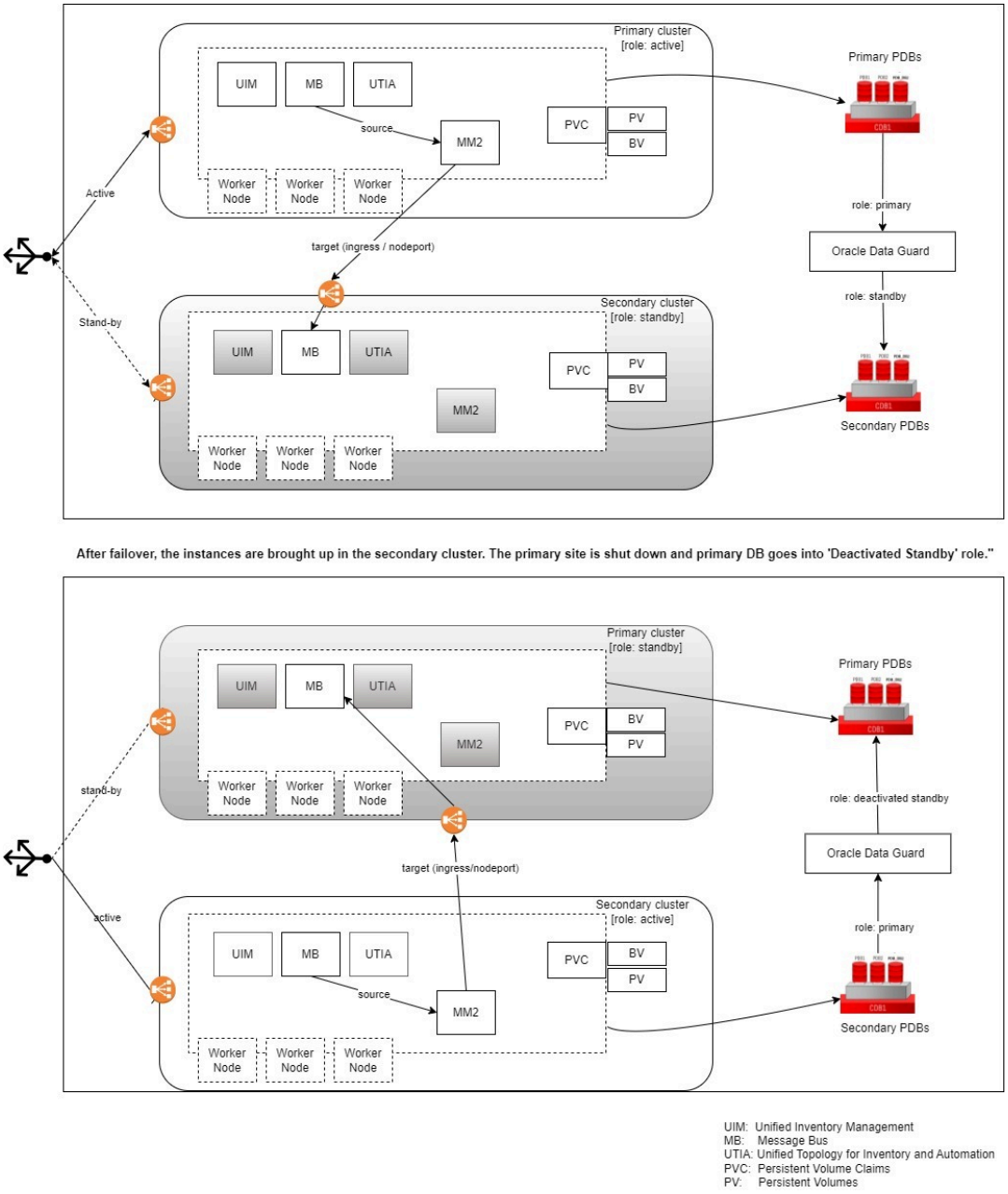
For DB High Availability we can use the Oracle Real Application Clusters (RAC) to run a single Oracle Database across multiple servers in order to maximize availability and enable horizontal scalability.

Disaster Recovery across Data Centers

The disaster recovery when the data center completely goes down is maintained with another passive data center.

Figure 5-2 documents the disaster recovery plan for the data center. A parallel passive data center is maintained, where the runtime data is periodically replicated from the active data center to the passive data center. In the event of any catastrophic failures in the primary (or active) data center, the load must be switched to secondary (or passive) data center. Before switching the load to secondary data center, you should shutdown all the services in the primary data center and start all the services in the secondary data center.

Figure 5-2 Disaster Recovery Plan for Data Center



About Switchover and Failover

The purpose of a geographically redundant deployment is to provide resiliency in the event of a complete loss of service in the primary site, due to a natural disaster or other unrecoverable failure in the primary UIM site. This resiliency is achieved by creating one or more passive standby sites that can take the load when the primary site becomes unavailable. The role reversal from the standby site to the primary site can be accomplished in any of the following ways:

- **Switchover**, in which the operator performs a controlled shutdown of the primary site before activating the standby site. This is primarily intended for planned service interruptions in the primary UIM site. Following a switchover, the former primary site

becomes the standby site. The site roles of primary site and standby site can be restored by performing a second switchover operation, which is switchback.

- **Failover**, in which the primary site becomes unavailable due to unanticipated reasons and cannot be recovered. The operator then transitions the standby site to the primary role. The primary site that is down cannot act as a standby site and will require reconstruction of the database as a standby database before restoring the site roles.

About Kafka Mirror Maker

Kafka's Mirror Maker functionality makes it possible to maintain a replica of an existing Kafka cluster (which is used in Message Bus service). This mirrors a source Kafka cluster into a target (mirror) Kafka cluster. To use this mirror, it is a requirement that the source and target Kafka clusters (that is, Message Bus service) are up and running. If the target Kafka cluster is down or offline, we cannot mirror into the target cluster.

Oracle Data Guard

Oracle Data Guard is responsible for replicating transactions from the Active DB to the Standby DB. It is included as a part of every Oracle DB Enterprise Edition installation.

> **Note:**
>
> When using multi-tenant databases involving CDBs and PDBs with Data Guard, the replication happens at the CDB level. This means all the PDBs from the active CDB will be replicated over to the standby CDB and also, the commands to enable Data Guard must be run at the CDB level.

Installation and Configuration

If UTIA is disabled in UIM Cloud Native then it is not required to deploy Message Bus, UTIA and Mirror Maker Services in the clusters. These commands are intended to be used as samples. For detailed documentation on deploying UIM, see *UIM Cloud Native Deployment Guide*.

Setting up the Primary (active) Instance

To set up the primary (active) instance:

1. Provision Databases one for the primary site and another for the secondary site.
2. Set up Data Guard between primary site and secondary site. Primary site should be in **ACTIVE** role. Secondary site should be in **STANDBY** role. Refer to Oracle 19c Documentation.
3. Deploy UIM Cloud Native.
   a. Create image pull secrets (if required).
   b. Create UIM secrets for WLS admin, OPSS, WLS RTE, RCU DB and UIM DB.

> **Note:**
>
> `uimprimary` here refers to the Kubernetes namespace where the primary instance will be deployed. Replace this with the desired namespace.

```
$UIM_CNTK/scripts/manage-instance-credentials.sh -p uimprimary -i dr
create wlsadmin,opssWP,wlsRTE,rcudb,uimdb
```

c. Create Weblogic encrypted password.

```
$UIM_CNTK/scripts/install-uimdb.sh -p uimprimary -i dr -s $SPEC_PATH -c
8
```

d. Create UIM users secrets.

```
$UIM_CNTK/samples/credentials/manage-uim-credentials.sh -p uimprimary -
i dr -c create -f "/home/spec_dir/users.txt"
```

e. Create DB schemas.

```
$UIM_CNTK/scripts/install-uimdb.sh -p uimprimary -i dr -s $SPEC_PATH -c
1
$UIM_CNTK/scripts/install-uimdb.sh -p uimprimary -i dr -s $SPEC_PATH -c
2
```

f. Create UIM instance.

```
$UIM_CNTK/scripts/create-ingress.sh -p uimprimary -i dr -s $SPEC_PATH
$UIM_CNTK/scripts/create-instance.sh -p uimprimary -i dr -s $SPEC_PATH
```

g. Add UIM user roles.

```
$UIM_CNTK/samples/credentials/assign-role.sh -p uimprimary -i dr -f uim-
users-roles.txt
```

4. Deploy Message Bus.

```
$COMMON_CNTK/scripts/create-applications.sh -p uimprimary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml -a messaging-bus
```

5. Deploy UTIA:

a. Create Topology DB secrets:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p uimprimary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml -a unified-topology
create database
```

b. Create Topology UIM secrets:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p uimprimary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml -a unified-topology
create uim
```

c. Create Topology users:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p uimprimary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml -a unified-topology
create appUsers
```

d. Create Topology UI users:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p uimprimary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml -a unified-topology
create appUIUsers
```

e. Create DB schemas:

```
$COMMON_CNTK/scripts/install-database.sh -p uimprimary -i dr -
f $SPEC_PATH/<proejct>/<instance>/database.yaml -a unified-topology -c 1
```

f. Deploy Topology:

```
$COMMON_CNTK/scripts/create-applications.sh -p uimprimary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml -a unified-topology
```

See "Deploying Unified Operations Message Bus" for deploying Message Bus, "Deploying the Unified Topology for Inventory and Automation Service" for deploying UTIA.

See *UIM Cloud Native Deployment Guide* for deploying UIM.

Setting up the Secondary (standby) Instance

To set up the secondary (standby) instance:

1. Perform switchover operation on active (primary site) DB. Now secondary site DB should be in **ACTIVE** role and primary site DB should be in **PASSIVE** role. Refer to Oracle 19c Documentation.
2. Deploy UIM Cloud Native:
   a. Export OPSS wallet file secret from primary instance and recreate in secondary instance.

> **Note:**
>
> Where, `uimsecondary` refers to the Kubernetes namespace where the secondary instance will be deployed. Replace this with the desired namespace.

```
kubectl -n uimprimary get configmap uimprimary-dr-weblogic-domain-
introspect-cm -o jsonpath='{.data.ewallet\.p12}' > ./primary_ewallet.p12
```

```
$UIM_CNTK/scripts/manage-instance-credentials.sh -p uimsecondary -i dr
create opssWF
```

b. (Optional) Create image pull secrets.

c. Create UIM secrets for WLS admin, OPSS, WLS RTE, RCU DB and UIM DB:

```
$UIM_CNTK/scripts/manage-instance-credentials.sh -p uimsecondary -i
quick create wlsadmin,opssWP,wlsRTE,rcudb,uimdb
```

d. Create Weblogic encrypted password:

```
$UIM_CNTK/scripts/install-uimdb.sh -p uimsecondary -i dr -s $SPEC_PATH -
c 8
```

e. Create UIM users secrets:

```
$UIM_CNTK/samples/credentials/manage-uim-credentials.sh -p uimsecondary
-i dr -c create -f "/home/spec_dir/users.txt"
```

f. Create UIM instance:

```
$UIM_CNTK/scripts/create-ingress.sh -p uimsecondary -i dr -s $SPEC_PATH
$UIM_CNTK/scripts/create-instance.sh -p uimsecondary -i dr -s $SPEC_PATH
```

g. Add UIM user roles:

```
$UIM_CNTK/samples/credentials/assign-role.sh -p uimsecondary -i dr -f
uim-users-roles.txt
```

3. Deploy message bus:

```
$COMMON_CNTK/scripts/create-applications.sh -p uimsecondary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml  -a messaging-bus
```

4. Deploy UTIA:

a. Create Topology DB secrets:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p uimsecondary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml  -a unified-
topology create database
```

b. Create Topology UIM secrets:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p uimsecondary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml  -a unified-
topology create uim
```

c. Create Topology users:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p uimsecondary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml  -a unified-
topology create appUsers
```

d. Create Topology UI users:

```
$COMMON_CNTK/scripts/manage-app-credentials.sh -p uimsecondary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml  -a unified-
topology create appUIUsers
```

e. Deploy Topology:

```
$COMMON_CNTK/scripts/create-applications.sh -p uimsecondary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml  -a unified-topology
```

5. Deploy Mirror Maker. See "Installing and Configuring Mirror Maker 2.0" for more information.
6. After the secondary instance has been setup, switchover back to the primary (active) site.

Switchover Sequence

To perform a switchover between site A (active) and site B (standby):

1. Bring down instances in site A. These include UIM and UTIA. Message Bus must be enabled to perform the replication using Mirror Maker.

```
#Disable topology
$COMMON_CNTK/scripts/delete-applications.sh -p uimprimary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml -a unified-topology
#Disable UIM
$UIM_CNTK/scripts/delete-instance.sh -p uimprimary -i dr -s $SPEC_PATH
```

2. Perform switchover on DB. Site B DB will now become Primary. Site B DB will assume Standby role. Refer to Oracle 19c Documentation.
3. Bring up instances in site B. This includes UIM and UTIA. Message Bus should already be active:

```
#EnableUIM
$UIM_CNTK/scripts/create-instance.sh -p uimsecondary -i dr -s $SPEC_PATH
#Enable topology
$COMMON_CNTK/scripts/create-applications.sh -p uimsecondary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml -a unified-topology
```

4. Perform DNS switching to route all traffic to site B.

Failover Sequence

In case of any irrecoverable failure in the primary site, perform a failover operation on the standby site. To do so:

1. Perform failover on DB. Standby (secondary) DB will now become Primary. Primary site DB will assume Deactivated Standby role. Refer to Oracle 19c Documentation.
2. Bring up instances in standby. This includes UIM and Topology. Message Bus should already be active:

```
#EnableUIM
$UIM_CNTK/scripts/create-instance.sh -p uimsecondary -i dr -s $SPEC_PATH
#Enable topology
```

```
$COMMON_CNTK/scripts/create-applications.sh -p uimsecondary -i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml -a unified-topology
```

3. Perform DNS switching to route all traffic to secondary instances.

Once the primary site to restored, establish a synchronization between secondary and primary site. To do so:

1. Bring up Message Bus and DB in primary site:

```
#Enable message bus
$COMMON_CNTK/scripts/create-applications.sh -p uimprimary-i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml -a messaging-bus
```

2. Setup Kafka Mirror Maker with secondary Message Bus as source and primary Message Bus as target. See "About Kafka Mirror Maker" for more information.
3. Switch primary DB role from **Deactivated Standby → Standby**. See "Deploying Unified Operations Message Bus" for more information.

As the synchronization between secondary and primary site is established, perform a switchover to the primary site. To do so:

1. Bring up UIM in primary site:

```
$UIM_CNTK/scripts/create-instance.sh -p uimprimary -i dr -s $SPEC_PATH
```

2. Bring up Topology in primary site:

```
$COMMON_CNTK/scripts/create-applications.sh -p uimprimary-i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml -a unified-topology
```

3. Perform DNS switching to route all traffic to primary instances.
4. Bring down instances in secondary site. This includes UIM and Topology. Message Bus should remain active for Kafka Mirror Maker synchronization:

```
#Disable topology
$COMMON_CNTK/scripts/delete-applications.sh -p uimsecondary-i dr -
f $SPEC_PATH/<project>/<instance>/applications.yaml -a unified-topology
#Disable UIM
$UIM_CNTK/scripts/delete-instance.sh -p uimsecondary -i dr -s $SPEC_PATH
```

Debugging and Troubleshooting

Common Problems and Solutions

- Unified Topology DBInstaller pod is not able to pull the dbinstaller image.

```
NAME                                              READY   STATUS
RESTARTS   AGE
project-instance-unifed-topology-dbinstaller   0/1     ErrImagePull
0          5s

### OR

NAME                                              READY   STATUS
```

```
RESTARTS   AGE
project-instance-unifed-topology-dbinstaller   0/1     ImagePullBackOff
0          45s
```

To resolve this issue

1. Verify that the image name and the tag provided in database.yaml for unified-topology-dbinstaller and that it is accessible from the repository by the pod.
2. Verify that the image is copied to all worker nodes.
3. If pulling image from a repository, verify the image pull policy and image pull secret in database.yaml for unified-topology-dbinstaller.

- Unified Topology API, PGX and UI pod is not able to pull the images.

  To resolve this issue

  1. Verify that the image names and the tags are provided in **applications.yaml** for unified-topology and that it is accessible from the repository by the pod.
  2. Verify that the image is copied to all worker nodes
  3. If pulling image from a repository, verify the image pull policy and image pull secret in **applications.yaml** for UTIA service.
- Unified Topology pods are in crashloopbackoff state.

  To resolve this issue, describe the Kubernetes pod and find the cause for the issue. It could be because of missing secrets.
- Unified Topology API pod did not come up.

```
NAME                                          READY   STATUS
RESTARTS   AGE
project-instance-unifed-topology-api   0/1     Running   0          5s
```

  To resolve this issue, verify that the Message Bus bootstrap server provided in topology-static-config.yaml is a valid one.

Test Connection to PGX server

To troubleshoot PGX service, connect to pgx service using graph client by running the following command.

Connect to pgx service endpoint http://<LoadbalancerIP>:<LoadbalancerPort>/<topology-project>/<topology-instance>/pgx by providing pgx client user credentials.

```
C:\TopologyService\oracle-graph-client-22.1.0\oracle-graph-
client-22.1.0\bin>opg4j -b http://<hostIP>:30305/sr/quick/pgx -u
<PGX_CLIENT_USER>

password:<PGX_CLIENT_PASSWORD>
For an introduction type: /help intro
Oracle Graph Server Shell 22.1.0
Variables instance, session, and analyst ready to use.
```

Fallout Events Resolution

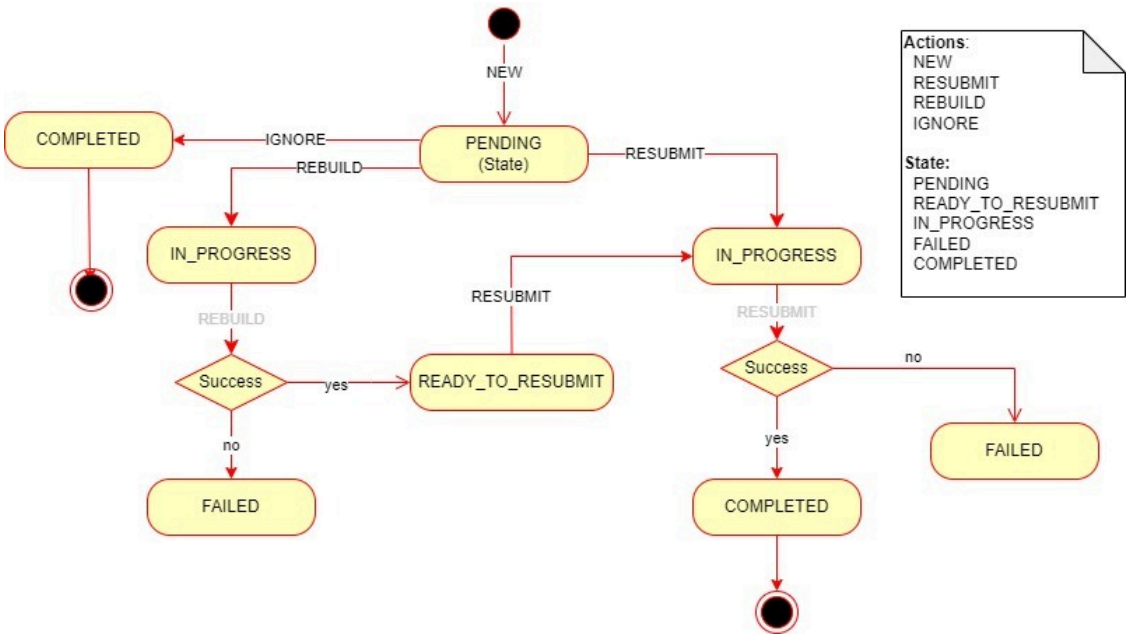
The **TOPOLOGY_FALLOUT_EVENTS** table in the UTIA schema, persists the failed events from the **Dead-Letter-Topic** (that is: ora-dlt-topology) for further analysis and re-processing. The data between UIM and UTIA can go out of sync when UIM application fails to send topology events to message-bus and UIM transaction is committed. It can also happen when topology is disabled in UIM temporarily and re-enabled, or when the UTIA is consuming events at a much slower rate than that of the rate at which UIM is producing events. These lead to UTIA data being out of sync with that of the UIM, hence resulting in failed events eventually.

These failed events in the TOPOLOGY_FALLOUT_EVENTS table can be rebuilt and resubmitted. When a fallout event comes into the table it's in "PENDING" state. These events can be Rebuilt or Resubmitted as follows:

- REBUILD: This action processes the Fallout Event and gets any out of sync data from UIM into UTIA via the Database Link.
- RESUBMIT: This action takes the events from the TOPOLOGY_FALLOUT_EVENTS table in "PENDING" or "READY_TO_RESUBMIT" states and moves them back into the "ora_uim_topology" topic to be re-processed.

The following figure illustrates the fallout events resolution process flow.

Figure 5-3 Process Flow of Fallout Events Resolution



Prerequisites for REBUILD

- Before Rebuild is performed, the UTIA Schema user should have the following privileges:
  - CREATE JOB
  - ALTER SYSTEM
  - CREATE DATABASE LINK

- Ensure a Database Link exists from UTIA schema to UIM schema with the name “REM_SCHEMA” (that is, UTIA schema user should be able to access objects from UIM schema). For more information, see https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-DATABASE-LINK.html#GUID-D966642A-B19E-449D-9968-1121AF06D793

Performing REBUILD Action

You can perform the Rebuild action in the following ways:

- **DBMS Job Scheduling:** In this approach the REBUILD action on the Fallout Events in “PENDING” state is scheduled to run for every 6 hours. The frequency at which the job runs automatically can be configured by changing the **repeat_interval**.

```
BEGIN
  DBMS_SCHEDULER.create_job (
    job_name        => 'FALLOUT_DATA_REBUILD',
    job_type        => 'PLSQL_BLOCK',
    job_action      => 'BEGIN
PKG_FALLOUT_CORRECTION.SCHEDULE_FALLOUT_JOBS(commitSize => 1000, cpusJobs
=> 4, waitTime => 2); END;',
    start_date      => SYSTIMESTAMP,
    repeat_interval => 'FREQ=HOURLY; INTERVAL=6',
    enabled         => TRUE
  );
END;
/
```

- **On-Demand REST API Call:** In this approach the REBUILD action on the Fallout Events in “PENDING” state is invoked via the REST API. Before invoking the Rebuild API.
  - POST - fallout/events/rebuild – To rebuild the Fallout Events on demand as and whenever required.
  - DELETE - fallout/events/scheduledJobs – To drop any running or previously scheduled jobs.

Performing RESUBMIT Action

Resubmit Action is performed through a REST call and it takes the fallout events in “READY_TO_RESUBMIT” (post Rebuild) and “PENDING” states based on the query parameters and pushed the events into the “ora_uim_topology” topic:

`POST` - fallout/events/resubmit – To resubmit the Fallout Events on demand.

For more information on APIs available, see *UTIA REST API Guide*.

Deleting and Recreating a Unified Topology Instance

- Run the following command to delete the Unified Topology service:

```
$COMMON_CNTK/scripts/delete-applications.sh -p sr -i quick -f $COMMON_CNTK/
samples/applications.yaml -a unified-topology
```

- Run the following command to delete the Unified Topology schema:

```
$COMMON_CNTK/scripts/install-database.sh -p sr -i quick -f $COMMON_CNTK/
samples/database.yaml -a unified-topology -c 2
```

- Run the following command to create the Unified Topology schema:

```
$COMMON_CNTK/scripts/install-database.sh -p sr -i quick -f $COMMON_CNTK/
samples/database.yaml -a unified-topology -c 1
```

- Run the following command to create the Unified Topology service:

```
$COMMON_CNTK/scripts/create-applications.sh -p sr -i quick -f $COMMON_CNTK/
samples/applications.yaml -a unified-topology
```

UTIA Support for Offline Maps

UTIA support for map visualization is provided by the third-party service providers such as Open Street Maps (OSM), MapBox, Carto, Esri, and Web Map Service (WMS).

UTIA integrates with these service providers and they provide the required components and computing resources, so that you can avoid setting up and maintaining a local tile server.

Oracle offers the following options to support offline maps:

- Allowlisting map URLs
- Setting up a local tile server

Allowlisting Map URLs

In highly secured installations, you may not provide internet access to the location. In such situations, Oracle recommends using an allowlist solution so the base maps can include the streets, cities, buildings, and so on.

For the map tiles to render, allowlist the following URLs:

- Tile 1:
  - http://a.tile.openstreetmap.org/11/472/824.png
  - http://b.tile.openstreetmap.org/11/472/825.png
  - http://c.tile.openstreetmap.org/11/472/825.png
- Tile 2:
  - http://a.tiles.mapbox.com/v4/mapbox.satellite/10/236/412@2x.png?access_token=pk.eyJ1IjoidzhyIiwiYSI6ImNpeGhwaXF1ejAwMHQydG8yZ3pyanZ5aTkifQ.QNScWNGnLRHIHXeAsGMvyw
  - http://b.tiles.mapbox.com/v4/mapbox.satellite/10/235/411@2x.png?access_token=pk.eyJ1IjoidzhyIiwiYSI6ImNpeGhwaXF1ejAwMHQydG8yZ3pyanZ5aTkifQ.QNScWNGnLRHIHXeAsGMvyw
  - http://c.tiles.mapbox.com/v4/mapbox.satellite/10/236/411@2x.png?access_token=pk.eyJ1IjoidzhyIiwiYSI6ImNpeGhwaXF1ejAwMHQydG8yZ3pyanZ5aTkifQ.QNScWNGnLRHIHXeAsGMvyw
- Tile 3:

  - http://a.basemaps.cartocdn.com/light_all/10/235/412@2x.png
  - http://b.basemaps.cartocdn.com/light_all/10/235/412@2x.png
  - http://c.basemaps.cartocdn.com/light_all/10/235/412@2x.png
- Tile 4:
  - http://a.basemaps.cartocdn.com/dark_all/10/236/413@2x.png
  - http://b.basemaps.cartocdn.com/dark_all/10/236/413@2x.png
  - http://c.basemaps.cartocdn.com/dark_all/10/236/413@2x.png
- Tile 5: http://server.arcgisonline.com/ArcGIS/rest/services/World_Street_Map/MapServer/tile/10/412/235
- Tile 6: http://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile/10/412/236

Setting Up a Local Tile Server

To set up a local tile server, you must deploy the prepared tile files on a local server. Every file must have its direct link: **http://tileserver.com/{z}/{y}/{x}**. This format allows getting the required response for the request **http://tileserver.com/{z}/{x}/{y}.png**.

Contact your system administrator to install, deploy, and run your own tile server. The configuration process is dependent on the tile server you choose to implement. The tile server requires high computing power and requires operations support and maintenance.

The tile server is responsible for caching the tiles, sharing the load, and processing the request queue at regular intervals.

You can consider some options available in the market such as MapTiler, QGIS, Switch2OSM, ArcGis Enterprise, and so on.

After you set up the tile server and a successful deployment, you can access the map tiles through APIs in the format: **http://{hostname}:{port}/{baseUrl}/{z}/{x}/{y}.png**.

Manual Changes for Setting Up a Local Tile Server

The following manual changes are required to set up a local tile server:

- Update **visualization-start-page.js** as per your requirement.
- Open **unified-topology-ui.jar** and navigate to **unified-topology-ui/flows/visualization/pages/visualization-start-page.js**.
- In the **loadGeoMaps** method of **visualization-start-page.js**, update the **mapurl** variable of the custom map API URL.

After you redeploy the updated jar file and run the application, you can see the map tiles served from your local server.

6

Data Migration and Dynamic Attribute Mapping

This chapter describes how to perform the Data Migration and Dynamic Attribute Mapping.

Planning the Topology Migration

In preparation for implementing UTIA, you must set up the topology migration and the UIM to topology configuration. The UIM to topology migration extracts and loads necessary information from UIM into the topology graph model consisting of vertices and edges. Following the Database per Service Micro service Design Pattern, the topology graph resides in a Pluggable Database (PDB) container separated from the UIM database.

The migration consists of the following:

- Index Rebuilding: The index rebuilding consists of re-creating indexes on tables with migrated data, dropping the temporary tables created during migration and renaming the tables with migrated data to actual topology tables.

> **Note:**
>
> If the UIM Entities are in 'UNAVAILABLE' state prior to migration, such entities will not be migrated.

Data Migration Approaches

You can follow the following approaches for data migration:

- Data Migration through Database Link: Database Link (DBLink) is created from UTIA schema to UIM schema.
- Data Migration through Read Access on UIM schema: UTIA schema is set up within the same PDB as that of the UIM schema. UTIA schema user is granted with **SELECT** (read access) along with the tables owned by UIM schema user. Data dump files are created for the migrated topology data. These dump files are then imported in the target PDB where the UTIA schema will be placed.

The prerequisites are:

- Add DATAFILE to increase the TABLESPACE available (SYSTEM by default) for the UTIA schema user. Preferably one-fourth the size of UIM schema.
- Data Migration to custom tablespace can be achieved by making the custom tablespace as the default tablespace for the UTIA schema user.

The Migration Steps are as follows:

1. Build Characteristics tables for the following topology enabled entities such as Equipment, Logical Device, Network, Network Edge, Physical Device, Pipe and Place. These **<ENTITY>_CHAR_MIG** tables are used to store all characteristics on each entity which are used during Dynamic Attribute Migration and Customizing Topology JSON files. Build <ENTITY>_CHAR_MIG tables:

- Open a command line window and login to SQL*Plus for the UIM database.
- Run the following SQL scripts providing the full path of the files. For example, use the @scriptFileName command where scriptFileName is the full path and name of the file.
  - $WORKSPACEDIR/unified-topology-builder/migration_scripts/Char_Mig_tables/CREATE_CHAR_MIG_TABLE.sql
  - $WORKSPACEDIR/unified-topology-builder/migration_scripts/Char_Mig_tables/MIGRATION_CHAR1.sql
  - $WORKSPACEDIR/unified-topology-builder/migration_scripts/Char_Mig_tables/MIGRATION_CHAR2.sql
  - $WORKSPACEDIR/unified-topology-builder/migration_scripts/Char_Mig_tables/MIGRATION_CHAR3.sql
- To verify if the scripts ran successfully, you can verify that the UIM schema includes the following tables:
  - EQUIPMENT_CHAR_MIG
  - LOGICALDEVICE_CHAR_MIG
  - NETWORK_CHAR_MIG
  - NETWORKEDGE_CHAR_MIG
  - PHYSICALDEVICE_CHAR_MIG
  - PIPE_CHAR_MIG
  - PLACE_CHAR_MIG
  - CHARACTERISTICS_TABLE_MAPPING_MIG

> **Note:**
>
> You can perform this step for any of the data migration approaches.

2. The Topology schema user account must have the following privileges:
   - CREATE JOB
   - CREATE SESSION
   - ALTER SYSTEM
   - CREATE DATABASE LINK
   - CREATE PROCEDURE
   - CREATE SEQUENCE
   - CREATE TABLE
   - CREATE TYPE
   - UNLIMITED TABLESPACE
   - CREATE JOB

   These above privileges are sufficient for Approach 1, however for Approach 2:
   - Create **SYNONYM**.

- Grant **SELECT** permission to all the tables owned by UIM schema user and UTIA schema user.

```
CREATE PROCEDURE grant_select(
    username VARCHAR2,
    grantee VARCHAR2)
AS
BEGIN
    FOR r IN (
        SELECT owner, table_name
        FROM all_tables
        WHERE owner = username
    )
    LOOP
        EXECUTE IMMEDIATE
            'GRANT SELECT ON '||r.owner||'.'||r.table_name||' to ' ||
grantee;
    END LOOP;
END;
    "username" - UIM Schema User
    "grantee" - UTIA Schema User within the same PDB.
```

3. Static Attribute Migration:
   - Open a command line window and login to SQL*Plus for the Topology database.
   - **Approach 1**:
     - Migrate the static attributes data by running $WORKSPACEDIR/unified-topology-builder/migration_scripts/data_migration_script_using_dblink.sql
     - The following input arguments are expected:
       * UIM schema username
       * UIM schema password
       * Database Hostname
       * Database port number
       * Database Service name
       * Commit Size(Optional – 50000(Default))
       * Maximum number of parallel processes(Optional – 5(Default))
       * Wait Time(Optional – 2(Default in seconds))
   - **Approach 2**:
     - Migrate the static attributes data by running $WORKSPACEDIR/unified-topology-builder/migration_scripts/data_migration_script_using_localCopy.sql
     - The expects the following input arguments:
       * UIM schema username with in the PDB
       * Commit Size(Optional – 50000(Default))
       * Maximum number of parallel processes(Optional – 5(Default))
       * Wait Time(Optional – 2(Default in seconds))

> **Note:**
>
> Commit Size: The number of records handled by a single process, Maximum number of parallel processes – Depends on number of CPU's available, Wait Time – Waiting interval after which the listener checks for the availability of jobs.

4. Modify the topology JSON files in **$WORKSPACEDIR/unified-topology-builder/migration_scripts/scriptGenerator/scriptGenerator_Execuable/topologyjsonfiles/** and run the following commands:
   Approach 1: **java -jar scriptgenerator_dblink-1.0-jar-with-dependencies.jar**

   Approach 2: **java -jar scriptgenerator_localCopy-1.0-jar-with-dependencies.jar**
5. Dynamic Attribute Migration: Once the **scriptgenerator_<Approach>-1.0-jar-with-dependencies.jar** is run, the SQLs required for Dynamic attribute migration are generated in $WORKSPACEDIR/unified-topology-builder/migration_scripts/scriptGenerator/scriptGenerator_Executable/scriptOutFiles/dynamicAtt.sql. Run the SQL queries sequentially.
6. Verify the migrated data by going through tables with `%_FINAL` or `%_NEW` name.
7. Index Rebuild: The tables with names as `%_FINAL` and `%_NEW` contain the actual migrated data and indexes and constraints have to be added to these tables, these are generated in $WORKSPACEDIR/unified-topology- builder/migration_scripts / scriptGenerator/scriptGenerator_Executable/ scriptOutFiles/indexRebuild.sql. Run the SQL queries sequentially.
8. In case of performing data migration using Approach 2, export the migrated Topology Data and import the migrated Topology Data into the target PDB where the UTIA schema is expected to be.
9. Oracle Optimizer determines the cost of each execution plan based on database, schema, table and other statistics. The changes inside database result in stale statistics. To gather new statistics, run the following command:

   ```
   EXEC DBMS_STATS.gather_schema_stats( '<TopologySchema_Name>' );
   ```

> **Note:**
>
> PG_PROFILE tables which store the Service Topology Data are not supported in existing migration. If you want service topology profile data in the topology schema you can create a new service configuration and approve it. In 7.5.1.0.0, Profile Data is created for every service configuration in Approved State.

Customizing Topology JSON files for Migration

The $WORKSPACEDIR/unified-topology-builder/migration_scripts/scriptGenerator/scriptGenerator_Execuable/topologyjsonfiles/ contains three topology JSON files:

- topologyAttributeMapping.json
- topologyRoleMapping.json
- topologySpecificationMapping.json

Customize topologyAttributeMapping.json

```
[
    {
        "name": "LogicalDeviceDAO",
        "properties": [
            {
                "name": "NativeEMSName",
                "property": "NativeEMSName",
                "vertex": "",
                "columnName": ""
            }
        ]
    }
]
```

TopologyAttributeMapping (TAM) is an array defining how attributes of different DAO's can map to Topology Schema. Each TAM object consists of key-value pairs of **name** and **properties**.

- **name** – Maps to different entity classes and entity specification classes. For example: "LogicalDeviceDAO", "EquipmentSpecificationDAO", "PlaceSpecificationDAO", "PropertyLocationDAO" and so on.
- **properties** – This is an array defining how individual attributes of an entity are supposed to be stored in Topology schema. Each JSON object of the **properties** has:
  - name – Name of the Attribute.
  - property – Name of the key used to store the value retrieved from Attribute.
  - vertex – Build the relationship with the Vertices, from Topology Schema.
  - columnName – Column from Topology Schema used to store the Attribute values.

> **Note:**
>
> In "properties" array objects, "name" is a mandatory field to be provided which maps to either "property" or "vertex" or "columnName".

An example of TAM is:

Assume, the topologyAttributeMapping.json contains the following:

```
[
    {
        "name": "LogicalDeviceSpecificationDAO",
        "properties": [
            {
                "name": "vendorName",
                "property": "",
                "vertex": "vendor",
                "columnName": ""
            },
            {
                "name": "modelnumber",
                "property": "Model",
```

```
                "vertex": "",
                "columnName": ""
            }
        ]
    },
    {
        "name": "EquipmentDAO",
        "properties": [
            {
                "name": " NativeEMSName",
                "property": "",
                "vertex": "",
                "columnName": "DEVICEIDENTIFIER"
            }
        ]
    }
]
```

In the above example:

- LogicalDeviceSpecification table from UIM schema is expected to have “vendorName” and “modelnumber” columns which are used to do the following:
  - All LogicalDeviceSpecification’s which have a vendorName as some non-null value is moved to PG_VENDOR table and containment edges between the devices of LogicalDevice type and their respective vendors are created in PG_DEVICE_TO_VENDOR table.
    Example: Assume there are 2 Logical Devices (“LDSampleDevice1” and “LDSampleDevice2”) of specification “LDSampleSpec”, and “LDSampleVendor” is the “vendorName”. Then, vertex/record for “LDSampleVendor” is created in PG_VENDOR table and the logical devices have their respective containment edges to the “LDSampleVendor” in PG_DEVICE_TO_VENDOR table.
  - All LogicalDeviceSpecification’s which have a “modelnumber” as some non-null value is stored in “PROPERTIES” column of PG_DEVICE table. For example: “LDSampleSpec” has “APTS-123” as “modelnumber”, then it’s stored as:

    ```
    {
        "Model": "APTS-123"
    }
    ```

  - Equipments which have non-null value in “NativeEMSName” are stored in “DEVICEIDENTIFIER” column of PG_DEVICE table.

Customizing “topologyRoleMapping.json”

```
[
    {
        "name": "ADM",
        "entityClass": [
            "LogicalDeviceDAO",
            "PhysicalDeviceDAO",
            "EquipmentDAO"
        ],
        "property": "",
        "vertex": "domain",
        "columnName": ""
```

```
    }
]
```

TopologyRoleMapping (TRM) is an array defining how entities which are role-enabled are stored in Topology schema. Each TRM object contains key-values pairs of “name”, “entityClass”, “property”, “vertex” and “columnName”.

- name – Name of the Role.
- entityClass – Entities which are enabled by the role and want data migrated for.
- property – Name of the key used to store the Role.
- vertex – Build the relationship with the Vertices, from Topology Schema
- columnName – Column from Topology Schema used to store the Role.

> **Note:**
>
> In each TRM object “name” is a mandatory field with role information which can be mapped to either “property” or “vertex” or “columnName”. If “entityClass” is empty ([]) that is same as role information to be checked in Logical Device, Equipment, Physical Device, Place, Pipe and Network.

An example of TRM is:

Assume, the topologyRoleMapping.json contains the following:

```
[
    {
        "name": "ADM",
        "entityClass": [
            "LogicalDeviceDAO",
            "PhysicalDeviceDAO",
            "EquipmentDAO"
        ],
        "property": "",
        "vertex": "domain",
        "columnName": ""
    },
    {
        "name": "EIGRP",
        "entityClass": [
            "LogicalDeviceDAO"
        ],
        "property": "routingProtocol",
        "vertex": "",
        "columnName": ""
    },
    {
        "name": "Router",
        "entityClass": [
            "EquipmentDAO"
        ],
        "property": "",
```

```
        "vertex": "",
        "columnName": "nodeCategory"
    }
]
```

In the above example,

- A record for **ADM** is created in PG_DOMAIN table and all logical devices, equipments, and physical devices that are enabled by the **ADM** role, have the corresponding records in the PG_DEVICE_TO_DOMAIN table.
- All logical devices enabled by the **EIGRP** role have the **PROPERTIES** column populated with

```
{
    "routingProtocol": "EIGRP"
}
```

- All equipments enabled by the **Router** role have **Router** stored in the **NODECATEGORY** column of PG_DEVICE table.

Customizing “topologySpecificationMapping.json”

```
[
    {
        "name": "EthernetDevice",
        "entityType": "LogicalDeviceSpecificationDAO",
        "relatedVertices": [
            {
                "vertex": "domain",
                "value": "Ethernet"
            }
        ],
        "characteristics": [
            {
                "name": "zoneID",
                "property": "",
                "vertex": "",
                "columnName": "ZONEID"
            }
        ]
    }
]
```

TopologySpecificationMapping (TSM) is an array defining how characteristics of a specification are mapped Topology schema and how all entities of a specification can have containment edge to other entities. Each TSM object contains key-values pairs of “name”, “entityType”, “relatedVertices” and “characteristics”.

- name – Name of the Specification.
- entityType – The type of entity does the specification represent.
- relatedVertices – Create containment edges for all entities of the given specification with the vertex and value. This contains an array of objects which have:
  - vertex – To which vertex the containment edges must be created to.

  - Value – The value of the vertex.
- characteristics – Array of characteristics provided by the specification and how they are stored in Topology schema.
  - name – Name of the characteristic(case-sensitive)
  - property- Name of the key used to store the characteristic.
  - vertex – Build the relationship with vertices in Topology schema.
  - columnName – Column from Topology schema in which the characteristic is stored.

> **Note:**
>
> In each TSM object “name” and “entityType” are mandatory fields with specification and type of specification information. “relatedVertices” is used to create direct containment edges for all entities of the specification in question. “characteristics” is an array of objects where “name” is mandatory and talks about the characteristics provided by specification and can be mapped to either “property” or “vertex” or “columnName”.

An example of TSM is:

Assume, the topologySpecificationMapping.json contains the following:

```
[
    {
        "name": "cableModem",
        "entityType": "PhysicalDeviceSpecificationDAO",
        "characteristics": [
            {
                "name": "deviceType",
                "property": "deviceType",
                "vertex": "",
                "columnName": ""
            }
        ]
    },
    {
        "name": "EthernetDevice",
        "entityType": "LogicalDeviceSpecificationDAO",
        "relatedVertices": [
            {
                "vertex": "domain",
                "value": "Ethernet"
            }
        ],
        "characteristics": [
            {
                "name": "Tech",
                "property": "",
                "vertex": "Technology",
                "columnName": ""
            }
        ]
```

```
    },
    {
        "name": "Generic_Address",
        "entityType": "PlaceSpecificationDAO",
        "characteristics": [
            {
                "name": "CityName",
                "property": "",
                "vertex": "",
                "columnName": "city"
            },
            {
                "name": "StateName",
                "property": "",
                "vertex": "",
                "columnName": "state"
            },
            {
                "name": "PostalCode",
                "property": "",
                "vertex": "",
                "columnName": "postalCode"
            }
        ]
    }
]
```

In the above example,

- “cableModem” is a PhysicalDeviceSpecification which has a characteristic “deviceType”. This characteristic is stored in “PROPERTIES” column of PG_DEVICE table.

```
{
    "DeviceType": "deviceType"
}
```

- A record for “Ethernet” is added to PG_DOMAIN table. All devices of “EthernetDevice” specification have containment edges to “Ethernet” in PG_Device_To_Domain table.
- “EthernetDevice” has a characteristic called “Tech”, so all unique values of “Tech” characteristic are added to PG_Technology. And for each “EthernetDevice” depending on its “Tech” characteristic respective containment edges are built.
- “Generic_Address” is a Place which has “CityName”, “StateName” and “PostalCode” characteristics which are mapped to “CITY”,”STATE” and “POSTALCODE” columns of PG_LOCATION table.

Customizing Topology JSON Files

To customize the topology JSON files:

1. When migrating Attribute or Role or Characteristic data to “PROPERTIES” column of respective entity, make sure the key used doesn’t include any empty space or special characters:

```
{
    "name": "Vendor Name",
```

```
    "property": "",
    "vertex": "vendor",
    "columnName": ""
}
```

The above example “Vendor Name” contains empty space. Instead use “VendorName” or “Vendor_Name”.

2. In topologySpecificationMapping.json if the characteristic being migrated has length greater than 30 characters or contains special characters, the <ENTITY>_CHAR_MIG, do not have the characteristic as is. Instead, it has been casted to coded value, which can be derived from “CHARACTERISTICS_TABLE_MAPPING_MIG” in UIM schema.

   For example: “Inter-rack_Power_Distribution” (CHAR_NAME) is the name of the characteristic which has been casted to “C46575002” (COLUMN_NAME).

```
{
    "name": "Inter-rack_Power_Distribution",
    "property": "",
    "vertex": "",
    "columnName": "nodeCategory"
}
```

The above example would result in a column not found error, instead characteristic must be migrated as follows:

```
{
    "name": "C46575002",
    "property": "",
    "vertex": "",
    "columnName": "nodeCategory"
}
```

Dynamic Data Mapping from UIM

The dynamic data mapping takes advantage of UIM characteristics and provides maximum flexibility for mapping fields from UIM to the topology model.

The dynamic data mapping:

- Does not require any additions, updates, migrations, or deployments of your existing specifications.
- Guarantees the value is set correctly and does not require a user to select the correct value.
- Allows UTIA to support data extensions to the topology model without an upgrade.
- Vertex and Edge Labels or Properties in UTIA may require different names than Characteristics, or Attributes or Roles in the implemented UIM model.
- These items are supported through dynamic data mapping.

The examples are:

- UIM has a 'Vendor' attribute on the Logical Device and Equipment Specifications but some users have added 'manufacturer' to their Physical Device Specifications.

- Some vertices are not identified specifically in UIM such as Domain and Service Type. These values are implied based on the '5G' cartridge or the 'FTTx' cartridge but are not specifically identified on the entity.

Prerequisites for Dynamic Data Mapping from UIM

The prerequisites are as follows:

- The following configuration files are required:
  - topologyAttributeMapping.json
  - topologyRoleMapping.json
  - topologySpecificationMapping.json.
- These files must exist in the <domain>/UIM/config/topologyMappings directory.
- Files with these names plus the extension .sample are provided.
- Prior to migration, the correct configurations must be provided. Else, the data will not be mapped correctly to UTIA.
- If the file does not exist an error occurs during UIM entity creation.
- If you want to skip this process, you can remove the **.sample** extension and proceed with the default settings.

Mapping the Dynamic Data from UIM

To map the dynamic data from UIM, the following definitions are required:

- vertex: A node in the Topology Model, examples are Vendor, Domain, Technology, Network Type, Device, Location
- property: A column on every vertex and edge in the Topology model.
  - It supports JSON allowing for unlimited additional attributes.
  - Property is the name of the key used to store the value retrieved from the UIM attribute.
- properties: is an array defining how individual attributes of an entity are to be stored in Topology schema.
- columnName: An existing column on a physical table in the Topology Model used to store the attribute.
- name: Maps to different entity classes and entity specification classes. For example: “LogicalDeviceDAO”, “EquipmentSpecificationDAO”, “PlaceSpecificationDAO”, “PropertyLocationDAO” and so on.

The following POST operation creates a logical device, you can see the relationships and properties with which the dynamic properties are supported.

POST: http://localhost:8080/vertex

Body:

```
{"entityId":<entityID>,"entityVersion":<entityVersion>,"businessObjectClass":"
LogicalDeviceDAO","id":"<ID>","name":"<name>","specName":"<specificationName>"
,"latitude":0.0,"longitude":0.0,"inventoryStatus":"INSTALLED","referenceId":<r
eferenceID>,"relationships":{"vendor":"<vendor>"},"properties":
{"deviceIdentifier":"<deviceIdentifier>"}}
```

> **Note:**
>
> - In this example, the TopologyAttributesMapping.json file provides the instructions to UTIA and the file is available in the UIM/config/topologyMappings directory.
> - The topologyAttributesMapping file is used to address hard coded attributes from UIM tables.
> - See topologyAttributesMapping.json for more information.

The POST operation tells the topology:

- Map LogicalDevice.deviceIdentifier to the property deviceIdentifer.
- Map LogicalDeviceSpecification.vendorName to the vertex = vendor
- This is based on the UIM ClassName, it works with any Class or specification that is topology-enabled.

You can add a role to the Logical Device from the list of roles that are configured in the TopologyRoleMapping.json file.

You can see that GET that the Logical Device tracks the **deviceIdentifier** in the **properties** column using:

GET: http://localhost:8080/vertex/typeid/1/referenceid/<refID>

```
{"businessObjectClass":"LogicalDeviceDAO","entityId":<entityID>,"entityVersion
":<entityVersion>,"id":"<versionID>","inventoryStatus":"INSTALLED","latitude":
0.0,"longitude":0.0,"name":"<name>","properties":
{"deviceIdentifier":"<deviceID>"},"referenceId":<referenceID>,"specName":"<spe
cificationName>"}
```

PUT: http://localhost:8080/vertex

```
{"businessObjectClass":"LogicalDeviceDAO","entityId":<entityID>,"entityVersion
":3,"id":"<ID>","inventoryStatus":"INSTALLED","latitude":0.0,"longitude":0.0,"
name":"<name>","properties":
{"deviceIdentifier":"<ID>","transmission":"Optical_Transmission"},"referenceId
":<refernceID>,"specName":"<specificationName>"}
```

In the body:

- The role “Optical_Transmission” is mapped to the property field with name = “transmission”.
- The role was given a name = “transmission” which was provided by the UIM admin.
- Add, update and delete are supported. This works for Equipment and Physical Device (any topology-enabled entity that supports roles).
- Roles can be mapped to properties, vertices or columns.

The rules to perform this are:

- The Vertex must exist. The mapping can be performed to multiple vertices and can have multiple values.

- Property: There can be multiple properties. The UIM integrator is responsible for not having similar or misspelled values.
- ColumnName: A column can only have 1 value. The user is currently responsible for assuring this value is unique. It can be overlaid. This should be used for a queried attributes where an index is needed.
- The possible values of "columnName" are the following:
  - PG_DEVICE - [NODECATEGORY, MACADDRESS, IPV4, IPV4SUBNET, IPV6, IPV6SUBNET, ZONEID, DEVICEIDENTIFIER, NETWORKSTATUS, NODETYPE]
  - PG_LOCATION - [DISTRICT, PROVINCE, OPERATOR, CITY, STATE, POSTALCODE, COUNTRY, AREA, CIRCLE]
  - PG_COMMICATION - [FROMNODEDATA, TONODEDATA, RATECODE, TECHNOLOGY]
  - PG_NETWORK - [CATEGORY, SUBCATEGORY, TOPOLOGYTYPE, SUBTYPE]

> **Note:**
>
> UIM currently supports city, state, country and postalcode attributes from the PropertyLocationDAO and PropertyAddressDAO. The street address or subunit (apt#, room #) are not supported.

The supported UIM classes are:

LogicalDeviceDAO, GeographicPlaceDAO, PhysicalDeviceDAO, NetworkDAO, NetworkEdgeDAO, EquipmentDAO, GeographicSiteDAO, PropertyLocationDAO

> **Note:**
>
> This includes the corresponding supported specification classes.

The last configuration is **TopologySpecificationMapping.json**.

- The **related vertices** field automatically adds a relationship edge between any instance of the specification to the vertex with the provided name and value.
- A characteristic does not need to be added and set on the specification to be tracked in topology.
- This allows our current RI cartridges to be used without any modifications.
- The **characteristics** column works the same as roles.
- It automatically adds a relationship to a vertex, sets properties or sets a column value.
- Any current characteristics can be used. No changes are needed.

PUT: http://localhost:8080/vertex

```
{"entityId":<entityID>,"entityVersion":<entityVersion>,"businessObjectTypeId":
1,"businessObjectClass":"LogicalDeviceDAO","id":"<ID>","name":"<name>","specNa
me":"router","latitude":<latitude>,"longitude":<longitude>,"inventoryStatus":"
INSTALLED","isTopLevelNode":true,"nodeAvailable":true,"placeNode":false,"refer
```

```
enceId":<referenceID>,"createdUser":"test","lastModifiedUser":"test","relation
ships":{"vendor":"<vendor>","domain":"Ethernet"},"properties":
{"deviceIdentifier":"<deviceID>"}}
```

7
Upgrading UTIA

This chapter describes how to upgrade the UTIA application.

Prerequisites for Upgrading UTIA

The prerequisites for upgrading UTIA are:

- UTIA Topology Schema should have a database link to the UIM schema with the name `rem_schema`. This is mandatory if only UTIA is used with UIM. However, the database link is not required if UTIA is used with some external system. The `rem_schema` database link is created during the first time of complete migration. If the database link is not present, the database link can be created as follows:

```
ACCEPT schema CHAR PROMPT "Enter username for remote schema: "
ACCEPT passwd CHAR PROMPT "Enter password for remote schema: " HIDE
ACCEPT host CHAR PROMPT "Enter pingable hostname/ipaddress for remote
schema database host : "
ACCEPT port CHAR PROMPT "Enter port number for remote schema database : "
ACCEPT service_name CHAR PROMPT "Enter SQL*Net / service for remote schema
database: "
ACCEPT commitSize CHAR PROMPT "Enter Batch/Commit size for a single
parallel process(Optional): "
ACCEPT threads CHAR PROMPT "Enter Maximum no.of total parallel process at
any given time(Optional): "
ACCEPT waitTime CHAR PROMPT "Enter Waiting interval after which the
listener checks for the availabilty of jobs in Seconds(Optional): "

PROMPT

alter system set global_names=FALSE scope=both;

CREATE DATABASE LINK rem_schema CONNECT TO &schema IDENTIFIED BY &passwd
USING '(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=&host)(PORT=&port))
(CONNECT_DATA=(SERVICE_NAME=&service_name)))';
```

- For UTIA 1.0.0.1.0 or later versions, the installer will create an **ApplicationInfo** table and will update the **VERSION** after every upgrade. If you have UTIA 1.0.0.0.0 installed, you will not be having **ApplicationInfo** table. Therefore, create **ApplicationInfo** table before running an upgrade as follows:

```
CREATE TABLE APPLICATIONINFO ( ENTITYID NUMBER(19,0) NOT NULL ENABLE,
  ENTITYCLASS VARCHAR2(255 BYTE),
  BUILDDATE TIMESTAMP (6) WITH LOCAL TIME ZONE,
  CREATEDDATE TIMESTAMP (6) WITH LOCAL TIME ZONE,
  CREATEDUSER VARCHAR2(255 BYTE),
  ENDDATE TIMESTAMP (6) WITH LOCAL TIME ZONE,
  ENTITYVERSION NUMBER(10,0),
  FILENAME VARCHAR2(255 BYTE),
  LASTMODIFIEDDATE TIMESTAMP (6) WITH LOCAL TIME ZONE,
```

```
  LASTMODIFIEDUSER VARCHAR2(255 BYTE),
  NAME VARCHAR2(255 BYTE),
  STARTDATE TIMESTAMP (6) WITH LOCAL TIME ZONE,
  STATUS VARCHAR2(255 BYTE),
  TYPE VARCHAR2(255 BYTE),
  VERSION VARCHAR2(255 BYTE),
   PRIMARY KEY (ENTITYID));

   INSERT INTO APPLICATIONINFO VALUES (ENTITYID_SEQ.NEXTVAL,
'ApplicationInformationDAO', SYSDATE, SYSDATE, NULL, SYSDATE, 1, NULL,
SYSDATE, NULL, 'Unified Topology for Inventory and Automation', SYSDATE,
'SUCCESS', 'Topolgy', '1.0.0.0.0');
```

Upgrading the UTIA Application

To upgrade the UTIA application:

1. Download the latest Unified Topology Builder Tool Kit and Common Cloud Native Tool Kit into the workspace directory.
2. Export the unzipped path to the **WORKSPACEDIR** environment variable.

   ```
   export WORKSPACEDIR=$(pwd)/workspace
   ```

3. Set the **COMMON_CNTK** variable to the path of the common-cntk directory in the workspace.

   ```
   export COMMON_CNTK=$WORKSPACEDIR/common-cntk
   ```

4. Set **SPEC_PATH** variable to the location where applications.yaml and database.yaml files are copied :

   ```
   $ export SPEC_PATH=$WORKSPACEDIR/utia_spec_dir
   ```

5. Create UTIA images using the latest Unified Topology Builder Tool Kit. See "Creating UTIA Images" for more information.
6. Upgrade the UTIA schema. See "Upgrading the UTIA Schema" for more information.
7. Upgrade the UTIA instance. See "Upgrading the UTIA Instance" for more information.

Upgrading the UTIA Schema

To upgrade the UTIA schema:

1. Upgrade PDB by starting $UIM_CNTK/scripts/install-database.sh.
2. To only update the model of UTIA and skip the data migration:

   ```
   $COMMON_CNTK/scripts/install-database.sh -p sr -i quick -f $SPEC_PATH/sr/
   quick/database.yaml -a unified-topology -c 4
   ```

3. To update the model of UTIA and also populate the data from the UIM schema:

```
$COMMON_CNTK/scripts/install-database.sh -p sr -i quick -f $SPEC_PATH/sr/
quick/database.yaml -a unified-topology -c 40
```

Upgrading the UTIA Instance

To upgrade the UTIA instance:

1. Update $COMMON_CNTK/samples/applications.yaml with the latest UTIA API, Unified PGX, and UTIA UI image names and the corresponding tags.
2. Run $COMMON_CNTK/scripts/upgrade-applications.sh to upgrade the UTIA instance:

```
$COMMON_CNTK/scripts/upgrade-applications.sh -p sr -i quick -
f $SPEC_PATH/sr/quick/applications.yaml -a unified-topology
```

8

Checklists for Integration of Services

This chapter provides a checklist of integrating the services.

The checklists include the following variables:

- <topology-project>: Refers to the Kubernetes namespace on which the UTIA service is running.
- <topology-instance>: Refers to the instance name of the UTIA service running on<topology-project> namespace.
- <messaging-project>: Refers to the Kubernetes namespace on which Message Bus service is running.
- <messaging-instance>: Refers to the instance name of Message Bus running on <messaging-project> namespace.
- <loadbalancerport>: Refers to the port of loadbalancer configured. If you use Oracle Cloud Infrastructure LBaaS, or any other external load balancer, if TLS is enabled set loadbalancerport to 443. Otherwise, set loadbalancerport to 80. If there is no external loadbalancer configured for the instance, change the value of loadbalancerport to the default Traefik NodePort. If TLS is enabled on Unified Topology Traefik NodePort is 30443 and if TLS is disabled, is 30305.
- <loadbalancerhost>: Refers to the host of loadbalancer configured. If you use Oracle Cloud Infrastructure LBaaS, or any other external load balancer, update the value for loadbalancerhost appropriately. If there is no external loadbalancer configured for the instance change the value of loadbalancerhost to the worker node IP/ Kubernetes cluster IP.
- <hostSuffix> : Refers to the host suffix configured using applications.yaml file. The default is: uim.org.
- <oauth-token-endpoint-uri>: Get the OAuth token endpoint URI from your IdP. Usually, you can find it on .well-known/openid-configuration endpoint of your IdP. In case of OAM, it is https://<instance>.<project>.ohs.<hostSuffix>:<loadbalancerport>/oauth2/rest/token
- <oauth-scope>: Provide the configured scope to your OAuth client. If not configured, keep it empty.
- <oauth-audience>: Provide the configured audience to your OAuth client. If not configured, keep it empty.

Use the following checklist for integrating UIM cloud native instance, Message Bus, and UTIA:

Table 8-1 Checklist for UIM cloud native instance, Message Bus, and UTIA

Source Application	SSL Enablement	Deployment Configuration	Application Properties

UIM CN	See **Setting Up Secure Communication with SSL** in *UIM Cloud Native Deployment Guide*.	See **Enabling OAM Authentication** in *UIM Cloud Native Deployment Guide*	For communications between applications on the same Kubernetes cluster provide internal Kubernetes service details.
			Configure the Message Bus and UTIA settings.
			See *UIM System Administrator's Guide* for more information.
			$UIM_CNTK/charts/uim/custom-config.properties
			UIM CN to Message Bus service settings
			bootstrap.server.url=<messaging-project>-<messaging-instance>-messaging-kafka-bootstrap.<messaging-project>.svc.cluster.local:9092
			#Set below properties to pass Authentication service details
			kafka.client.isOAuth=true
			kafka.client.oauth.token.endpoint.uri=https://<oam-instance>.<oam-project>.ohs.<oam-host-suffix>:<loadbalancerport>/oauth2/rest/token
			Note: This is applicable only if OAM is used as IdP. Otherwise, use the Token URL from the IdP.
			kafka.client.oauth.token.endpoint.uri=<oauth-token-endpoint-uri>
			kafka.client.oauth.scope=<oauth-scope>
			kafka.client.oauth.audience=<oauth-audience>
			kafka.client.oauth.client.id= <oauth-client-id>
			kafka.client.oauth.client.secret= <oauth-client-secret>
			#Internal commmunications between kubernetes services is non-ssl. Set kafka.client.isTLs to false.
			kafka.client.isTLs=false
			UIM CN to Unified Topology API settings
			disableTopology=false
			microServiceEnabled=true
			For Same Namespace: microServiceUrl=http://<topology-project>-<topologyinstance>-unified-topology-api:8080/topology/v2/
			For Different Namespace : microServiceUrl=http://<topology-project>-<topologyinstance>-unified-topology-api.<namespace>.svc.cluster.local:8080/topology/v2/
			UIM CN to Unified Topology UI settings
			uim.rest.filter.CORSAllowedOrigin=https://<topology-instance>.<topology-project>.topology.<hostSuffix>:<loadbalancerport>
			topology.ui.host= https://<topology-instance>.<topology-project>.topology.<hostSuffix>
			topology.ui.port= <loadbalancerport>
			topology.ui.path=/apps/unified-topology-ui

Table 8-1 (Cont.) Checklist for UIM cloud native instance, Message Bus, and UTIA

Source Application	SSL Enablement	Deployment Configuration	Application Properties
Message Bus	N/A	See **Enable Authentication on Kafka Cluster** from "Configuring Authentication"	N/A
Topology API or UI	"Setting up Secure Communication using TLS"	"Creating Secrets" "Configuring the applications.yaml File" "Registering UTIA in Identity Provider"	"Integrate Unified Topology Service with Message Bus Service"

Use the following checklist for integrating traditional UIM, Message Bus, and UTIA:

Checklist for entries in **/etc/hosts** for integration:

- Authentication service
  - If OAM is deployed as IdP:

    ```
    <loadbalancerIP>  <oam-instance>.<oam-project>.ohs.<oam-host-suffix>
    ```

  - If any IdP is used other than OAM, for accessing UIM:

    ```
    <instance>.<project>.<hostSuffix>
    ```

- Message service

  ```
  <loadbalacerIP> <messaging-instance>.<messaging-
  project>.messaging.bootstrap.<hostSuffix>
  <loadbalacerIP> <messaging-instance>.<messaging-
  project>.messaging.broker0.<hostSuffix>
  <loadbalacerIP> <messaging-instance>.<messaging-
  project>.messaging.broker1.<hostSuffix>
  ```

- UTIA service

  ```
  <loadbalancerIP>  <topology-instance>.<topology-
  project>.topology.<hostSuffix>
  ```

Table 8-2 Checklist for UIM, Message Bus, and UTIA

Source Application	SSL Enablement	Deployment Configuration	Application Properties

UIM	N/A	For enabling SSO authentication on UIM On Premise instance, see **Setting Up Unified Inventory Management for Single Sign-On Authentication** section in *UIM Installation Guide*.	**UIM on-prem to Message Bus settings**
			Provide ingress bootstrap server details as UIM traditional instance is outside of kubernetes cluster. External access is TLS enabled
			bootstrap.server.url=<messaging-instance>.<messaging-project>.messaging.bootstrap.uim.org:<loadbalancerport>
			#set below properties to pass Authentication service details kafka.client.isOAuth=true
			kafka.client.oauth.token.endpoint.uri=<oauth-token-endpoint-uri>
			kafka.client.oauth.scope=<oauth-scope>
			kafka.client.oauth.audience=<oauth-audience>
			kafka.client.oauth.client.id=<oauth-client-id>
			kafka.client.oauth.client.secret=<oauth-client-secret>
			#External commmunications is ssl enabled, provide truststore details.
			kafka.client.isTLs=true
			Add common certificate to JAVA HOME of UIM managed servers:
			`keytool -import -alias common-cert -keystore $JAVA_HOME/jre/lib/security/cacerts -file $COMMON_CNTK/certs/commoncert.pem`
			Configure the UTIA settings.
			See *UIM System Administrator's Guide* for more information.
			UIM on-prem to UTIA API settings
			#provide Unified Topology API kubernetes service name and port along with endpoint as provided in the sample below.
			disableTopology=false
			microServiceEnabled=true
			microServiceUrl=https://<topology-instance>.<topology-project>.topology.<hostSuffix>:<loadbalancerport>/topology/v2
			UIM on-prem to UTIA UI settings
			uim.rest.filter.CORSAllowedOrigin=https://<topology-instance>.<topology-project>.topology.<hostSuffix>:<loadbalancerport>
			topology.ui.port=<loadbalancerport>
			topology.ui.path=/apps/unified-topology-ui

Message Bus	See **Message Bus Ingress Listener** in "Configuring Message Bus Listeners"	See **Enable Authentication on Kafka Cluster** from "Configuring Authentication"	N/A

Table 8-2 (Cont.) Checklist for UIM, Message Bus, and UTIA

Source Application	SSL Enablement	Deployment Configuration	Application Properties
Topology API or UI	"Setting up Secure Communication using TLS"	"Creating Secrets" "Configuring the applications.yaml File" "Registering UTIA in Identity Provider"	"Integrate Unified Topology Service with Message Bus Service"

Integrating UIM with UTIA and Message Bus

This section provides you with instructions to integrate UIM (traditional and cloud native) with UTIA and Message Bus. The samples for IDCS Idp are packaged along with UTIA.

Integrating UIM CN with Message Bus and UTIA

To integrate UIM CN with Message Bus and UTIA:

1. Update **$UIM_CNTK/charts/uim/custom-config.properties** file with the following details:
   - UIM CN to Message Bus service settings:

```
bootstrap.server.url=<messaging-project>-<messaging-instance>-messaging-
kafka-bootstrap.<messaging-project>.svc.cluster.local:9092
#Set below properties to pass Authentication service details
kafka.client.isOAuth=true
kafka.client.oauth.token.endpoint.uri=<oauth-token-endpoint-uri> (Ex.
https://idcs-df3***********f64b21.identity.pint.oc9qadev.com:443/
oauth2/v1/token)
kafka.client.oauth.client.id=<oauth-client-id> (Ex.
e6e0b2cxxxxxxxxxxxxxxx)
kafka.client.oauth.client.secret=<oauth-client-secret> (Ex. xxxx-xxxx-
xxxx-xxxx)
kafka.client.oauth.client.scope=<oauth-client-scope> (Ex. https://
quick.sr.topology.uim.org:30443/utiaScope)
kafka.client.oauth.client.audience=<oauth-client-audience> (Ex. https://
quick.sr.topology.uim.org:30443/)
#Internal commmunications between kubernetes services is non-ssl. Set
kafka.client.isTLs to false.
kafka.client.isTLs=false
```

   - UIM CN to UTIA API settings:

```
disableTopology=false
microServiceEnabled=true
microServiceUrl=http://<topology-project>-<topology-instance>-unified-
topology-api:8080/topology/v2/
```

- UIM CN to UTIA UI settings:

```
uim.rest.filter.CORSAllowedOrigin=https://<topology-instance>.<topology-
project>.topology.<hostSuffix>:<loadbalancerport>
topology.ui.host=https://<topology-instance>.<topology-
project>.topology.<hostSuffix>
topology.ui.port=<loadbalancerport>
topology.ui.path=/apps/unified-topology-ui
```

2. Create or restart the UIM CN instance as usual, after the above configurations.

Integrating Traditional UIM with Message Bus and UTIA

To integrate traditional UIM with Message Bus and UTIA:

1. Update the **system-config.properties** file with the following details:
   - UIM to Message Bus service settings:

```
Provide ingress bootstrap server details as UIM traditional instance is
outside of kubernetes cluster.
bootstrap.server.url=<messaging-instance>.<messaging-
project>.messaging.bootstrap.uim.org:<loadbalancerport>
#Set below properties to pass Authentication service details
kafka.client.isOAuth=true
kafka.client.oauth.token.endpoint.uri=<oauth-token-endpoint-uri>  (Ex.
https://idcs-df3***********f64b21.identity.pint.oc9qadev.com:443/
oauth2/v1/token)
kafka.client.oauth.client.id= <oauth-client-id> (Ex.
e6e0b2cxxxxxxxxxxxxxxx)
kafka.client.oauth.client.secret= <oauth-client-secret> (Ex. xxxx-xxxx-
xxxx-xxxx)
kafka.client.oauth.client.scope=<oauth-client-scope> (Ex. https://
quick.sr.topology.uim.org:30443/utiaScope)
kafka.client.oauth.client.audience=<oauth-client-audience> (Ex. https://
quick.sr.topology.uim.org:30443/)
# External access is TLS enabled
kafka.client.isTLs=true
```

   - UIM to UTIA API settings:

```
disableTopology=false
microServiceEnabled=true
microServiceUrl=https://<topology-instance>.<topology-
project>.topology.<hostSuffix>/topology/v2/
```

   - UIM to UTIA UI settings:

```
uim.rest.filter.CORSAllowedOrigin=https://<topology-instance>.<topology-
project>.topology.<hostSuffix>:<loadbalancerport>
topology.ui.host=https://<topology-instance>.<topology-
project>.topology.<hostSuffix>
topology.ui.port=<loadbalancerport>
topology.ui.path=/apps/unified-topology-ui
```

2. Add the Identity Providers certificate to **JAVA_HOME** as follows:

```
keytool -import -alias idp-cert -keystore $JAVA_HOME/jre/lib/security/
cacerts -file <idp-certificate-file>
```

3. Add the UTIA certificate to **JAVA_HOME** as follows:

```
keytool -import -alias utia-cert -keystore $JAVA_HOME/jre/lib/security/
cacerts -file <utia-certiricate>
```

4. Add the common certificate to **JAVA_HOME** as follows:

```
keytool -import -alias common-cert -keystore $JAVA_HOME/jre/lib/security/
cacerts -file $COMMON_CNTK/certs/commoncert.pem
```

> **Note:**
>
> Make sure that UTIA and Message bus are configured with **commoncert.pem**.