Oracle® Communications Offline Mediation Controller

Cloud Native Installation and Administration Guide

Release 15.1
G20498-02
August 2025

Oracle Communications Offline Mediation Controller Cloud Native Installation and Administration Guide, Release 15.1

G20498-02

Contents

12 Monitoring and Maintaining Offline Mediation Controller Cloud Native

13 Deploying into Oracle Cloud Infrastructure

14 Building Your Own Images

Preface

This guide provides general information on how to configure and install Oracle Communications Offline Mediation Controller in a cloud native environment.

Audience

This document is intended for DevOps administrators and those involved in installing and maintaining an Oracle Communications Offline Mediation Controller cloud native deployment.

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 customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.

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

Overview of the Offline Mediation Controller Cloud Native Deployment

You can configure Oracle Communications Offline Mediation Controller to run as a cloud native application in a containerized and orchestrated deployment architecture.

Topics in this document:

About the Offline Mediation Controller Cloud Native Deployment

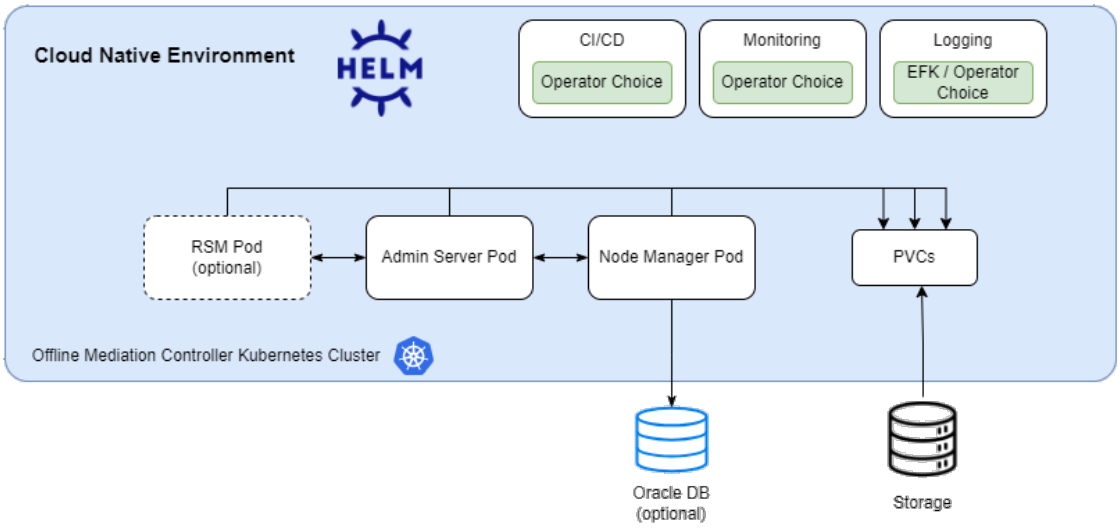
Oracle Communications Offline Mediation Controller supports its deployment on a cloud native environment. This allows you to harness the benefits of cloud with the services of Offline Mediation Controller. For more information about Offline Mediation Controller, see "Overview of Offline Mediation Controller" in *Offline Mediation Controller User's Guide*.

You can also set up your own cloud native environments. You use the cloud native deployment package to automate the deployment of Offline Mediation Controller products and speed up the process to get services up and running, with product deployments preconfigured to communicate with each other through Helm charts.

Offline Mediation Controller Cloud Native Deployment Architecture

Figure 1-1 shows the pods and other components in a typical Offline Mediation Controller cloud native deployment.

Figure 1-1 Offline Mediation Controller Cloud Native Deployment Architecture



2

Planning Your Installation

The Oracle Communications Offline Mediation Controller cloud native deployment package includes Docker images and Helm charts to help you deploy and manage pods of product services in Kubernetes.

Topics in this document:

Overview of the Offline Mediation Controller Deployment Package

The Offline Mediation Controller cloud native deployment package includes the following:

- Ready-to-use images and Helm charts to help you orchestrate containers in Kubernetes.
- Sample Dockerfiles and scripts that you can use as a reference for building your own images.

You can use the images and Helm charts to help you deploy and manage pods of Offline Mediation Controller product services in Kubernetes. Communication between pods of services of Offline Mediation Controller products is preconfigured in the Helm charts.

About Offline Mediation Controller Pods and Images

Table 2-1 lists the pods and images for Offline Mediation Controller whose containers are created, and services are exposed through them.

Table 2-1 Offline Mediation Controller Pods and Images

Pod	Replica Type	Image Name	Container Port
ocomc-admin-server-0	Single	**oc-cn-ocomc-core:15.1.0.0.0**	55105
nm-cc-0	Multiple	**oc-cn-ocomc-core:15.1.0.0.0**	55109
ocomc-rsm-54c56bf4c4-7b8td	Multiple	**oc-cn-ocomc-rsm:15.1.0.0.0**	8080
mediation-ui-6dc7556f9-9cbgf	Single	**oc-cn-ocomc-mediation-ui:15.1.0.0.0**	8080

About Offline Mediation Controller Services

Table 2-2 lists the services for Offline Mediation Controller.

Table 2-2 Offline Mediation Controller Services

Service	Service Type	Port	Notes
nm-cc-0	ClusterIP	55105	Client on same worker node.
nm-cc-0-metrics	ClusterIP	9090	Client on same worker node.
nm-epdc-0	ClusterIP	55109	Client on same worker node.
nm-epdc-0-metrics	ClusterIP	9090	Client on same worker node.
ocomc-admin-server	ClusterIP	31200, 31201, 31202	Client on same worker node. 31200 is the default service port, 31201 is the firewall port and 31202 is the callback port.
ocomc-rsm	NodePort	31250	Client on Windows system.

To connect to the Administration Server running in a Kubernetes cluster, you must install the Administration Client outside of the Kubernetes cluster and then connect it to the service and port of the Administration Server.

- If the Administration Client is installed on the same node where the Administration Server pod is running, use a clusterIP service type with the Administration Server.
- If the Administration Client is located remotely or is on a Windows system, use a NodePort service type with the Administration Server.

For more information about connecting the Administration Client, see "Connecting Your Administration Client".

3

Preparing Your Offline Mediation Controller Cloud Native Environment

You prepare your system for the Oracle Communications Offline Mediation Controller cloud native deployment by installing all prerequisite software and downloading the Offline Mediation Controller Helm charts and images.

Topics in this document:

Tasks for Preparing Your Offline Mediation Controller Cloud Native Environment

To prepare your system for the Offline Mediation Controller cloud native deployment:

1. If you want to integrate Offline Mediation Controller with Elastic Charging Engine (ECE) and the Billing and Revenue Management (BRM) cloud native deployment package, set up the ECE and BRM cloud native deployment prior to setting up Offline Mediation Controller. See *BRM Cloud Native Deployment Guide*.
2. Set up your Offline Mediation Controller environment by installing and configuring all prerequisite software. See "Setting Up Your Environment".
3. Download the Helm charts for the Offline Mediation Controller cloud native deployment. See "Downloading Packages for the Offline Mediation Controller Cloud Native Helm Charts".
4. Download the Offline Mediation Controller cloud native images in one of these ways:
   - From the Oracle Container Registry. To do so, see "Pulling Offline Mediation Controller Images from the Oracle Container Registry".
   - From the Oracle Software Delivery website. To do so, see "Downloading Offline Mediation Controller Images from Oracle Software Delivery Website".

Setting Up Your Environment

Set up your environment with the following technologies installed, configured, and tuned for performance, networking, security, and high availability. Make sure backup nodes are available in case of system failure in any of the cluster's active nodes.

- **Podman:** The Podman tool is used to containerize Offline Mediation Controller products.

For more information, see the Podman documentation (https://docs.podman.io/en/latest/index.html).

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

  Set up a Kubernetes cluster for your BRM cloud native deployment, securing access to the cluster and its objects with the help of service accounts and proper authentication and authorization modules. Also, set up the following in your cluster:

  – **Volumes:** A container's file system lives only as long as the container does. When a container terminates and restarts, file system changes are also lost. You shouldn't access the container file system or pods frequently, and sharing data between container and host systems is not easy. Volumes appear as a directory in the container file system and provide a way to share data. The Offline Mediation Controller cloud native deployment package uses persistent volumes for sharing data in and out of containers but doesn't enforce any particular type. You can choose from the volume type options available in Kubernetes.

    You can choose an external incubator to create persistent volumes, but ensure it supports the ReadWriteMany access mode and PVC sharing between pods.

  – **A networking model:** Kubernetes assumes that pods can communicate with other pods, regardless of which host they land on. Every pod has a different IP address, so you don't need to explicitly create a link between pods. You rarely need to deal with mapping container ports to host ports. While Kubernetes doesn't offer a solution to support its assumption, several implementations meet the fundamental requirements of Kubernetes' networking model. Choose the networking element depending on the cluster requirement.

  For more information about Kubernetes, see *Kubernetes Concepts* (https://kubernetes.io/docs/concepts/).

  > **Note**
  >
  > Secure your cluster according to standard DevOps practices.

- **Fluentd:** Fluentd forms your logging layer, collecting log files from your Offline Mediation Controller service pods and transforming them. The Fluentd-concat plugin is used to concatenate multiline log files. You set up Fluentd on your Kubernetes nodes. Configure all applications to redirect their logs to STDOUT so Fluentd can parse your log files.

  For more information about Fluentd, see *Fluentd Overview* (https://docs.fluentd.org/quickstart).

- **Helm:** Helm is a package manager that helps you install and maintain software on a Kubernetes system. In Helm, a package is called a *chart*, which consists of YAML files and templates rendered into Kubernetes manifest files. The BRM cloud native deployment package includes Helm charts that help create Kubernetes objects, such as ConfigMaps, Secrets, controller sets, and pods, with a single command.

  For more information about Helm, see *Helm Introduction to Helm* (https://helm.sh/docs/using_helm/).

- **Oracle Database:** An Oracle database must be installed and accessible through the Kubernetes network so that the pods can perform database operations. It can be either a CDB or a non-CDB.

For the complete list of software compatible with the Offline Mediation Controller cloud native deployment package, see "Offline Mediation Controller Cloud Native Deployment Software Compatibility" in *Offline Mediation Controller Compatibility Matrix*.

Downloading Packages for the Offline Mediation Controller Cloud Native Helm Charts

To download the Offline Mediation Controller cloud native Helm charts:

1. Go to https://edelivery.oracle.com.
2. Sign into the Oracle Software Delivery website using an Oracle account.
3. Search for and select **Oracle Communications Offline Mediation Controller Cloud Native Deployment Option 15.1.0.*x*.0** and then click **Continue**.
4. Make sure all packages are selected, and then click **Continue**.
5. Accept the Oracle standard terms and restrictions, and then click **Continue**.
6. Select the Docker Helm chart packages and then click **Download**.

   Each package is downloaded to a separate Zip file.
7. Extract the following Helm chart and Docker archive files from each Zip file:
   - Offline Mediation Controller Dockerfiles: **oc-cn-ocomc-core-docker-files-15.1.0.*x*.0.tgz**
   - Offline Mediation Controller Core and REST Services Manager Helm Chart: **oc-cn-ocomc-helm-chart-15.1.0.*x*.0.tgz**
   - Offline Mediation Designer UI Helm Chart: **oc-cn-ocomc-mediation-ui-helm-chart-15.1.0.*x*.0.tgz**
   - Ingress Controller Sample Helm Chart: **oc-cn-ocomc-nginx-ingress-controller-sample-helm-chart-15.1.0.*x*.0.tgz**
   - Apache Relying Party Sample Helm Chart: **oc-cn-ocomc-apache-relying-party-sample-helm-chart-15.1.0.*x*.0.tgz**
8. Extract the Helm charts and Docker files from the archive files by running these commands:

```
tar xvzf oc-cn-ocomc-core-docker-files-15.1.0.x.0.tgz

tar xvzf oc-cn-ocomc-helm-chart-15.1.0.x.0.tgz

tar xvzf oc-cn-ocomc-mediation-ui-helm-chart-15.1.0.x.0.tgz

tar xvzf oc-cn-ocomc-nginx-ingress-controller-sample-helm-chart-15.1.0.x.0.tgz

tar xvzf oc-cn-ocomc-apache-relying-party-sample-helm-chart-15.1.0.x.0.tgz
```

Pulling Offline Mediation Controller Images from the Oracle Container Registry

To pull Offline Mediation Controller cloud native images from the Oracle Container Registry, do the following:

1. In a web browser, go to https://container-registry.oracle.com.
2. Sign into the Oracle Container Registry using an Oracle account.

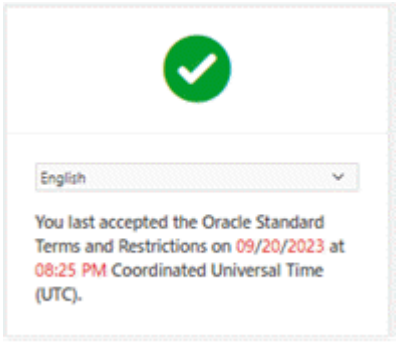
> **Note**
>
> To pull images for licensed software on the Oracle Container Registry, you must have an Oracle account. You can create an Oracle account at https://profile.oracle.com/myprofile/account/create-account.jspx.

3. Select the **Oracle Communications Cloud Scale Monetization** container.

   The Oracle Communications Cloud Scale Monetization page appears.
4. Click one of the following repository names:
   - **oc-cn-ocomc-core**: Offline Mediation Controller image
   - **oc-cn-ocomc-rsm**: Offline Mediation Controller REST Services Manager image
   - **oc-cn-ocomc-mediation-ui**: Offline Mediation Controller Offline Mediation Designer UI image

   The repository page appears.
5. Accept the Oracle terms and restrictions by:
   a. For non-CPU repositories, selecting your desired language.
   b. Clicking **Continue**.
   c. Scrolling to the bottom of the terms and restrictions page, and clicking **Accept**.

   If successful, you will see something similar to this:



6. On your host system, log in to the Oracle Container Registry using the Podman command-line interface (CLI):

```
podman login container-registry.oracle.com
```

7. When prompted for a user name and password, enter your Oracle credentials.
8. Pull the Offline Mediation Controller cloud native image from the registry:

   ```
   podman pull container-registry.oracle.com/communications_monetization/imageName:tag
   ```

   where:

   - *imageName* is the name of the software image: **oc-cn-ocomc-core** or **oc-cn-ocomc-rsm**.
   - *tag* is the tag name for the image, such as **15.1.0.*x*.0**.

   For example, to pull the Offline Mediation Controller cloud native image from the registry:

   ```
   podman pull container-registry.oracle.com/communications_monetization/oc-cn-ocomc-core:15.1.0.x.0
   ```

   The image is pulled from the Oracle Container Registry and stored locally, where it is ready to be used to deploy containers.
9. Confirm the images were pulled from the Oracle Container Registry:

   ```
   podman images
   ```

   If successful, you will see something similar to this:

   ```
   REPOSITORY
   TAG            IMAGE ID        CREATED
   container-registry.oracle.com/communications_monetization/oc-cn-ocomc-core
   15.1.0.x.0     133dd3580b87    2 seconds ago
   container-registry.oracle.com/communications_monetization/oc-cn-ocomc-rsm
   15.1.0.x.0     136dd3593b47    3 seconds ago
   container-registry.oracle.com/communications_monetization/ocomc-mediation-ui
   15.1.0.x.0     136dd3598k23    5 minutes ago
   ```
10. Log out of the registry to prevent unauthorized access and to remove any record of sign-in credentials that Podman might store for future operations:

    ```
    podman logout container-registry.oracle.com
    ```

Downloading Offline Mediation Controller Images from Oracle Software Delivery Website

To download Offline Mediation Controller cloud native images from the Oracle Software Delivery website:

1. Go to https://edelivery.oracle.com.
2. Sign in to the Oracle Software Delivery website using an Oracle account.
3. Search for and select **Oracle Communications Offline Mediation Controller Cloud Native Deployment Option 15.1.0.*x*.0**.
4. Download Zip files for the following:
   - Oracle Communications Offline Mediation Controller Cloud Native Deployment Option REST Services Manager 15.1.0.*x*.0
   - Oracle Communications Offline Mediation Controller Cloud Native Deployment Option 15.1.0.*x*.0
5. From the Zip files, extract the following archive files:
   - Offline Mediation Controller image (**oc-cn-ocomc-core-15.1.0.*x*.0.tar**)

- Offline Mediation Controller REST Services Manager image (**oc-cn-ocomc-rsm-15.1.0.*x*.0.tar**)
- Offline Mediation Designer UI image (**oc-cn-ocomc-mediation-ui-helm-chart-15.1.0.*x*.0.tar**)

6. Load the files as images into your Podman system using the following command:

   ```
   podman load --input fileName
   ```

   where *fileName* is **oc-cn-ocomc-core-15.1.0.*x*.0.tar**, **oc-cn-ocomc-rsm-15.1.0.*x*.0.tar**, or **oc-cn-ocomc-mediation-ui-helm-chart-15.1.0.*x*.0.tar**.

If you use an internal registry to access images from different Kubernetes nodes, push the images from your local system to the registry server. For example, if the registry is identified by *RepoHost*:*RepoPort*, you would push the Offline Mediation Controller REST Services Manager image to the registry using the Podman CLI like this:

1. Tag the Offline Mediation Controller REST Services Manager image with the registry server:

   ```
   podman tag ocomc-rsm:15.1.0.x.0 RepoHost:RepoPort/oc-cn-ocomc-rsm:15.1.0.x.0
   ```

2. Push the image to the registry server:

   ```
   podman push RepoHost:RepoPort/oc-cn-ocomc-rsm:15.1.0.x.0
   ```

4

Installing the Offline Mediation Controller Cloud Native Deployment Package

Learn how to install the Oracle Communications Offline Mediation Controller cloud native deployment package on a cloud native environment.

Topics in this document:

About Deploying into Kubernetes

Helm is the recommended package manager for deploying Offline Mediation Controller cloud native services into Kubernetes. A Helm chart is a collection of files that describe a set of Kubernetes resources. It includes YAML template descriptors for all Kubernetes resources and a **values.yaml** file that provides default configuration values for the chart.

The Offline Mediation Controller cloud native deployment package includes **oc-cn-ocomc-core-helm-chart-15.1.0.*x*.0.tgz**.

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

Automatically Pulling Images from Private Docker Registries

You can automatically pull images from your private Docker registry by creating an **ImagePullSecrets**, which contains a list of authorization tokens (or Secrets) for accessing a private Docker registry. You then add references to the **ImagePullSecrets** in your Offline Mediation Controller Helm chart's **override-values.yaml** file. This allows pods to submit the Secret to the private Docker registry whenever they want to pull images.

Automatically pulling images from a private Docker registry involves these high-level steps:

1. Create a Secret outside of the Helm chart by entering this command:

   ```
   kubectl create secret docker-registry SecretName --docker-
   server=RegistryServer --docker-username=UserName --docker-
   password=Password -n NameSpace
   ```

   where:

   - *SecretName* is the name of your Kubernetes Secret.
   - *RegistryServer* is your private Docker registry's fully qualified domain name (FQDN) (*repoHost*:*repoPort*).
   - *UserName* and *Password* are your private Docker registry's user name and password.
   - *NameSpace* is the namespace you use for installing the Offline Mediation Controller Helm chart.

   For example:

   ```
   kubectl create secret docker-registry my-docker-registry --docker-
   server=example.com:2660 --docker-username=xyz --docker-password=password -n oms
   ```

2. Add the **imagePullSecrets** key to your **override-values.yaml** file for **oc-cn-ocomc-core**:

   ```
   imagePullSecrets: SecretName
   ```

3. Add the **ocomc.imageRepository** key to your **override-values.yaml** file:

   ```
   imageRepository: "RegistryServer"
   ```

4. Deploy **oc-cn-ocomc-core**.

Automatically Rolling Deployments by Using Annotations

Whenever a ConfigMap entry or a Secret file is modified, you must restart its associated pod. This updates the container's configuration, but the application is notified about the configuration updates only if the pod's deployment specification has changed. Thus, a container could be using the new configuration, while the application keeps running with its old configuration.

You can configure a pod to automatically notify an application when a Container's configuration has changed. To do so, configure a pod to automatically update its deployment specification whenever a ConfigMap or Secret file changes by using the **sha256sum** function. Add an **annotations** section similar to this one to the pod's deployment specification:

```
kind: Deployment
spec:
  template:
    metadata:
      annotations:
        checksum/config: {{ include (print $.Template.BasePath "/
configmap.yaml") . | sha256sum }}
```

For more information, see *Chart Development Tips and Tricks* in the Helm documentation (https://helm.sh/docs/howto/charts_tips_and_tricks/#automatically-roll-deployments).

About StatefulSet Implementation

You can implement StatefulSet deployment for Node Managers in Kubernetes. StatefulSets ensure that each pod has a stable and unique network identity and consistent storage, simplifying scaling through the Horizontal Pod Autoscaler. You can use the StatefulSet controller to create and delete pods and confirm that each new pod receives a consistent identity and associated resources. You can also customize the deployed StatefulSets to meet your specific requirements.

About Sidecars

Offline Mediation Controller cloud native uses Kubernetes sidecars to interact with the Offline Mediation Controller REST Services Manager. For more information about sidecars, see "Sidecar Containers" in the Kubernetes documentation.

Offline Mediation Controller cloud native deploys two types of sidecars:

- Node Manager Sidecar
- Admin Server Sidecar

About the Node Manager Sidecar

You can use Node Manager sidecars to perform tasks related to the Node Managers.

- If the Node Manager is running, you can register it with the Administration Server. This ensures that the Node Manager can immediately participate in cluster activities.
- If the replication function is enabled, you can manage the replication for new Node Managers. New Node Managers replicate the node chain and inherit necessary state and data from the parent Node Manager through this replication process.
- You can persist the registration and replication process of the new Node Managers to preserve them across pod restarts. This ensures that after a pod restart, the sidecar can resume its operations accurately.

About the Admin Server Sidecar

You can use admin server sidecars to perform tasks related to the Admin server.

- If the import on install function is enabled, you can check whether all Node Managers involved in the import process are available and registered. If a Node Manager is unavailable, the sidecar waits until it is available. The sidecar runs the upload and imports APIs after all the Node Managers are available. It also continues to monitor the import process until it completes successfully.
- In the event of a failure during the import process, you can diagnose and resolve issues. The sidecar handles error scenarios and marks the import as a dangling request.
- You can persist the states of the operations to preserve them across pod restarts. This ensures that after a pod restart, the sidecar can resume its operations accurately.

Configuring Sidecars

You can configure debugging and analysis of applications using the following entries in the respective ConfigMap files:

- **SIDE_CAR_INTERVAL:** You can define the frequency at which the sidecar should handle its closed-loop operations. The value is in milliseconds, and the default value is **10000**.
- **SIDECAR_NODE_MANAGER_AUTO_REGISTRATION_DISABLE:** You can define whether to register a Node Manager with the administration server on startup. The value can be either **TRUE** or **FALSE**. The default value is set to **FALSE**.

> **Note**
>
> The names of the ConfigMap files depend on the names of the Node Manager sets which are set using the **ocomcCore.ocomc.nodeManagerConfigurations.sets** key.

About Data Persistent Volume (PV) Configuration

You can control the sharing of the data Persistent Volume (PV) across Node Managers with more granularity using the **ocomcCore.ocomc.nodeManagerConfigurations.storage.data.scope** key. You can set the **scope** key to one of these values:

- **Application:** The data PV is shared across all applications. All sets and their pods share the same data PV.
- **Set:** Each set has a dedicated data PV that is shared across only the pods within that set. The data PV is isolated for each set, meaning no two pods from different sets can access the same data PV.
- **Pod:** Each pod has a dedicated data PV. It provides data PV isolation between each pod, regardless of the sets that they belong to.

Offline Mediation Controller Persistent Volume Claim Configuration

Table 4-1 lists the Persistent Volume Claims (PVCs) used by the Offline Mediation Controller server.

Table 4-1 List of PVCs in Offline Mediation Controller Server

PVC Name	Default Pod Internal File System
pvc-vol-install-admin-server	*home*/**ocomcuser/install**
pvc-vol-install-*SET_NAME-SET_ORDINAL_INDEX* For example, pvc-vol-install-nm-cc-0	*home*/**ocomcuser/install**
pvc-vol-keystore	*home*/**ocomcuser/keystore**
pvc-vol-suspense (Optional PVC)	*home*/**ocomcuser/suspense**
• pvc-vol-data (Application scoped) • pvc-*SET_NAME*-vol-data (Set scoped) For example, pvc-nm-cc-vol-data • pvc-*SET_NAME*-vol-data-*SET_NAME-SET_ORDINAL_INDEX* (Pod scoped) For example, pvc-nm-cc-vol-data-nm-cc-0	*home*/**ocomcuser/data**
pvc-vol-external	*home*/**ocomcuser/external**

Table 4-1 (Cont.) List of PVCs in Offline Mediation Controller Server

PVC Name	Default Pod Internal File System
pvc-vol-backup **Note:** The pvc-vol-backup is only created when the **ocomc.storage.backup.enabled** attribute is set to **true**.	*home***/ocomcuser/backup**

To share these PVCs between Offline Mediation Controller pods, you must use a persistent volume provisioner that:

- Provides ReadWriteMany access and sharing between the pods
- Mounts all external volumes with a user (**ocomcuser**) that has a UID and GID of 1000 and that has full permissions
- Has its volume reclaim policy set to avoid data and configuration loss in a mounted file system
- Is configured to share data, external KeyStore volumes, and wallets between Offline Mediation Controller pods and the Administration Client

You must place all CDR files inside the vol-data PVC and then configure the internal file system path of the vol-data PVC in your Administration Client. The **ocomcuser** user must have read and write permissions for all CDRs.

You must place all necessary third-party and cartridge JAR files in *home***/ocomcuser/external/3rd_Party** and *home***/ocomcuser/external/cartridges** directories inside the vol-external PVC, and then restart the pods. After the PVC is mounted, the JAR files are copied to *home***/ocomcuser/install/ocomc/3rd_Party** and *home***/ocomcuser/install/ocomc/cartridges**.

The Offline Mediation Controller wallet files will be created and used through the shared vol-keystore PVC.

The Node Managers can be deployed in the specific Kubernetes node by setting the affinity in the **values.yaml** file.

Configuring Offline Mediation Controller Services

The Offline Mediation Controller unified Helm chart (**oc-cn-ocomc-core-helm-chart**) configures and deploys all of your product services. YAML descriptors in the **oc-cn-ocomc/templates** directory use the **oc-cn-ocomc/values.yaml** file for most of the values. You can override the values by creating an **override-values.yaml** file.

The unified Helm chart includes both Offline Mediation Controller Core and REST Services Manager under a single Helm chart. It contains both Core and REST Services Manager Helm charts as subcharts within it. You can use the following keys to toggle deployment between Offline Mediation Controller Core or REST Services Manager by setting their values to either **true** or **false**:

- Use **charts.enableCore** to enable Offline Mediation Controller Core.
- Use **charts.enableRSM** to enable Offline Mediation Controller REST Services Manager.

Table 4-2 lists the keys that directly impact Offline Mediation Controller services. Add these keys to your **override-values.yaml** file with the same path hierarchy.

> **Note**
>
> - If you are using a Windows-based client, the **adminsvrIp**, **nmExternalPort**, **adminsvrExternalPort**, and **adminsvrFirewallPort** keys must be set. To connect with the Windows-based client, use external services with a NodePort type. In this case, the **adminsvrIp** will be the worker node IP. Restart the pod after setting **adminsvrIp**.
> - If graphical desktop support such as VNC is available on a worker node, the client can be installed on the same worker node in which Administration Server and Node Manager pods are running. In this case, set the service type to ClusterIP and do not set the **nmExternalPort**, **adminsvrExternalPort**, and **adminsvrFirewallPort** keys.

Table 4-2 Offline Mediation Controller Server Keys

Key	Path in values.yaml File	Description
enableCustomizationFileUpload	**global**	Whether to allow custom file uploads during imports (**true**) or not (**false**). The default value is **false**.
enableTestNodeChain	**global**	Whether node chain testing is enabled (**true**) or not (**false**). The default value is **false**.
RSMcontainer.imageRepository	**global**	The repository where the RSM image needs to be pulled from.
RSMcontainer.imagePullPolicy	**global**	The image pull policy for the RSM image.
RSMcontainer.image	**global**	The name of the RSM image.
runMigrationDataJob	**global.statefulSetUpgrade**	Whether to initiate a job for migrating data from an older setup to a new 15.1 setup. The default value is **false**.
payloadFilePath	**global.statefulSetUpgrade**	The path to the payload file used to migrate data from an older setup to a new 15.1 setup.
restartCount	**ocomcCore.ocomc**	Increment the current value by 1 to trigger a restart of all Offline Mediation Controller components. The starting value is **0**.
sslEnabled	**ocomcCore.ocomc**	Whether to enable SSL for secure communication between components (**true**) or not (**false**). The default value is **true**.
forceGenSslcert	**ocomcCore.ocomc**	Whether to regenerate the SSL certificates for the Administration Server and Node Manager (**true**) or not (**false**). The default value is **false**.
upgradeEnabled	**ocomcCore.ocomc**	Set to **true** when using a new version of the Offline Mediation Controller image with an existing installation to trigger the upgrade process. The default value is **false**.
rsmURL	**ocomcCore.ocomc**	The URL of the Offline Mediation Controller REST Services Manager for integration. The default values is **http://ocomc-rsm:8080**.

Table 4-2 (Cont.) Offline Mediation Controller Server Keys

Key	Path in values.yaml File	Description
cartrdigeFolder	**ocomcCore.ocomc**	The directory where Offline Mediation Controller cartridge packs are installed. Set this key to **/home/ocomcuser/ext/cartridges** unless you are creating custom images.
storageClass	**ocomcCore.ocomc.storage**	The Kubernetes storage class for persistent volumes.
keytore.name	**ocomcCore.ocomc.storage**	The name of the KeyStore volume used for storing sensitive credentials. The default value is **keystore-vol**.
external.name	**ocomcCore.ocomc.storage**	The name of the external volume used for additional storage. The default value is **external-vol**.
external.capacity	**ocomcCore.ocomc.storage**	The capacity of the external volume.
backup.enabled	**ocomcCore.ocomc.storage**	Whether to create a backup PV (**true**) or not (**false**).
backup.name	**ocomc.Core.ocomc.storage**	The name of the backup PV.
backup.accessModes	**ocomcCore.ocomc.storage**	The permission access mode of the backup PV. The default value is **ReadWriteMany**.
backup.capacity	**ocomcCore.ocomc.storage**	The capacity of the backup PV.
fsGroup	**ocomcCore.ocomc.securityContext**	The file system group ID for security contexts.
runAsUser	**ocomcCore.ocomc.securityContext**	The user ID under which the process runs.
runAsGroup	**ocomcCore.ocomc.securityContext**	The group ID under which the process runs.
enabled	**ocomcCore.ocomc.authentication**	Whether to enable authentication for accessing system resources (**true**) or not (**false**).
uniPass	**ocomcCore.ocomc.secrets**	Use this key to apply a uniform password to all Offline Mediation Controller cloud native services, including: • Database Schemas • Offline Mediation Controller Root Login • Oracle Wallets To override this password for a specific service, specify a different password in the service's key. **Note:** Use this key for test or demonstration systems only.
walletPass	**ocomcCore.ocomc.secrets**	The string password for opening the wallet.
nmKeypass	**ocomcCore.ocomc.secrets**	The password for the Node Manager domain SSL identity key.
nmKeystorepass	**ocomcCore.ocomc.secrets**	The Offline Mediation Controller Secrets required for SSL and installation.

Table 4-2 (Cont.) Offline Mediation Controller Server Keys

Key	Path in values.yaml File	Description
adminKeypass	**ocomcCore.ocomc.secrets**	The password for the Administration Server domain SSL Identity Key.
adminKeystorepass	**ocomcCore.ocomc.secrets**	The password for the Administration Server domain SSL Identity Store.
rsmOAuthToken	**ocomcCore.ocomc.secrets**	The access token used by Administration Server to communicate with the REST Services Manager when it is running with security enabled.
image.pullPolicy	**ocomcCore.ocomc.adminServerConfigurations**	The pull policy of the Administration Server container image.
image.pullSecret	**ocomcCore.ocomc.adminServerConfigurations**	The location of your imagePullSecrets, which stores the credentials (or Secret) for accessing your private Docker registry.
image.repository	**ocomcCore.ocomc.adminServerConfigurations**	The repository location for the Administration Server container image.
image.name	**ocomcCore.ocomc.adminServerConfigurations**	The name of your Administration Server container image.
restartCount	**ocomcCore.ocomc.adminServerConfigurations**	Increment the current value by 1 to trigger a restart of the Administration Server. The starting value is **0**.
log.level	**ocomcCore.ocomc.adminServerConfigurations**	The logging level for the Administration Server. There are three possible levels: • **INFO** • **DEBUG** • **WARN** The default value is **INFO**.
log.pattern	**ocomcCore.ocomc.adminServerConfigurations**	The pattern in which log messages are generated.
clientTimeout	**ocomcCore.ocomc.adminServerConfigurations**	The time to wait for Kubernetes commands to complete.
type	**ocomcCore.ocomc.adminServerConfigurations.service**	The service type: **ClusterIP**, **NodePort**, or **LoadBalancer**.
appPort	**ocomcCore.ocomc.adminServerConfigurations.service**	The application port for the Administration Server.
firewallPort	**ocomcCore.ocomc.adminServerConfigurations.service**	The firewall port for the Administration Server.
callbackPort	**ocomcCore.ocomc.adminServerConfigurations.service**	The callback port for the Administration Server.
name	**ocomcCore.ocomc.adminServerConfigurations.storage.install**	The name of the install volume used for the Administration Server installation.

Table 4-2 (Cont.) Offline Mediation Controller Server Keys

Key	Path in values.yaml File	Description
capacity	**ocomcCore.ocomc.adminServerConfigurations.storage.install**	The storage capacity allocated for the Administration Server install volume, such as 1Gi.
enabled	**ocomcCore.ocomc.adminServerConfigurations.import**	Whether to enable the feature that triggers import on initial setup of Offline Mediation Controller through the REST Services Manager.
import.mappingFile	**ocomcCore.ocomc.adminServerConfigurations.import**	The path to the mapping file for import, if enabled.
gcOptions	**ocomcCore.ocomc.adminServerConfigurations**	The garbage control (GC) options for the Administration server.
memoryOptions	**ocomcCore.ocomc.adminServerConfigurations**	The memory-related options to pass to the Administration Server process.
eceIntegration.*	**ocomcCore.ocomc.nodeManagerConfiguartions**	The details for connecting to ECE. Add these keys only if you are integrating Offline Mediation Controller with ECE: • **enabled**: Specifies that integration with ECE is enabled. • **image.repository**: The Docker registry URL for the ECE image. • **image.name**: The name of the ECE image. • **image.pull.Policy**: The pull policy of the ECE image. The default value is IfNotPresent, which specifies not to pull the image if it's already present. Applicable values are IfNotPresent and Always. • **clusterName**: The ECE cluster name. The default is BRM. • **persistenceEnabled**: Whether ECE will persist its cache data in the Oracle database: true or false. The default is false. • **coherenceClusterPort**: The value indicating the Coherence port used by the ECE component.
data.name	**ocomcCore.ocomc.nodeManagerConfigurations.storage**	The name of the volume for data storage.
data.accessModes	**ocomcCore.ocomc.nodeManagerConfigurations.storage**	The permission access mode of the data PV.
data.capacity	**ocomcCore.ocomc.nodeManagerConfigurations.storage**	The capacity of the volume.

Table 4-2 (Cont.) Offline Mediation Controller Server Keys

Key	Path in values.yaml File	Description
data.scope	**ocomcCore.ocomc.nodeManagerConfigurations.storage**	The scope of the volume. Possible values are: • **Application**: Only one data PV would be deployed and would be shared by all the Node Manager Pod. • **Set**: Each Node Manager set would have their dedicated PV (all the pods in the set would share the same data PV). • **Pod**: Each Node Manager pod will get a dedicated data PV.
replication.enabled	**ocomcCore.ocomc.nodeManagerConfigurations.scaling**	Whether to enable auto-replication of Node Manager pods upon scaling (**true**) or not (**false**).
createServiceAccount	**ocomcCore.ocomc.nodeManagerConfigurations.scaling.hpa.serviceAccount**	Whether to create a service account (**true**) or not (**false**).
serviceAccount.name	**ocomcCore.ocomc.nodeManagerConfigurations.scaling.hpa**	The service account to be used by Offline Mediation Controller. If the service account does not exist, set the **createServiceAccount** key to **true**.
serviceAccount.enabled	**ocomcCore.ocomc.nodeManagerConfigurations.scaling.hpa**	Whether to enable the Kubernetes Horizontal Pod Autoscaler (HPA) for dynamic scaling of the Node Manager.
hpaScaleDownEnabled	**ocomcCore.ocomc.nodeManagerConfigurations.scaling.hpa**	Whether to allow HPA to scale down pods when the relevant metrics fall below the specified threshold (**true**) or not (**false**).
restartCount	**ocomcCore.ocomc.nodeManagerConfigurations**	Increment the current value by 1 to trigger a restart of all Node Manager components. The starting value is **0**.
level	**ocomcCore.ocomc.nodeManagerConfigurations.log**	The logging level for Node Managers. There are three possible levels: • **INFO** • **DEBUG** • **WARN** The default value is **INFO**.
jmxEnabled	**ocomcCore.ocomc.nodeManagerConfigurations**	Whether to enable JMX monitoring for Node Manager diagnostics (**true**) or not (**false**).
jmxPort	**ocomcCore.ocomc.nodeManagerConfigurations**	The port used for JMX monitoring connections.
cpu	**ocomcCore.ocomc.nodeManagerConfigurations.resources.requests**	The minimum CPU resources allocated for Node Manager pods.
memory	**ocomcCore.ocomc.nodeManagerConfigurations.resources.requests**	The minimum memory allocated for Node Manager pods.

Table 4-2 (Cont.) Offline Mediation Controller Server Keys

Key	Path in values.yaml File	Description
cpu	**ocomcCore.ocomc.nodeManagerConfigurations.resources.limits**	The maximum CPU resources for Node Manager pods.
memory	**ocomcCore.ocomc.nodeManagerConfigurations.resources.limits**	The maximum memory limit for Node Manager pods.
serviceMonitor.enabled	**ocomcCore.ocomc.nodeManagerConfigurations**	Enable or disable service monitor being deployed.
serviceMonitor.interval	**ocomcCore.ocomc.nodeManagerConfigurations**	The interval for service monitoring scraping.
serviceMonitor.labels.app	**ocomcCore.ocomc.nodeManagerConfigurations**	The app label to be added for service monitor.
serviceMonitor.labels.release	**ocomcCore.ocomc.nodeManagerConfigurations**	The release label to be added for service monitor.
metrics.enabled	**ocomcCore.ocomc.nodeManagerConfigurations**	Enable or disable metrics.
suspenseManagementIntegration*	**ocomcCore.ocomc.nodemanagerConfigurations**	The details for integrating to suspense management. Add these keys only if you are integrating Offline Mediation Controller with suspense management: • **enabled**: Whether to enable or disable suspense management integration. • **createPV**: Whether to enable or disable PV creation for Suspense Management. This determines if Offline Mediation Controller should use an existing shared suspense PV. • **storage.suspense.name**: The name of the volume for suspense storage. • **storage.suspense.accessModes**: The access modes for the suspense storage volume. • **storage.suspense.capacity**: The storage capacity allocated for the suspense volume.
external.name	**ocomcCore.ocomc.nodeManagerConfigurations.storage**	The name of the volume for external storage.
external.accessModes	**ocomcCore.ocomc.nodeManagerConfigurations.storage**	The access modes for the external storage.
external.capacity	**ocomcCore.ocomc.nodeManagerConfigurations.storage**	The storage capacity allocated for the external volume.
jvmOpts	**ocomcCore.ocomc.nodeManagerConfigurations**	The JVM options for the Node Manager.

Table 4-2 (Cont.) Offline Mediation Controller Server Keys

Key	Path in values.yaml File	Description
affinity	**ocomcCore.ocomc.nodeManagerConfigurations**	The Node Manager affinity rules for pod scheduling.
sets	**ocomcCore.ocomc.nodeManagerConfigurations**	The various Node Manager sets to be deployed. Each set would have a dedicated StatefulSet.
type	**ocomcCore.ocomc.nodeManagerConfigurations.service**	The type of Kubernetes service used to expose the Node Manager.
port	**ocomcCore.ocomc.nodeManagerConfigurations.service**	The port number exposed by the Node Manager service inside the cluster.
nodePort	**ocomcCore.ocomc.nodeManagerConfigurations.service**	The NodePort value for exposing the Node Manager service externally. Applies only if **service.type** is set to **NodePort**.
rdm.threadCount	**ocomcCore.ocomc.nodeManagerConfigurations.service**	The number of RDM threads for the Node Manager.

Deploying Offline Mediation Controller Services

To deploy Offline Mediation Controller services on your cloud native environment, do this:

> **Note**
>
> To integrate the Offline Mediation Controller cloud native deployment with the ECE and BRM cloud native deployments, they must use the same namespace.

1. Validate the content of your charts by entering this command from the **helmcharts** directory:

   ```
   helm lint --strict oc-cn-ocomc-core-helm-chart
   ```

   You'll see this if the command completes successfully:

   ```
   1 chart(s) linted, no failures
   ```

2. Run the **helm install** command from the **helmcharts** directory:

   ```
   helm install ReleaseName oc-cn-ocomc-core-helm-chart --namespace NameSpace
   --values OverrideValuesFile
   ```

   where:

   - *ReleaseName* is the release name, which is used to track this installation instance.

- *NameSpace* is the namespace in which to create Offline Mediation Controller Kubernetes objects. To integrate the Offline Mediation Controller cloud native deployment with the ECE and BRM cloud native deployments, they must use the same namespace.
- *OverrideValuesFile* is the path to a YAML file that overrides the default configurations in the chart's **values.yaml** file.

For example, if the **override-values.yaml** file is in the **helmcharts** directory, the command for installing Offline Mediation Controller cloud native services would be:

```
helm install ocomc oc-cn-ocomc-core-helm-chart --namespace ocgbu --values
override-values.yaml
```

Installing the Offline Mediation Controller Web-Based UI

Offline Mediation Designer is a web-based UI that runs on top of Offline Mediation Controller. You can use it to create, design, and manage nodes, node chains, and Node Managers within mediation processes.

Prerequisites

Before deploying the Offline Mediation Designer UI, you must first install the following software.

About Installing an Ingress Controller

You use ingress controllers to expose services outside the Kubernetes cluster, enabling clients to communicate with Offline Mediation Controller cloud native. Ingress controllers route external traffic to services within the Kubernetes cluster using the rules you define.

The Offline Mediation Controller cloud native deployment package includes a sample NGINX Ingress Controller (**oc-cn-ocomc-nginx-ingress-controller-sample-helm-chart-15.1.0.*x*.0.tgz**) that you can install and configure for the Offline Mediation Designer UI. The archive file includes a Helm chart and a README file explaining how to configure the NGINX Controller for your system.

For information about NGINX Ingress Controller, see the NGINX documentation: https://docs.nginx.com/nginx-ingress-controller/.

About Installing the Relying Party

Relying Party applications authenticate users by working with a trusted Identity Provider, such as Oracle Identity Cloud Service (IDCS). The relying party delegates user authentication to the identity provider, which can be an OpenID Connect provider, a Security Assertion Markup Language (SAML) identity provider, or any other authentication service.

The Offline Mediation Controller cloud native deployment package includes a sample Apache Relying Party (**oc-cn-ocomc-apache-relying-party-sample-helm-chart-15.1.0.*x*.0.tgz**) that you can install and configure for the Offline Mediation Designer UI. The archive file includes a Helm chart and a README file explaining how to configure the software for your system.

> **Note**
>
> When integrating Apache as a Relying Party using **mod_auth_openidc**, Oracle recommends the following session management settings to align with your Identity Provider (IdP) policies:
>
> 1. Set **OIDCSessionMaxDuration** to **0** to defer the session's maximum duration to the ID token expiry, making the IdP the authority.
> 2. Set **OIDCSessionInactivityTimeout** to a value slightly lower (for example, 10–30 seconds less) than your IdP's inactivity timeout for a seamless user experience. If your IdP does not enforce an inactivity timeout, choose an appropriate value based on your security needs (typical values are 900 to 3600 seconds).

About the Offline Mediation Designer UI Helm Chart

The Offline Mediation Controller cloud native deployment package includes the **oc-cn-ocomc-mediation-ui-helm-chart-15.1.0.*x*.0.tgz** file. It is a Helm chart archive used for deploying the Offline Mediation Designer UI on a Kubernetes cluster. Extract the Helm chart and files from the archive by entering this command:

```
tar zxvf oc-cn-ocomc-mediation-ui-helm-chart-15.1.0.x.0.tgz
```

The following files and directories are extracted:

```
profiles/
profiles/client-side-auth-idcs.yaml
profiles/client-side-auth-oam.yaml
profiles/deploy-oci.yaml
profiles/relying-party.yaml
mediation-ui-charts.tgz
```

The **profiles** directory contains these sample YAML files that you can copy and modify to meet your configuration requirements:

- **replying-party.yaml:** Use this file for deploying the Offline Mediation Designer UI with client-side authentication disabled, meaning that the UI sits behind a relying party.
- **client-side-auth-idcs.yaml:** Use this file as a reference for deploying the Offline Mediation Designer UI with client-side authorization enabled and the API secured by IDCS.
- **client-side-auth-oam.yaml:** Use this file as a reference for deploying the Offline Mediation Designer UI with client-side authorization enabled and the API secured by Oracle Access Management.

Table 4-3 lists the keys that impact Offline Mediation Designer UI referenced in the above YAML files.

Table 4-3 List of UI keys

Key	Description
security.clientSideAuthEnabled	Controls whether client-side authentication is enabled (**true**) or not (**false**). Set it to **false** if the Offline Mediation Designer UI is deployed in conjunction with a relying party. **Note:** When set to **false**, it is not necessary to set the **authorizationUrl**, **authorizationEndpoint**, **clientId**, **scope**, **redirectUri**, and **postLogoutRedirectUri** keys. This configuration is instead managed within the relying party service.
security.authorizationURL	(Only used when **security.clientSideAuthEnabled** is set to true) The URL of the IdP (Identity Provider). Different IdPs have different values for the URL: • For Oracle Access Management: **https://***OAMHostname***:***Port***/oauth2/rest** • For IDCS: **https://***IDCSidentifier***/identity.oraclecloud.com/oauth2/v1** • For other IdPs: **http://***hostname***:***port***/realms/***Realm***/protocol/openid-connect**
security.authorizatonEndpoint	(Only used when **security.clientSideAuthEnabled** is set to true) The name of the endpoint for initiating the authorization flow, which is added to the URL specified in **authorizationURL**. For Oracle Access Management and IDCS, the value is **authorized**. Other IDPs may have different values, such as **auth**. During the authorization flow, a POST call is made to **https://** *IDCSidentifier***.identity.oraclecloud.com/oauth2/v1/authorize** for IDCS and **https://** *OAMHostname***:***Port***/oauth2/rest /authorize** for Oracle Access Management.
security.logoutEndpoint	The value for the logout endpoint to initiate the logout process. Typically, this is the user logout endpoint configured in the IDP.
security.clientId	(Only used when **security.clientSideAuthEnabled** is set to true) The unique identifier of the client application requesting authorization. This must match the value of the client created in the IDP.
security.scope	(Only used when **security.clientSideAuthEnabled** is set to true) The permissions being requested.
security.redirectUri	(Only used when **security.clientSideAuthEnabled** is set to true) The URI where the user is redirected after authorization. **Note:** The **redirectUri** key must match one of the values for the redirectURIs in the client created in the IDP. Typically, this is the URL of the mediation UI.
security.postLogoutRedirectUri	(Only used when **security.clientSideAuthEnabled** is set to true) The URI where the user is redirected after log out.
security.mediationUri	The URL of the Offline Mediation Controller API service. This is the URL that the UI will use to make call to the API so it must be accessible through the browser. Typically, this should point to the ingress controller URL as all calls from the UI should be made through the ingress controller and get forwarded accordingly.
security.domain	(Required when using IDCS, optional for OAM) Specifies the domain used for authentication. Set to the appropriate IDCS domain when integrating with IDCS. For OAM, this key may be omitted unless your configuration requires it.

Before proceeding to deploy the web-based UI, you must do the following steps:

1. Make a copy of the appropriate YAML file you wish to use and update it according to your configuration requirements. For example, if you want to use the **relying-party.yaml** file, run the following command:

   ```
   cp profiles/relying-party.yaml my-custom-profile.yaml
   ```

2. Make a copy of the deployment configuration file using the following command:

   cp *profiles*/**deploy-oci.yaml deploy-mediation-ui.yaml**

3. If you are using a private registry, update the **deploy-mediation-ui.yaml** file with image registry and secret details. For example:

```
image:
  repository: my-docker-registry

imagePullSecret:
  imagePullSecrets:
    - name: my-docker-secret

service:
  type: NodePort
  nodePort: 31503
```

Deploying the Offline Mediation Designer UI

To deploy the Offline Mediation Designer UI in your cloud native environment, do the following:

1. Validate the content of your charts by entering this command from the **helmcharts** directory:

   helm lint --strict oc-cn-ocomc

2. Run the **helm install** command from the **helmcharts** directory:

   helm -n *namespace* **install mediation-ui mediation-ui-charts.tgz -f deploy-mediation-ui.yaml -f mediation-ui-values.yaml**

   where *namespace* is the namespace in which to create the Offline Mediation Controller Kubernetes objects.

Afterward, you can access the Offline Mediation Designer UI at the following URL:

https://*hostname*/**webApps/mediation/**

where *hostname* is the host name of the configured ingress controller deployment.

5

About Integrating Offline Mediation Controller REST Services Manager with Cloud Native

You can integrate an external application with Oracle Communications Offline Mediation Controller cloud native by using Offline Mediation Controller REST Services Manager.

Topics in this document:

About Offline Mediation Controller REST Services Manager

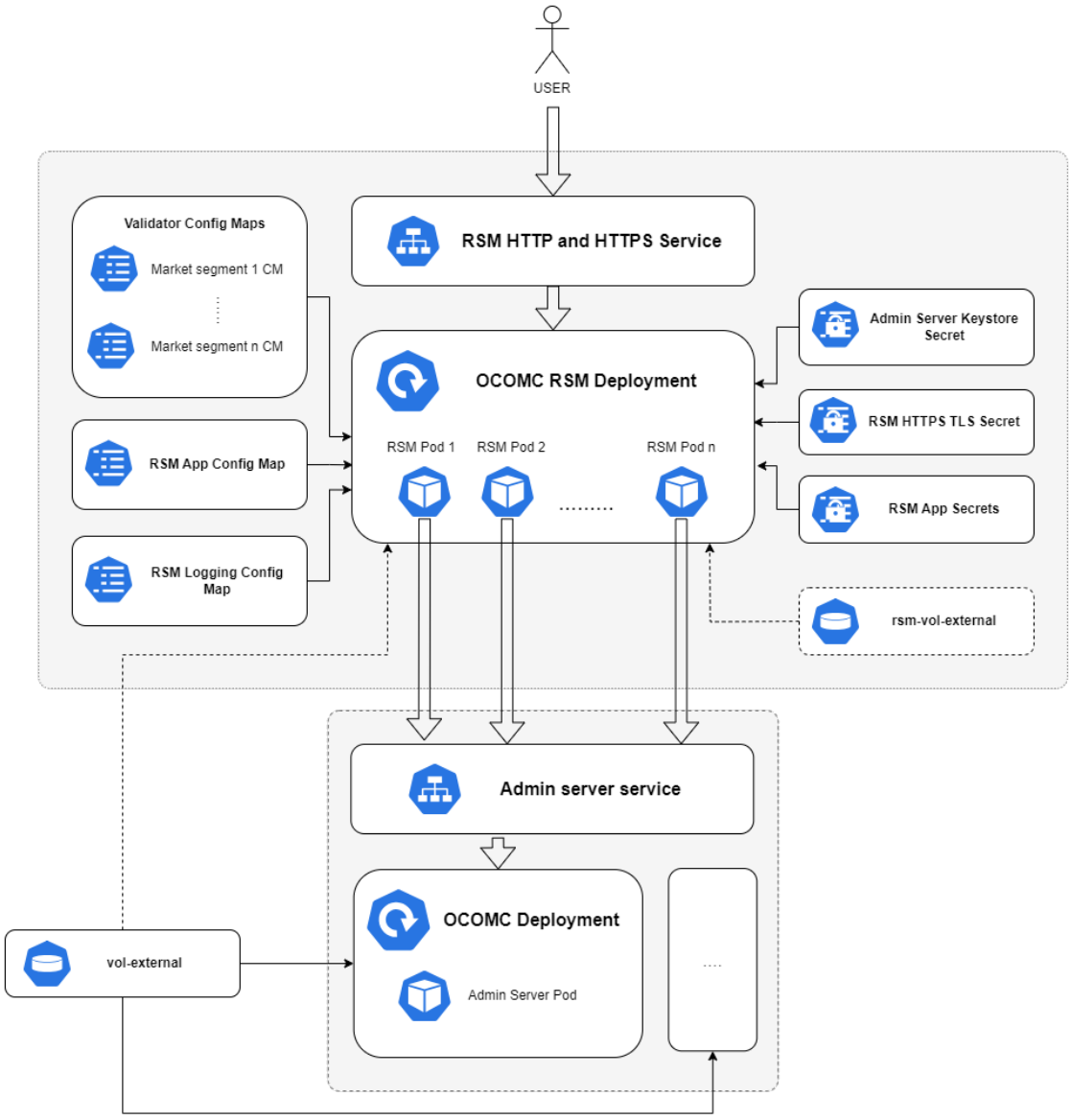
The Offline Mediation Controller REST Services Manager allows you to perform the same operations as the **NMShell** application using external client applications. For example, it allows your external application to do the following in Offline Mediation Controller:

- Manage Nodes
- Manage Node Managers
- Retrieve a list of node chains
- Compile and save the NPL rules file
- Export node configurations and customizations

About Offline Mediation Controller REST Services Manager Cloud Native Architecture

Figure 5-1 shows all the components of the Offline Mediation Controller REST Services Manager cloud native architecture.

Figure 5-1 Offline Mediation Controller REST Services Manager Cloud Native Architecture



The components in this figure include:

- **REST Services Manager Deployment**: The primary deployment of REST Services Manager with all the necessary components and configurations.
- **REST Services Manager HTTP and HTTPS Service**: This service exposes REST Services Manager to ports, allowing access to REST Services Manager through HTTP and HTTPS protocols.
- **Validator ConfigMap**: There is a unique ConfigMap for each market segment.
- **REST Services Manager App ConfigMap**: The ConfigMap contains the **application.yaml** file, which holds the configurations required to initiate the REST Services Manager server.

- **REST Services Manager Logging ConfigMap**: The ConfigMap holds the **log4j2.yaml** file, encompassing logging-related configurations.
- **Admin Server Keystore Secret**: This secret contains the administration server KeyStore file in a Base64-encoded format.
- **RSM HTTPS TLS Secret**: This secret contains the HTTPS TLS store utilized by REST Services Manager when the HTTPS protocol is enabled.
- **REST Services Manager App Secret**: This secret contains all confidential information necessary to launch the REST Services Manager server.
- **vol-external**: This is an optional PV reference. REST Services Manager will incorporate it only if the flag **rsm.pvc.ocomcExternal.enabled** in the **override-values.yaml** file is set to **true**. When enabled, the REST Services Manager will share the **vol-external** PV of the OCOMC core deployment. It is mandatory to enable this flag if the node chain solution includes cartridges containing sensitive information such as FTP or database passwords.
- **rsm-vol-external**: This PV is optional and can be enabled by setting the flag **rsm.pvc.external.enabled** to **true** in the **values.yaml** file. When enabled, the REST Services Manager will load custom cartridges from the specified PV into the classpath. The source directory for it can be configured in the **override-values.yaml** file.

Installing Offline Mediation Controller REST Services Manager

The Offline Mediation Controller REST Services Manager can be installed along with core Offline Mediation Controller components using a unified Helm chart.

To install Offline Mediation Controller REST Services:

1. Configure and install all required third-party software. See "Setting Up Prerequisite Software".
2. Configure the Offline Mediation Controller server and REST Services Manager connection. See "Configuring the Offline Mediation Controller Core and REST Services Manager Connection".
3. Configure the REST Services Manager server. See "Configuring the REST Services Manager Server".
4. Load custom validators. See "Configuring and Loading Custom Validators".
5. Deploy Offline Mediation Controller REST Services Manager. See "Deploying Offline Mediation Controller Services".

Setting Up Prerequisite Software

As part of preparing your environment for Offline Mediation Controller REST Services Manager, you install and set up various components and services in ways that are best suited for your cloud native environment. The following shows the high-level prerequisite tasks for deploying Offline Mediation Controller REST Services Manager:

1. Ensure that you have downloaded the latest software that is compatible with Offline Mediation Controller cloud native. See "Offline Mediation Controller Cloud Native System Requirements" in *Offline Mediation Controller Compatibility Matrix.*
2. Ensure that your environment setup is complete. See "Setting Up Your Environment".
3. Download the Offline Mediation Controller cloud native Helm chart. See "Downloading Packages for the Offline Mediation Controller Cloud Native Helm Charts".

Configuring the Offline Mediation Controller Core and REST Services Manager Connection

To configure the Offline Mediation Controller core and REST Services Manager connection:

1. In your **override-values.yaml** file for **oc-cn-ocomc-helm-chart**, set the following keys:
   - **ocomcRSM.rsm.adminServerConnection.hostname**: Specify the hostname where the Offline Mediation Controller Admin Server is running.
   - **ocomcRSM.rsm.adminServerConnection.port**: Specify the port where Offline Mediation Controller Admin Server listens.
2. If Offline Mediation Controller core uses SSL, do the following:
   a. Copy your **adminClientTruststore.jks** file from the **vol-keystore PV** of Offline Mediation Controller core to the **oc-cn-ocomc/rsm/ocomc-rsm-keystore** directory.
   b. In your **override-values.yaml** file, set the following keys:
      - **ocomcRSM.rsm.adminServerConnection.ssl.enabled**: Set this key to **true**. This enables SSL between REST Services Manager and the Admin Server.
      - **ocomcRSM.rsm.adminServerConnection.ssl.keystoreName**: Specify the name of your KeyStore file, such as **adminClientTruststore.jks**.
3. If authentication is enabled for Offline Mediation Controller core, set the following keys in your **override-values.yaml** file:
   - **ocomcRSM.rsm.adminServerConnection.username**: Specify the user name for logging in to the Admin Server.
   - **ocomcRSM.rsm.adminServerConnection.password:** Specify the password for logging in to the Admin Server.

Configuring the REST Services Manager Server

To configure the Offline Mediation Controller REST Services Manager Server:

1. Enable HTTPS in REST Services Manager by doing the following:
   a. Copy your generated **.p12** KeyStore file to the REST Services Manager Helm chart directory (**oc-cn-ocomc/charts/oc-cn-ocomc-rsm/ocomc-rsm-keystore**).
   b. Set the following keys in your **override-values.yaml** file for **oc-cn-ocomc:**
      - **ocomcRSM.rsm.https.enabled**: Set this to **true**.
      - **ocomcRSM.rsm.https.ketsotreName**: Specify the name of the KeyStore file with the extension.
      - **ocomcRSM.rsm.https.keystorePassPhrase**: Specify the KeyStore passphrase.
2. Expose REST Services Manager through a NodePort by setting the following keys in your **override-values.yaml** file:
   - **ocomcRSM.rsm.service.type**: Set this to **NodePort**.
   - **ocomcRSM.rsm.service.nodePort**: Specify the port number.
   - **ocomcRSM.rsm.https.service.nodePort**: If the HTTPS port is enabled, specify the port for exposing the HTTPS port outside the cluster.

3. Enable Oracle Access Management authentication by setting the following keys in your **override-values.yaml** file:
   a. **ocomcRSM.rsm.security.provider**: Set this to **OAM**.
   b. **ocomcRSM.rsm.security.configuration.oam**: Fill in the Oracle Access Management and Oracle Unified Directory configuration details.
4. Set the log levels to the appropriate level in the **ocomcRSM.rsm.logging.packagingLogging** keys in your **override-values.yaml** file.

Configuring and Loading Custom Validators

In Offline Mediation Controller REST Services Manager, you can configure custom validators.

To load custom validators:

1. Enable custom validators for Offline Mediation Controller RSM. In your **override-values.yaml** file for **oc-cn-ocomc-helm-chart**, set the **ocomcRSM.rsm.customisation.nodeConfigValidator.validators.enabled** key to **true**.
2. Create a subdirectory within the RSM Helm chart directory (**oc-cn-ocomc-rsm/ocomc-rsm-validator**) with the name of the market segment for the validator. For example, create a directory named **oc-cn-ocomc-rsm/ocomc-rsm-validator/my-market**.
3. Copy the validator YAML files into the directory created in the previous step.
4. In your **override-values.yaml** file, set the **ocomcRSM.rsm.customisation.nodeConfigValidator.validators.marketSegments** key to a list of supported market segments.

About the Offline Mediation Controller REST Services Manager Keys

Table 5-1 lists the keys that directly impact Offline Mediation Controller REST Services Manager. Add these keys to your **override-values.yaml** file with the same path hierarchy.

Table 5-1 Offline Mediation Controller REST Services Manager Keys

Key	Path in values.yaml file	Description
imagePullSecrets	-	The location of your **imagePullSecrets**, which stores the credentials (or Secret) for accessing your private Docker registry.
name	**ocomcRSM.rsm**	The name to use for the deployment. The final name of the deployment is derived using the name provided.
fullname	**ocomcRSM.rsm**	The final name of the deployment to use. This would be used for the deployment without any modification.
replicas	**ocomcRSM.rsm**	The total number of REST Services Manager pods to run in the deployment.
restartCount	**ocomcRSM.rsm**	Tracks the number of restarts. To restart the pods, increment the value by 1 and run the **helm upgrade** command.
serviceMonitor.enabled	**ocomcRSM.rsm**	Whether to enable the service monitor for REST Services Manager metrics.

Table 5-1 (Cont.) Offline Mediation Controller REST Services Manager Keys

Key	Path in values.yaml file	Description
imageRepository	**ocomcRSM.rsm.container**	The repository from where the REST Services Manager image can be pulled. **Note:** The repository URI should not end with a trailing slash.
imagePullPolicy	**ocomcRSM.rsm.container**	The image pull policy to use for the deployment. The default value is **IfNotPresent**, which specifies not to pull the image if it's already present. Applicable values are **IfNotPresent** and **Always**.
image	**ocomcRSM.rsm.container**	The REST Services Manager image name and tag concatenated with a colon (**:**). Ensure to align with the REST Services Manager image version to be deployed.
enabled	**ocomcRSM.rsm.https**	Whether REST Services Manager should run with HTTPS.
keystoreName	**ocomcRSM.rsm.https**	The KeyStore file name with its extension to use for HTTPS. The file must be present in the **oc-cn-ocomc-rsm/ocomc-rsm-keystore** directory.
keystorePassPhrase	**ocomcRSM.rsm.https**	The passphrase for the HTTPS KeyStore file.
extRsmKeystoreSecret	**ocomcRSM.rsm.https**	The external KeyStore Secret name.
service.nodePort	**ocomcRSM.rsm.https**	The node port to use for HTTPS service. This would be used when the service type of REST Services Manager is set to NodePort.
hostname	**ocomcRSM.rsm.adminServerConnection**	The host name for accessing the Administration Server.
port	**ocomcRSM.rsm.adminServerConnection**	The port at which the Administration Server is listening on.
username	**ocomcRSM.rsm.adminServerConnection**	The user name to use for logging into the Administration Server.
password	**ocomcRSM.rsm.adminServerConnection**	The password for the specified user to use during login.
ocomcExternal.enabled	**ocomcRSM.rsm.pvc**	Whether REST Services Manager shares the same external PV of the Offline Mediation Controller core. Enabling this is mandatory when REST Services Manager is involved in creating node chain solutions involving cartridges with sensitive password information (FTP or database passwords). The mount path is **/app/volumes/ocomc-ext**.
ocomcExternal.name	**ocomcRSM.rsm.pvc**	The name of the external volume in Offline Mediation Controller Core.
external.enabled	**ocomcRSM.rsm.pvc**	Whether to create an external PV for REST Services Manager. The mount path is **/app/volumes/ext**.
name	**ocomcRSM.rsm.storageClass**	The storage class to use if REST Services Manager's external PV is enabled.
cartridgeFolder	**ocomcRSM.rsm.configEnv**	The directory path where REST Services Manager retrieves and loads cartridges from.
nodeTypeMapper.enabled	**ocomcRSM.rsm.customisation**	Whether to load custom nodeMappers into REST Services Manager. The content of the file needs to be added to **oc-cn-ocomc-rsm/templates/configmap-nodetypemapper.yaml**.
nodeConfigValidator.validators.enabled	**ocomcRSM.rsm.customisation**	Whether to load custom validators into REST Services Manager.

Table 5-1 (Cont.) Offline Mediation Controller REST Services Manager Keys

Key	Path in values.yaml file	Description
nodeTypeMetadata.enabled	**ocomcRSM.rsm.customisation**	Whether to load custom node type metadata files into REST Services Manager.
transformers.enabled	**ocomcRSM.rsm.customisation.nodeConfigTransformer**	Whether to enable custom transformers.
requestAutomation.enabled	**ocomcRSM.rsm.jobs**	Whether to enable a request automation job (**true**) or not (**false**).
requestAutomation.resources.limits.cpu	**ocomcRSM.rsm.cpu**	The CPU limit for job replicas.
requestAutomation.resources.limits.memory	**ocomcRSM.rsm.jobs**	The memory limit for job replicas.
requestAutomation.resources.requests.memory	**ocomcRSM.rsm.jobs**	The memory limit for job replicas.
requestAutomation.resources.requests.memory	**ocomcRSM.rsm.jobs**	The minimum memory for job replicas.
service.type	**ocomcRSM.rsm.service**	The Kubernetes service type to use.
nodePort	**ocomcRSM.rsm.service**	The NodePort that REST Services Manager should be exposed to if service type is set to **NodePort**.
limits.cpu	**ocomcRSM.rsm.resources**	The CPU limit for REST Services Manager pods.
limits.memory	**ocomcRSM.rsm.resources**	The memory limit for REST Services Manager pods.
requests.cpu	**ocomcRSM.rsm.resources**	The minimum CPU for REST Services Manager pods.
requests.memory	**ocomcRSM.rsm.resources**	The minimum memory for REST Services Manager pods.
rsmTrustStore.enabled	**ocomcRSM.rsm.rsmTrustStore**	Whether to enable a custom TrustStore for SSL/TLS.
trustStoreName	**ocomcRSM.rsm.rsmTrustStore**	The TrustStore file name.
extRSMTruststoreSecret	**ocomcRSM.rsm.rsmTrustStore**	The external TrustStore Secret name.
trustStorePassPhrase	**ocomcRSM.rsm.rsmTrustStore**	The passphrase for the TrustStore.
provider	**ocomcRSM.rsm.security**	The security provider for user authentication.
jvmOpts	**ocomcRSM.rsm**	The required JVM configuration for REST Services Manager.
terminationGracePeriodSeconds	**ocomcRSM.rsm**	The termination grace period for the pod. This is optional.
format.type	**ocomcRSM.rsm.logging**	The logging layout to use. The value should be a supported log4j logging layout.
format.pattern	**ocomcRSM.rsm.logging**	The logging pattern to use.
rootLevel	**ocomcRSM.rsm.logging**	The REST Services Manager's root logging level.
packageLogging	**ocomcRSM.rsm.logging**	The logging levels specific to individual packages.

6

Offline Mediation Controller REST Services Manager Security

Learn how to implement the security capabilities supported by the Oracle Communications Offline Mediation Controller REST Services Manager. Offline Mediation Controller REST Services Manager supports stringent authorization and authentication requirements.

Topics in the document:

About Authentication and Authorization

Offline Mediation Controller REST Services Manager uses the OAuth 2.0 protocol to authenticate a client application's identity and to authorize the client application to access its REST API. It does this by validating an OAuth access token that is passed in the header of the client's HTTP/HTTPS request to the Offline Mediation Controller REST Services Manager. See *REST API Reference for Offline Mediation Controller* for more information.

Your client must pass this OAuth access token in the header of every HTTP/HTTPS request sent to Offline Mediation Controller REST Services Manager. To set up authentication and authorization for your client, you can use either Oracle Identity Cloud Service or Oracle Access Management.

Setting Up OAuth Using Oracle Identity Cloud Service

You can set up your client application to use OAuth authentication at either the user or the application level to access the Offline Mediation Controller REST Services Manager API. For a typical Offline Mediation Controller setup, you create several integrated applications in Oracle Identity Cloud Service (IDCS).

To set up OAuth authentication using IDCS, perform the following steps:

1. Creating a Confidential OAuth Application
2. Creating Groups
3. Creating a Resource Server
4. Creating a Confidential Client Application
5. Creating the Public Client
6. Generating Two-Legged Access Tokens
7. Configuring IDCS in REST Services Manager

Creating a Confidential OAuth Application

You use the Administration Application to create other Oracle Identity Cloud Service applications that are used by the Offline Mediation Controller.

To create the Administration Application in IDCS:

1. Open your Oracle Identity Cloud Service domain.

   The **Overview in the Domain** window appears.
2. From the Identity domain navigation pane, click **Integrated applications** and then **Add application**.
3. Select **Confidential Application** and then click **Launch workflow**.

   The **Add Confidential Application** window appears.
4. Click **Next** or **Configure OAuth**.
5. Specify the name of the application, such as **OCOMC Admin App**, and add an optional description.
6. In the **Client configuration** card, select **Configure this application as a client now**.

   The **Client configuration** area expands.
7. Under **Allowed grant types**, select the **Client Credentials** option.
8. Under **Token issuance policy**, select **Add app roles**.

   The **App roles** area appears.
9. Click **Add roles**.

   The **Add app roles** dialog box appears.
10. Select the **Application Administrator** role and click **Add**.
11. Click **Next** or **Configure policy**.
12. Under **Web tier policy**, select **Skip and do later** and click **Finish**.

    The application is created.
13. Click **Activate** and then, in the confirmation pop-up, click **Activate application**.

    The application is activated.

Write down the **clientId** and **clientSecret**. You will need it for the following procedures.

Creating Groups

You manage user access to the Offline Mediation Controller functionality using groups. The Offline Mediation Controller resource server contains scopes for Designer, Operator, and Viewer users.

To create groups on the IDCS Cloud Console:

1. Open your Oracle Identity Cloud Service domain.

   The **Overview in the Domain** window appears.
2. From the Identity domain navigation pane, click **Groups**.

   The **Groups in the Domain** window appears.
3. Click **Create Group**.

The **Create Group** dialog box appears.

4. Create the **Designer** group by doing the following:
   a. In the **Name** field, enter **Designer**.
   b. Select the users to assign to the group.
   c. Click **Create**.

> **Note**
>
> You can edit the group and assign users at a later stage as well.

5. Repeat step 4 to create the **Operator** and **Viewer** groups.

Creating a Resource Server

To create a Resource Server on IDCS:

1. Generate an access token using this cURL command:

```
curl --location 'https://idcs_hostname/oauth2/v1/token'
--header 'Content-Type: application/x-www-form-urlencoded'
--header 'Authorization: Basic *******encoded_client'
--data-urlencode 'grant_type=client_credentials'
--data--urlencode 'scope=urn:opc:idm:__myscopes__'
```

   where:

   - *idcs_hostname* is the hostname of your Identity Cloud Service instance.
   - *encoded_client* is the base64-encoded string of the *clientId*:*clientSecret* that you created.

   For more information, see "Generate Access Token and Other OAuth Runtime Tokens to Access the Resource" in *REST API for Oracle Identity Cloud Service*.

2. Create an Offline Mediation Controller Resource Server application. This example creates a confidential application named **OCOMC-ResourceServer** with the following values:
   - The allowed grants are **client_credentials** and **refresh_token**
   - The audience is **ocomc**
   - The application scopes are **Designer**, **Operator**, and **Viewer**

```
curl --location 'https://idcs_hostname/admin/v1/Apps'
--header 'Authorization: Bearer *******access_token'
--header 'Content-Type: application/json'
--data '{"schemas":["urn:ietf:params:scim:schemas:oracle:idcs:App"],
"basedOnTemplate":{"value":"CustomWebAppTemplateId"},
"displayName":"OCOMC-ResourceServer","description":"Resource Server for protecting
the mediation backend API","name":"OCOMC-ResourceServer",
"clientType":"confidential","isAliasApp":false,"isOPCService":false,"active":true,
"isOAuthClient":true,"isUnmanagedApp":true,"isWebTierPolicy":false,
"isOAuthResource":true,"allowedGrants":["client_credentials","refresh_token"],
"allowOffline":true,"allUrlSchemesAllowed":true,"trustScope":"Account",
"accessTokenExpiry":1800,"refreshTokenExpiry":3600,"audience":"ocomc",
"scopes":[{"description":"Scope for Operator role in
OCOMC","requiresConsent":false,"value":"Operator"}, {"description":"Scope for
Designer role in  OCOMC","requiresConsent":false,"value":"Designer"},
```

```
{"description":"Scope for Viewer role in
OCOMC","requiresConsent":false,"value":"Viewer"}]}'
```

where:

- *idcs_hostname* is the hostname of your Identity Cloud Service instance.
- *access_token* is the access token returned in step 1.

  For more information, see "Create an App" in *REST API for Oracle Identity Cloud Service*.

Creating a Confidential Client Application

To create a confidential client application, use cURL to send an HTTP/HTTPS request to the Oracle IDCS URL. The following command creates a confidential client application named **OCOMC-RestClient** with the following values:

- The allowed grants are **client_credentials** and **refresh_token**.
- The allowed operation is **introspect**.
- The application scopes are **Designer**, **Operator**, and **Viewer**.
- The allowed scopes specify the fully qualified server (FQS) value in the format *AudienceScope*. If you have changed the default audience, update it here. The default value is **ocomc***Scope*. For example, **ocomcDesigner**, or **ocomcOperator**.

```
curl --location 'https://idcs_hostname/admin/v1/Apps'
--header 'Authorization: Bearer *******access_token'
--header 'Content-Type: application/json'
--data '{"schemas":["urn:ietf:params:scim:schemas:oracle:idcs:App"],
"basedOnTemplate":{"value":"CustomWebAppTemplateId"},"displayName":"OCOMC-
RestClient","name":"OCOMC--RestClient","clientType":"confidential","isAliasApp":false,
"isOPCService":false,"active":true,"isOAuthClient":true,"isUnmanagedApp":false,"isWebTier
Policy":false,"isOAuthResource":false,"allowedGrants":
["client_credentials","refresh_token"],"allowOffline":true,
"allUrlSchemesAllowed":true,"trustScope":"Explicit",
"redirectUris":["https://omdUI_hostname/webApps/mediation"],
"postLogoutRedirectUris":["https://omdUI_hostname/webApps/mediation"],
"allowedOperations":["introspect"],
"allowedScopes":[{"fqs":"ocomcOperator","value":"Operator","description":"RSM Operator
scope","requiresConsent":false},
{"fqs":"ocomcDesigner","designer_value":"Designer","description":"RSM Designer
scope","requiresConsent":false},{"fqs":"ocomcViewer","value":"Viewer","description":"RSM
Viewer scope","requiresConsent":false}]}'
```

where:

- *idcs_hostname* is the hostname of your Identity Cloud Service instance.
- *access_token* is the access token.
- *omdUI_hostname* is the hostname of the server where the Offline Mediation Controller UI is deployed. If this is specified incorrectly, you will get errors from IDCS after logging in.
- *designer_value* is the scope or role of the user.

Write down the **clientId**, which appears as the **name** field, once you create your Confidential Client Application. You will need it for the following procedures as **clientId_conf_app**.

For information, see "Create an App" in *REST API for Oracle Identity Cloud Service*.

Assigning the Authenticator App Role to the Confidential Client Application

You need to assign the App Role Authenticator Client to the Confidential Application once it has been created. To assign the Authenticator App Role:

1. Generate an access token.
2. Create the Authenticator Client role using this cURL command:

```
curl --location 'https://idcs_hostname/admin/v1/AppRoles?
filter=displayName%20eq%20%22Authenticator%20Client%22'
--header 'Authorization: Bearer access_token*****'
--header 'Content-Type: application/json'
```

If successful, you will see a response similar to this:

```
{
    "schemas": [
        "urn:ietf:params:scim:api:messages:2.0:ListResponse"
    ],
    "totalResults": 1,
    "Resources": [
        {
            "id": "1234567890abcdef1234567890abcdef",
            "meta": {
                "resourceType": "AppRole",
                "location": "https://www.example.com/admin/v1/AppRoles/
1234567890abcdef1234567890abcdef"
            },
            "adminRole": true,
            "availableToUsers": false,
            "uniqueName": "IDCSAppId_Authenticator Client",
            "app": {
                "value": "IDCSAppId",
                "name": "IDCSApp",
                "display": "IDCS Application",
                "$ref": "https://www.example.com/admin/v1/Apps/IDCSAppId"
            },
            "availableToGroups": false,
            "displayName": "Authenticator Client",
            "public": false,
            "availableToClients": true,
            "ocid": "ocid1.domainapprole.oc1.phx.xxxx",
            "idcsLastModifiedBy": {
                "value": "UnAuthenticated"
            },
            "idcsCreatedBy": {
                "value": "UnAuthenticated"
            },
            "schemas": [
                "urn:ietf:params:scim:schemas:oracle:idcs:AppRole"
            ]
        }
    ],
    "startIndex": 1,
    "itemsPerPage": 50
}
```

Write down the resource ID value. This is the ID for the Authenticator Client role. For information, see "Create an AppRole" in *REST API for Oracle Identity Cloud Service*.

3. Assign the Authenticator Client role to the Confidential Client application using this command:

```
curl --location 'https://idcs_hostname/admin/v1/Grants'
--header 'Authorization: Bearer access_token*******'
--header 'Content-Type: application/json'
--data '{"app":{"value":"IDCSAppId"},
"entitlement":{"attributeName":"appRoles","attributeValue":"auth_client_roleID"},
"grantMechanism":"ADMINISTRATOR_TO_APP","grantee":
{"value":"clientId_conf_app","type":"App"},
"schemas":["urn:ietf:params:scim:schemas:oracle:idcs:Grant"]}'
```

where:

- *auth_client_roleID* is the resource ID of the Authenticator client role from step 2.
- *clientId_conf_app* is the client ID of the Confidential Application that you received in response, when creating your Confidential Client Application.

For information, see "Add a Grantee to an AppRole" in *REST API for Oracle Identity Cloud Service*.

> **Note**
>
> REST Services Manager caches the roles of the Confidential Application at startup. If you add or remove the Authenticator Client role after REST Services Manager has been started, restart the REST Services Manager to ensure that the new role is picked up correctly.

Creating the Public Client

The Offline Mediation Controller UI uses the Public Client when the UI is deployed with client-side authentication enabled to manage the user login flow.

To create the Public Client:

1. Generate an access token.
2. Create a Public Client using cURL.

   This example command creates a public client named **OCOMC-Public-Client** with the following values:

   - The allowed grants are **refresh_token** and **authorization_code**
   - The application scopes are **Designer**, **Operator**, and **Viewer**

```
curl --location 'https://idcs_hostname/admin/v1/Apps'
--header 'Authorization: Bearer access_token********'
--header 'Content-Type: application/json'
--data '{"schemas":["urn:ietf:params:scim:schemas:oracle:idcs:App"],
"basedOnTemplate":{"value":"CustomWebAppTemplateId"},"displayName":"OCOMC-Public-
Client","description":"Public client used by OCOMC Web UI","name":"OCOMC-Public-
Client","clientType":"public","isAliasApp":false,"isOPCService":false,"active":true,"
isOAuthClient":true,"isUnmanagedApp":false,"isWebTierPolicy":false,"isOAuthResource":
false,"allowedGrants":
["authorization_code","refresh_token"],"allowOffline":true,"allUrlSchemesAllowed":tru
e,"trustScope":"Explicit","redirectUris":["https://omdUI_hostname/webApps/
mediation"],
"postLogoutRedirectUris":["https://omdUI_hostname/webApps/mediation/"],
```

```
"allowedScopes":[{"fqs":"ocomcOperator"},{"fqs":"ocomcDesigner"},
{"fqs":"ocomcViewer"}]}'
```

Once the applications have been created, ensure that the appropriate users have been assigned to their respective groups.

Generating Two-Legged Access Tokens

To generate two-legged access tokens, use cURL to send an HTTP/HTTPS request to the Oracle IDCS URL.

```
curl --location 'https://domain_url/oauth2/v1/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--header 'Authorization: Basic encoded_client********\ \
--data-urlencode 'grant_type=client_credentials' \
--data-urlencode 'scope=scope'
```

where:

- *domain_url* is the hostname of your Identity Cloud Service instance.
- *encoded_client* is the base64-encoded string of the *clientId*:*clientSecret* that you created.
- *scope* is the concatenation of the primary audience (that you set when creating the Resource Server, for example **ocomc**) and the scope, such as **Designer**.

If successful, IDCS generates a token specific to the user with the specified scope. For information, see "Generate Access Token and Other OAuth Runtime Tokens to Access the Resource" in *REST API for Oracle Identity Cloud Service*.

Configuring IDCS in REST Services Manager

To configure Oracle IDCS in your REST Services Manager cloud native environment:

1. Open your **override-values.yaml** file for **oc-cn-ocomc-helm-chart**.
2. Set the **rsm.security.provider** key to **IDCS**.

> **Note**
>
> Do not leave the key empty, or RSM will run without authentication.

3. Set the **rsmOAuthToken** key to the RSM OAuth 2.0 token.
4. Add your security information under the **security.configuration.idcs** section:
   - **idcsUri**: Set this to the IDCS domain URL.
   - **idcsClientId**:Set this to the client ID for your IDCS client application.
   - **idcsClientSecret**: Set this to the client secret in Base64-encoded format.
   - **idcsIntrospectEndpointUri**: Set this to the IDCS introspect URL for token validation.
5. To enable **rsm-automation** jobs, add the valid RSM OAuth token to the **jobs.requestAutomation.config.rsmAuthToken** key.

Enabling Public Access to the JWKS Endpoint in IDCS

To enable public access to the JSON Web Key Set (JWKS) endpoint in Oracle Identity Cloud Service (IDCS):

1. Log in to the IDCS administration console as an administrator.
2. Navigate to the **Settings** tab for your identity domain.
3. On the Settings page, click **Edit Domain Setting** to open the settings drawer.
4. In the “Access signing certificate” section, enable the “Configure client access” option.
5. Click **Save** to apply your changes.

> **Note**
>
> To verify the configuration, access the JWKS endpoint URL (for example, https://<DOMAINURL>/admin/v1/SigningCert/jwk) from a web browser. The page should display a JSON object containing the public keys, without requiring authentication.

Setting Up OAuth Using Oracle Access Management

Setting up OAuth using Oracle Access Management involves these high-level steps:

1. Preparing the Environment
2. Configuring Oracle Unified Directory as the Identity Store
3. Creating a User Using Oracle Unified Directory
4. Fetching User Details from Oracle Unified Directory
5. Testing Oracle Unified Directory as the Identity Store in Oracle Access Management
6. Generating the Access Token
7. Configuring Offline Mediation Controller Cloud Native for Oracle Access Management
8. Accessing an Offline Mediation Controller REST Services Manager Endpoint

Preparing the Environment

Ensure that both Oracle Access Management and Oracle Unified Directory are installed and configured before integrating with Offline Mediation Controller.

When installing Oracle Access Management, ensure that:

- OAuth 2.0 and REST Endpoints are public.
  - You must configure all **/oauth2/rest/**** endpoints as **public resources**.
  - You use these endpoints for token introspection, which allows Oracle Access Management to validate and process OAuth tokens.
- OpenID Configuration Endpoint is public.
  - The **/.well-known/openid-configuration** endpoint must be **public**.
  - This endpoint provides metadata about the OpenID Provider, which is essential for the Offline Mediation Designer UI.

When installing Oracle Unified Directory, ensure that you enable the HTTP service and expose port 8080. For more information, see "Getting Started with Oracle Access Management 12c Series – Overview" in the Oracle Access Management documentation.

Configuring Oracle Unified Directory as the Identity Store

To configure Oracle Unified Directory as the identity store in Oracle Access Management:

1. Launch a browser and log in to the Oracle Access Management Console: **http://** *oam_hostname*:**7001/oamconsole**.
2. Click the **Configuration** tab on the top right and then click **User Identity Stores**.

   The **OCOMCStore** tab appears.
3. Set the following values in the **OCOMCStore** tab:
   a. **Store Name**: Set this to **OUDStore**
   b. **Store Type**: Set this to **OUD: Oracle Unified Directory**
   c. **Location**: Set this to **hostName:hostLDAPPort**
   d. **Bind DN**: Set this to **cn=Directory Manager**
   e. **Password**: Set this to the Oracle Unified Directory password
   f. **Login ID Attribute**: Set this to **uid**
   g. **User Password Attribute**: Set this to **userPassword**
   h. **User Search Base**: Set this to **ou=People,dc=ocomcexample.com**
   i. **Group Name Attribute**: Set this to **cn**
   j. **Group Search Base**: Set this to **ou=Groups,dc=ocomcexample.com**
4. Click **Test Connection** on the top right side of the tab.

   If the connection works, click **OK** in the **Connection Status** window. If not, correct the values and test again.
5. Click **Apply** on the top right to save the definition.
6. Click the **User Identity Store** tab.
7. From the **Default Store** list, select **OUDStore** and then click **Apply**.
8. Under the **Plug-ins** tile, click **Application Security** and then **Authentication Modules**.
9. Click **Search**.

   The **LDAP** module appears.
10. Click the **LDAP** module and set **User Identity Store** to **OUDStore**.
11. Click the **Launch Pad** tab and, in the **Access Manager** tile, click the **Authentication Schemes** link.
12. On the **Search Authentication Schemes** page, click **Search**.

    The search results appear.
13. Select the **LDAPScheme** row in the search results and click **Edit**.
14. In **LDAPScheme**, click **Duplicate**.

    This creates a new scheme with the name **'Copy of LDAP Scheme'**.
15. Set the following values in the scheme:
    a. **Name**: Set this to **LDAPOUDScheme**
    b. **Description**: Set this to **LDAP Scheme Over OUD**

c. **Authentication Module**: Set this to **LDAP**

16. Click **Apply**.
17. Click the **Set As Default** option and then click **OK** in the confirmation pop-up box.

> **Note**
>
> Oracle Unified Directory must be running over either HTTP or HTTPS. This is a required configuration for the Offline Mediation Controller service to establish successful communication with Oracle Unified Directory endpoints.

Creating a User Using Oracle Unified Directory

To create a user using Oracle Unified Directory:

1. Create a **user.ldif** file:

```
dn: uid=ocomcuser,ou=People,dc=Distinguished Name
sn: ocomcuser
cn: ocomcuser
userPassword: password
objectClass: top
objectClass: organizationalPerson
objectClass: person
objectClass: inetOrgPerson
uid: ocomcuser
```

2. To create a user in the OUD Store, run the following commands:

```
cd oud_home/instance_name/OUD/bin
./ldapmodify -a -h OUD_hostname -p OUD_port -D "cn=Directory Manager" -w password -f
path/add_user.ldif
```

3. Create an **add_group.ldif** file to create a group:

```
dn: ou=Designer,ou=groups,dc=ocomcexample.com
objectclass: top
objectclass: groupOfUniqueNames
cn: Designer
ou: groups
description: example description.
```

4. To add the group in the OUD Store, run the following commands:

```
cd oud_home/instance_name/OUD/bin
./ldapmodify -a -h OUD_hostname -p OUD_port -D "cn=Directory Manager" -w password -f
path/add_user.ldif
```

5. Create an **add_user_to_group.ldif** file:

```
dn: ou=PSA Designer,ou=Groups,dc=ocomcexample.com
changetype: modify
add: member
member: uid=ocomcuser,ou=People,dc=ocomcexample.com
```

or with the values:

```
dn: ou=Designer,ou=Groups,dc=ocomcexample.com
changetype: modify
add: member
member: uid=ocomcuser,ou=People,dc=ocomcexample.com
```

6. To add the user to the group in the OUD Store, run the following commands:

```
cd oud_home/instance_name/OUD/bin
./ldapmodify -a -h OUD_hostname -p OUD_port -D "cn=Directory Manager" -w password -f
path/add_user_to_group.ldif
```

Fetching User Details from Oracle Unified Directory

To fetch user details from Oracle Unified Directory, use cURL to send an HTTP/HTTPS request to the Oracle Access Management URL:

```
curl -X POST \
  http://OUD_hostname:OUD_port/rest/v1/directory \
  -H "Content-Type: application/json" \
  -H "Authorization: Basic encoded_password" \
  -d '{
        "msgType": "urn:ietf:params:rest:schemas:oracle:oud:1.0:SearchRequest",
        "base": "ou=Groups,dc=ocomcexample.com",
        "scope": "sub",
        "filter": "(&(objectclass=*)(member=uid=<UID>,ou=People,dc=ocomcexample.com))"
      }'
```

where:

- *OUD_hostname* and *OUD_port* are the hostname and port where Oracle Unified Directory is running.
- *encoded_password* is the Base64-encoded password in the format *BindDN*:*password*.

You will receive a response similar to this sample:

```
{
  "msgType": "urn:ietf:params:rest:schemas:oracle:oud:1.0:SearchResponse",
  "totalResults": 1,
  "searchResultEntries": [
    {
      "dn": "ou=PSA Operators,ou=Groups,dc=ocomcexample.com",
      "attributes": {
        "cn": "All Operator Users",
        "ou": [
          "PSA Users",
          "PSA Operators"
        ],
        "member": "uid=ocomcuser,ou=People,dc=ocomcexample.com",
        "objectClass": [
          "top",
          "groupofNames"
        ]
      }
    }
  ]
}
```

Or you may receive a response similar to this sample:

```
{
  "msgType": "urn:ietf:params:rest:schemas:oracle:oud:1.0:SearchResponse",
  "totalResults": 1,
```

```
  "searchResultEntries": [{
      "dn": "ou=Operator,ou=Groups,dc=ocomcexample.com",
      "attributes": {
        "cn": "All Operator Users",
        "ou": [
          "PSA Users",
          "PSA Operators"
        ],"member": "uid=ocomcuser,ou=People,dc=ocomcexample.com",
        "objectClass": [
          "top",
          "groupofNames"
        ]
      }
    }
  ]
}
```

For more information, see "Add, Delete, Search, Modify or Compare an OUD entry" in *REST API for Oracle Unified Directory Data Management*.

> **Note**
>
> You can create the **User** in any format but it should contain one of these:
>
> - "dn" : "ou=PSA *Scope*,...."
> - "dn" : "ou=*Scope*,..."
>
> For example:
>
> ```
> ou = PSA Operators
> ou = PSA Designer
> ou = PSA Viewers
> ```

Testing Oracle Unified Directory as the Identity Store in Oracle Access Management

To test whether Oracle Unified Domain has been successfully integrated with Oracle Access Management, go to **http://***OAM_hostname***:***OAM_port***/**. This action redirects you to the login page. If you can log in using any user from Oracle Unified Domain, the integration is successful.

Generating the Access Token

To generate an access token, you must create an OAuth identity domain, an OAuth resource server, and an OAuth client.

To enable **OAuth** service in Oracle Access Management:

1. Go to the **Available Services** option in the **Configuration** tab in the **OAM Console**.
2. Click **Enable Service** for **OAuth and OpenIDConnect Service**.

Creating an OAuth Identity Domain

An identity domain corresponds to the notion of a tenant. All clients and resource servers are created under an identity domain.

To create an identity domain, use cURL to send an HTTP/HTTPS request to the Oracle Access Management URL:

```
curl --location 'http://OUD_hostname:OAM_hostname:OAM_port/oam/services/
rest/ssa/api/v1/oauthpolicyadmin/oauthidentitydomain' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic encoded_password' \
--data '{
    "name": "domain_name",
    "identityProvider": "oud_storename",
    "description": "domain_name",
    "tokenSettings": [
        {
            "tokenType": "SSO_LINK_TOKEN",
            "tokenExpiry": 3600,
            "lifeCycleEnabled": false,
            "refreshTokenEnabled": false,
            "refreshTokenExpiry": 3600,
            "refreshTokenLifeCycleEnabled": false
        },
        {
            "tokenType": "ACCESS_TOKEN",
            "tokenExpiry": 3600,
            "lifeCycleEnabled": false,
            "refreshTokenEnabled": true,
            "refreshTokenExpiry": 3600,
            "refreshTokenLifeCycleEnabled": false
        },
        {
            "tokenType": "AUTHZ_CODE",
            "tokenExpiry": 3600,
            "lifeCycleEnabled": false,
            "refreshTokenEnabled": true,
            "refreshTokenExpiry": 3600,
            "refreshTokenLifeCycleEnabled": false
        }
    ],
    "errorPageURL": "/oam/pages/servererror.jsp",
    "consentPageURL": "/oam/pages/consent.jsp"
}'
```

where:

- *encoded_password* is the Base64-encoded format of *username*:*password*.
- *domain_name* is the name of the Oracle Access Management identity domain that you want to create.
- *oud_storename* is the name of Oracle Unified Directory store added in Oracle Access Management server.

For more information, see "Add a new OAuth Identity Domain" in *REST API for OAuth in Oracle Access Manager*.

Creating a Resource Server

A resource server hosts protected resources. The resource server can accept and respond to protected resource requests using access tokens.

To create a resource server, use cURL to send an HTTP/HTTPS request to the Oracle Access Management URL:

```
curl --location 'http://OAM_hostname:OAM_port/oam/services/rest/ssa/api/v1/
oauthpolicyadmin/application' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic encoded_password' \
--data '{
    "name": "resource_server",
    "description": "OIDC Resource Server for OCOMC",
    "scopes": [
        {
            "scopeName": "Operator",
            "description": "Scope for Operator role in OCOMC"
        },
        {
            "scopeName": "Designer",
            "description": "Scope for Designer role in OCOMC"
        },
        {
            "scopeName": "Viewer",
            "description": "Scope for Viewer role in OCOMC"
        }
    ],
    "resourceServerNameSpacePrefix": "ResourceServer",
    "tokenAttributes": [
        {
            "attrName":"sessionId",
            "attrValue":"$session.id",
            "attrType":"DYNAMIC"
        },
        {
            "attrName":"resSrvAttr",
            "attrValue":"RESOURCECONST",
            "attrType":"STATIC"
        }
    ],
    "idDomain": "domain_name",
    "audienceClaim": {
        "subjects": [
            "ab0",
            "ResourceServer"
        ]
    }
}'
```

where:

- *encoded_password* is the Base64-encoded password in the format *username*:*password*.
- *resource_server* is the name of the resource server that you want to create.

For more information, see "Add a new Resource Server" in *REST API for OAuth in Oracle Access Management*.

Creating an OAuth Client

A client is an application that makes protected resource requests on behalf of the resource owner and with the resource owner's authorization.

To create an OAuth client, use cURL to send an HTTP/HTTPS request to the Oracle Access Management URL:

```
curl
--location 'http://OAM_hostname:OAM_port/oam/services/rest/ssa/api/v1/
oauthpolicyadmin/client' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic encoded_password' \
--data '{
    "id": "client_id",
    "secret": "client_secret",
    "scopes": [
        "ResourceServer.Operator",
        "ResourceServer.Designer",
        "ResourceServer.Viewer"
    ],
    "defaultScope": "ResourceServer.Operator",
    "clientType": "client_type",
    "idDomain": "id_domain",
    "description": "Client entry for OAUTH OIDC Domain",
    "name": "client_name",
    "grantTypes": [
        "PASSWORD",
        "CLIENT_CREDENTIALS",
        "JWT_BEARER",
        "REFRESH_TOKEN",
        "AUTHORIZATION_CODE"
    ],
    "redirectURIs": [
        {
            "url": "redirect_url",
            "isHttps": True
        }
    ]
}'
```

where:

- *encoded_password* is the Base64-encoded authorization password in form of *username* : *password*.
- *client_id* and *client_secret* are the client ID and client secret.
- *client_type* is one of these client types:
  - **CONFIDENTIAL_CLIENT** is a client that requires a secret for authentication.
  - **PUBLIC_CLIENT** is a client that does not require a secret. This is used by UI applications to exchange an authorization code for a token.
- *id_domain* is the name of the identity domain under which the client is created.
- *client_name* is the name of the client.

- *redirect_url* is the URL for your client application.

For more information, see "Add a new OAuth Client" in *REST API for OAuth in Oracle Access Management*.

Generating Access Tokens with Two-Legged Flows

To generate an access token with two-legged flow, using client credentials, use cURL to send an HTTP/HTTPS request to the Oracle Access Management URL:

```
curl --location 'http://OAM_hostname:OAM_port'/oauth2/rest/token' \
--header 'X-OAUTH-IDENTITY-DOMAIN-NAME: domain_name' \
--header 'Authorization: Basic encoded_password' \
--data-urlencode 'grant_type=client_credentials' \
--data-urlencode 'scope=resource_server.Scope_name'
```

For more information, see "Create Access Token Flow" in *REST API for OAuth in Oracle Access Management*.

Generating Access Tokens with Three-Legged Flow

To generate three-legged OAuth authentication:

1. Open the following URL in a browser:

   ```
   http://OAM_hostname:OAM_port/oauth2/rest/authorize?
   response_type=code&domain=domain_name&client_id=client_name&scope=Scope&sta
   te=code1234&redirect_uri=redirect_url
   ```

2. Enter your user credentials in the Oracle Access Manager login screen.
3. Click **Allow**.
4. Copy the authorization code from the browser URL.
5. Generate the OAuth access token by submitting a cURL request to the Create Access Token Flow endpoint in the Oracle Access Manager OAuth REST API. For example:

   ```
   curl --location 'http://OAM_hostname:OAM_port/oauth2/rest/token' \
   --header 'X-OAUTH-IDENTITY-DOMAIN-NAME: domain_name' \
   --data-urlencode 'client_id=client_name' \
   --data-urlencode 'grant_type=AUTHORIZATION_CODE' \
   --data-urlencode 'code=authorization_code' \
   --data-urlencode 'code_verifier=zY6trXrusqzdjIQ6v8WsSiHZ5kPKUlqiCagRLnv' \
   --data-urlencode 'redirect_uri=http://localhost:8080/webApps/mediation/'
   ```

For more information, see REST API for OAuth in Oracle Access Manager.

Configuring Offline Mediation Controller Cloud Native for Oracle Access Management

To configure the Offline Mediation Controller cloud native environment to connect with Oracle Access Management, add the following keys to your **override-values.yaml** file for **oc-cn-ocomc-helm-chart**:

```
provider: "OAM"
configuration:
  oam:
    clientId: client_id
    clientSecret: client_secret
    tokenEndpointUri: http://OAM_hostname:14100/oauth2/rest/token
    authorizationEndpointUri: http://OAM_hostname:7777/oauth2/rest/authorize
    introspectEndpointUri: http://OAM_hostname:7777/oauth2/rest/token/
introspect
    oauthIdentityDomainName: IDStore
    oudHostName: oud_hostname
    oudAdminUserName: oud_adminuser
    oudAdminUserPassword: oud_password
    oudHttpPort: oud_httpport
    oudHttpsPort: oud_httpsport
    oudUsersBaseDn: user_basedn
    oudGroupsBaseDn: group_basedn
```

where:

- *client_id* is the client ID to be used for connecting with the OAM server.
- *client_secret* is the client secret to be used for connecting with the Oracle Access Management server. This must be encoded in Base64 format.
- *OAM_hostname* is the host name of the server where Oracle Access Management is running.
- *oud_hostname* is the host name of the Oracle Unified Directory server.
- *oud_adminuser* is the admin username for the Oracle Unified Directory server.
- *oud_password* is the admin password encoded in Base64 format.
- *oud_httpport* is the HTTP port for the Oracle Unified Directory server.
- *user_basedn* is the Oracle Unified Directory server Base-DN to be used by Offline Mediation Controller REST Services Manager.
- *group_basedn* is the Oracle Unified Directory server groups-DN to be used by Offline Mediation Controller REST Services Manager.

> **Note**
>
> Oracle Unified Directory must be running over either HTTP or HTTPS. This is a required configuration for the Offline Mediation Controller service to establish successful communication with Oracle Unified Directory endpoints.

Accessing an Offline Mediation Controller REST Services Manager Endpoint

After your system is configured, you can access an Offline Mediation Controller REST Services Manager endpoint using the access token with the required scope.

You can pass the generated access token as part of the request header. For example:

```
curl --location 'http://ocomc_host:port/v1/nodeManagers' \
     --header 'Authorization: Bearer access_token
```

where:

- *ocomc_host* and *port* are the host name and port for the Offline Mediation Controller REST server.
- *access_token* is the OAuth access token for your Offline Mediation Controller API client.

For more information, see REST API Reference for Offline Mediation Controller.

SSL-Enabled Actions for IDCS and Oracle Access Management

If you are running Oracle Access Management or IDCS with SSL enabled, you can communicate with external services using the following commands:

```
echo | openssl s_client -showcerts -servername serverName -connect
serviceName:servicePort 2>/dev/null | awk '/-----BEGIN CERTIFICATE-----/,/-----END
CERTIFICATE-----/' > certs.pem
# example
echo | openssl s_client -showcerts -servername idcs-12345678.identity.oraclecloud.com -
connect idcs-12345678.identity.oraclecloud.com:443 2>/dev/null | awk '/-----BEGIN
CERTIFICATE-----/,/-----END CERTIFICATE-----/' > idcs_certs.pem
# command to import cert to trustStore
keytool -importcert -trustcacerts -keystore trustStoreName -storepass password -alias
aliasName -file fileName
# example
keytool -importcert -trustcacerts -keystore idcs_trustStore.jks -storepass storePass -
alias idcs-certs -file idcs_certs.pem
# if user have couple of certificates in .pem file, they can split certificates and then
import them individually
#example
csplit -z idcs_certs.pem '/-----BEGIN CERTIFICATE-----/' '{*}'
```

Ensure that the required certificates are imported into the TrustStore. The following fields must be updated in the REST Services Manager charts under the **rsmTrustStore** section:

```
rsmTrustStore:
  enabled: enabledValue
  trustStoreName: trustStoreName
  extRsmTruststoreSecret: extRsmTruststoreSecret
  trustStorePassPhrase: trustStorePassPhrase
```

where:

- *enabledValue* is the action to enable or disable the TrustStore configuration.
- *trustStoreName* is the name of the TrustStore file containing the trusted certificates. This file should include CA (Certificate Authority) certificates necessary for establishing secure

SSL/TLS connections with external services. Ensure the file is present at **oc-cn-ocomc-rsm/ocomc-rsm-keystore**.

- *extRsmTruststoreSecret* is the secret name containing the external RSM TrustStore file.
- *trustStorePassPhrase* is the Base64-encoded passphrase for accessing the TrustStore.

7

Upgrading Offline Mediation Controller

Learn how to upgrade your existing Oracle Communications Offline Mediation Controller cloud native deployment to the latest release.

Topics in this document:

In this document, the Offline Mediation Controller release running on your production system is called the existing release. The release you are upgrading to is called the new release. For example, if you are upgrading from Offline Mediation Controller 12.0 Patch Set 8 to Offline Mediation Controller 15.1, 12.0 Patch Set 8 is the existing release and 15.1 is the new release.

Upgrading Offline Mediation Controller to 15.1

When you upgrade your Offline Mediation Controller cloud native services, it upgrades all core services in your Offline Mediation Controller cloud native environment.

Supported upgrade paths to 15.1:

- Offline Mediation Controller 12.0 Patch Set 8
- Offline Mediation Controller 15.0.0.0.0
- Offline Mediation Controller 15.0.1.0.0

To upgrade your Offline Mediation Controller cloud native services to the 15.1 release:

1. Ensure that you back up your data to prevent data loss.
2. Download the Helm charts for the Offline Mediation Controller cloud native deployment 15.1 version specification. For more information, see "Downloading Packages for the Offline Mediation Controller Cloud Native Helm Charts".
3. Download the Offline Mediation Controller cloud native images in one of these ways:
   a. From the Oracle Container Registry. To do so, see "Pulling Offline Mediation Controller Images from the Oracle Container Registry".
   b. From the Oracle Software Delivery website. To do so, see "Downloading Offline Mediation Controller Images from Oracle Software Delivery Website".
4. Extract the Offline Mediation Controller Helm chart from the archive:

   ```
   tar xvzf oc-cn-ocomc-15.1.0.0.0.tgz
   ```

   If you are extracting an interim patch, the file name will also have the interim patch number appended to it, such as **oc-cn-ocomc-helm-chart-15.1.0.0.0-12345678.tgz**.
5. Enable data migration in the Offline Mediation Controller existing release:
   a. Download the latest Helm charts for your existing Offline Mediation Controller release.
   b. Edit your **override-values.yaml** file and set the **ocomc.statefullSetUpgrade.runMigrationDataJob** key to **true**.
   c. Run the **helm upgrade** command from the **oc-cn-ocomc** directory:

```
helm upgrade --install ReleaseName oc-cn-ocomc -f OverrideValuesFile
```

6. Configure and deploy the Offline Mediation Controller 15.1 release:
   a. Configure the new Helm charts for Offline Mediation Controller 15.1.0.0.
   b. Run the **helm upgrade** command from the **oc-cn-ocomc-helm-chart** directory:

      ```
      helm upgrade --install ReleaseName oc-cn-ocomc-helm-chart -f OverrideValuesFile
      ```
7. Import your custom JAR files.
8. Check the status of the pods, especially the REST Services Manager pod.
9. Validate the stability of the application.
10. To import node chains:
    a. Open your **override-values.yaml** file for the 15.1 version of **oc-cn-ocomc-helm-chart**.
    b. Enable RSM request automation by setting the **rsm.jobs.requestAutomation** key to **true**.
    c. Configure the following keys under **rsm.jobs.requestAutomation.config**:
       - **apiUrl**: Set this to the URL of your Offline Mediation Controller RSM service, such as http://ocomc-rsm:8080.
       - **payloadFile**: Specify the absolute path to your workflow payload JSON file.
       - **loggingDir**: Define the directory where log files should be written.
    d. Set the **global.statefulSetUpgrade.runMigrationDataJob** key to **true**.
    e. Ensure that the **global.RSMcontainer** keys are set properly.
    f. Save and close your **override-values.yaml** file.
    g. Create a workflow payload JSON file that imports node chains from the previous release setup to the new release setup. You can use the sample JSON file to start with. See "Sample Workflow Payload JSON File".
    h. Edit the JSON file.
    i. Under the **customizationMapping** and **configurationMapping** sections, map all Node Managers from the old setup to your new setup.
    j. Set **customJarsToRemove** to the list of custom JAR files to remove.
    k. Save and close your JSON file.
    l. Move the workflow payload JSON file to the external PersistentVolumeClaim (**/home/ocomcuser/external**) or to **migration-pvc** (**/home/ocomcuser/migration**). This location must match the one specified in your **override-values.yaml** file's **payloadFile** key.
11. Run the **helm upgrade** command from the **oc-cn-ocomc-helm-charts** directory:

    ```
    helm upgrade NameSpace oc-cn-ocomc-helm-charts -values OverrideValuesFile -n
    ReleaseName
    ```

    where:
    a. *NameSpace* is the namespace in which to create Offline Mediation Controller Kubernetes objects.
    b. *OverrideValuesFile* is the path to a YAML file that overrides the default configurations in the chart's **values.yaml** file.

c. *ReleaseName* is the release name, which is used to track this installation instance.

> **Note**
>
> - If there are configuration keys present in the sub-charts, or from previous installations during upgrades, which are not found in the new **values.yaml** file, add them to your **override-values.yaml** file.
> - A failure in the above steps may lead to the loss of data.

Sample Workflow Payload JSON File

This sample specifies to import node chains from the previous release setup to the new release setup.

```
{
  "workflow": {
    "flow": [
      {
        "id": "UPLOAD_IMPORT_FILE_ACTION",
        "request": "UPLOAD_IMPORT_FILE",
        "onResponse": {
          "environment": {
            "IMPORT_ID": "/importId"
          },
          "statusMapping": {
            "2**": ["START_IMPORT_ACTION"]
          }
        }
      },
      {
        "id": "START_IMPORT_ACTION",
        "request": "START_IMPORT",
        "onResponse": {
          "environment": {
            "IMPORT_TASK_ID": "/importTaskId"
          },
          "statusMapping": {
            "202": ["CHECK_IMPORT_COMPLETION_ACTION"]
          }
        }
      },
      {
        "id": "CHECK_IMPORT_COMPLETION_ACTION",
        "request": "CHECK_IMPORT_STATUS",
        "retry": {
          "maxRetries": "5",
          "interval": "10000"
        },
        "onResponse": {
          "environment": {
            "END_TIME": "/importTaskItem/endTime",
            "START_TIME": "/importTaskItem/startTime"
          },
          "statusMapping": {
            "5**": ["RETRY"]
          },
          "dataMapping": [
            {
```

```
                "path": "/importTaskItem/taskFinished",
                "equals": "true",
                "then": []
              },
              {
                "path": "/importTaskItem/taskFinished",
                "equals": "false",
                "then": ["RETRY"]
              }
            ]
          }
        }
      ]
    },
    "headers": {},
    "environment": {},
    "requests": [
      {
        "id": "UPLOAD_IMPORT_FILE",
        "method": "POST",
        "uri": "/v1/imports/uploadImportFile",
        "headers": {},
        "multipartForm": {
          "configurationFile": "FILE::/home/ocomcuser/migration/nodeChains/
exported_node_chains.xml",
          "customizationFile": "FILE::/home/ocomcuser/migration/nodeChains/
exported_node_chains.nmx"
        }
      },
      {
        "id": "START_IMPORT",
        "method": "POST",
        "uri": "/v1/imports/{{ENV:IMPORT_ID}}/tasks",
        "headers": {},
        "payload": {
          "customizationMapping": [
            {
              "sourceTriplet": {
                "ip": "node-mgr-app",
                "name": "nm1",
                "port": 32170
              },
              "destinationTriplet": {
                "ip": "nm-cc-0",
                "name": "nm-cc-0",
                "port": 55109
              }
            },
            {
              "sourceTriplet": {
                "ip": "node-mgr-app-2",
                "name": "nm2",
                "port": 32172
              },
              "destinationTriplet": {
                "ip": "nm-epdc-0",
                "name": "nm-epdc-0",
                "port": 55109
              }
            },
            {
              "sourceTriplet": {
```

```
            "ip": "node-mgr-app-100",
            "name": "nm100",
            "port": 32270
          },
          "destinationTriplet": {
            "ip": "nm-epdc-0",
            "name": "nm-epdc-0",
            "port": 55109
          }
        }
      ],
      "configurationMapping": [
        {
          "sourceTriplet": {
            "ip": "node-mgr-app",
            "name": "nm1",
            "port": 32170
          },
          "destinationTriplet": {
            "ip": "nm-cc-0",
            "name": "nm-cc-0",
            "port": 55109
          }
        },
        {
          "sourceTriplet": {
            "ip": "node-mgr-app-2",
            "name": "nm2",
            "port": 32172
          },
          "destinationTriplet": {
            "ip": "nm-epdc-0",
            "name": "nm-epdc-0",
            "port": 55109
          }
        },
        {
          "sourceTriplet": {
            "ip": "node-mgr-app-100",
            "name": "nm100",
            "port": 32270
          },
          "destinationTriplet": {
            "ip": "nm-epdc-0",
            "name": "nm-epdc-0",
            "port": 55109
          }
        }
      ],
      "regenerateNodeIds": false,
      "skipConfiguration": false,
      "skipCustomization": false,
      "skipRestart": false,
      "customJarsToRemove": [],
      "importType": "NODE_MANAGER"
    }
  },
  {
    "id": "CHECK_IMPORT_STATUS",
    "method": "GET",
    "uri": "/v1/imports/{{ENV:IMPORT_ID}}/tasks/{{ENV:IMPORT_TASK_ID}}",
    "headers": {},
```

```
        "payload": {}
      }
    ]
  }
```

8

Connecting Your Administration Client

Learn how to connect your Oracle Communications Offline Mediation Controller cloud native deployment with an on-premises version of Offline Mediation Controller Administration Client.

Topics in this document:

About Administration Client

Administration Client is a GUI application that you use for creating node chains and editing rule files. You also use Administration Client for administrating Offline Mediation Controller. For example, you can use it to manage users and define instances of system components.

For more information about using Administration Client, see "About Configuring Nodes and Node Chains" in *Offline Mediation Controller User's Guide*.

Connecting Administration Client

Although Offline Mediation Controller can be deployed on a cloud native environment, you must install an on-premises version of Administration Client to work with it.

To set up a connection between your on-premises Administration Client and the Administration Server on a cloud native environment, do the following:

1. Configure the Administration Server cloud native service to connect to your Administration Client. See "Configuring Administration Server Cloud Native".
2. Install an on-premises version of Administration Client on one of the following:
   - On a physical server that is reachable to the Kubernetes node where the Administration Server pod is running.
   - If graphical desktop support such as VNC is available on a worker node, you can install Administration Client on the same worker node in which the Administration Server and Node Manager pods are running.

   See "Installing Offline Mediation Controller Administration Client" in *Offline Mediation Controller Installation Guide*.
3. Perform postinstallation tasks on the Administration Client machine. See "Postinstallation Tasks for Administration Client".
4. Verify that your Administration Client can connect to the Administration Server. See "Verifying the Administration Client Connection".

After your Administration Client has connected successfully, ensure that you place all CDR files inside the vol-data PVC and that all CDRs have read and write permission for the **ocomcuser** user.

Configuring Administration Server Cloud Native

When configuring your Offline Mediation Controller Administration Server cloud native service, ensure that you do the following:

1. Expose the Administration Server pod (admin-server-app):
   - If your Administration Client is located remotely or is on a Windows system, set the Administration Server's service type to **NodePort**.
   - If your Administration Client is installed on the same worker node in which the Administration Server pod is running, set the Administration Server's service type to **clusterIP**.
2. Open your **override-values.yaml** file for **oc-cn-ocomc-core-helm-chart**.
3. If your Administration Client is installed remotely or on a Windows system, set these additional keys:
   - **service.type**: Set this to **NodePort**.
   - **service.appPort**: Set this to the external port of the Administration Server.
   - **service.firewallPort**: Set this to the Administration Server firewall port.
   - **service.callbackPort**: Set this to the appropriate callback port.
4. Save and close your **override-values.yaml** file.
5. Deploy Offline Mediation Controller by following the instructions in "Deploying Offline Mediation Controller Services".

The following shows sample **override-values.yaml** entries for an Administration Client that is installed remotely or on a Windows system:

```
adminServerConfigurations:
    service:
        type: NodePort
        appPort: 31200
        firewallPort: 31201
        callbackPort: 31202
```

Postinstallation Tasks for Administration Client

After you install Administration Client, perform the following postinstallation tasks:

1. Copy all Offline Mediation Controller cartridges and your custom cartridges from the cloud native environment's **/home/ocomcuser/ext/cartridges** directory to the Administration Client's *OMC_home***/cartridges** directory.
2. In the Administration Client machine's **/etc/hosts** file, add the IP address of the Kubernetes node where Administration Server is running. For example:

   ```
   IPAddress        Hostname
   198.51.100.1     myhost.example.com
   ```

3. In your Administration Client, specify the location of the Offline Mediation Controller wallet.

a. Go to the *OMC_home***/bin/** directory and open either the **UDCEnvironment.bat** file (Windows) or the **UDCEnvironment** file (UNIX).

b. Set the **OCOMC_WALLET_LOCATION** parameter to the externally mounted wallet PV.

c. Save and close the file.

d. Restart Offline Mediation Controller. See "Starting Offline Mediation Controller" in *Offline Mediation Controller Installation Guide*.

4. Ensure that the Offline Mediation Controller Administration Client machine has access to the wallet files used by the Kubernetes deployment. The **checksum** of the wallet file referred by Administration Client must match with the wallet file in the Kubernetes PV.

5. If SSL is enabled, copy the **adminClientTruststore.jks** file from **vol-external PVC** on the cloud native environment to the Administration Client's *OMC_home***/config/GUI** directory.

Verifying the Administration Client Connection

Start your Administration Client to verify it can connect to your Administration Server on the cloud native environment.

To verify the Administration Client connection:

1. Start Administration Client.

   a. Go to the *OMC_home***/bin** directory.

   b. Run the following command:

   ```
   ./gui -f
   ```

   Administration Client starts in the foreground.

2. In the **Welcome to Oracle Communications Offline Mediation Controller** dialog box, do the following:

   - In the **Host** field, enter **admin-server-app**.
   - In the **Port** field, enter the Administration Server node port number.
   - Enter your user name and password.

3. Click **Connect**.

   If the Administration Client successfully connects to the Administration Server, you will see the Offline Mediation Controller Administration Client window.

9

Enabling TLS 1.3 Support in Offline Mediation Controller

Learn how to enable TLS 1.3 support in Oracle Communications Offline Mediation Controller deployments, enhancing communication security. TLS 1.3 offers improved security features compared to older protocols.

Topics in this document:

About TLS 1.3 Compatibility

Before enabling TLS 1.3, it is important to understand some potential compatibility considerations. When it comes to backwards compatibility, TLS 1.3 can negotiate with older clients (TLS 1.2 and below) but has some key differences:

- TLS 1.3 uses a half-close policy, while TLS 1.2 and above earlier use a duplex-close policy. Applications that depend on the latter duplex-close policy may encounter compatibility issues when upgrading to TLS 1.3.
- The signature_algorithms_cert extension warrants the use of predefined signature algorithms for certificate authentication.
- The DSA signature algorithm is not supported in TLS 1.3. A server cannot negotiate with a TLS 1.3 connection if it is configured to only use DSA certificates.
- The supported cipher suites for TLS 1.3 are not the same for TLS 1.2 and earlier versions. Applications with hard-coded cipher suites that are no longer supported may not be able to use TLS 1.3 without modifications to its code.
- Session resumption and key update behaviors are different for TLS 1.3 and TLS 1.2. Although the compatibility impact should be minimal, it is a potential risk if an application depends on the handshake details of the TLS protocols.

Enabling TLS 1.3 Support Automatically

To enable support for TLS 1.3 automatically for your Offline Mediation Controller cloud native deployment:

1. Ensure you are using the latest Offline Mediation Controller 15.1 image.
2. In your **override-values.yaml** file, set the **ocomcCore.forceGenSslcert** key to **true**.
3. Run the **helm upgrade** command for **oc-cn-ocomc-core-helm-chart-15.1.0.0.0**.

These steps automatically generate new certificates in the Offline Mediation Controller image using the latest JDK available. If you encounter compatibility issues, enable TLS 1.3 support manually.

Manually Enabling TLS 1.3 Support

To manually enable TLS 1.3 in your Offline Mediation Controller cloud native deployment:

1. Generate a new KeyStore using the **keytool** utility. If generating externally, use the latest Java version.
2. Use a signature algorithm supported by TLS 1.3 during certificate generation.
3. Load the newly generated KeyStore into the appropriate TrustStore.
4. Restart all Offline Mediation Controller components after loading the new KeyStore.

10

Uninstalling Your Offline Mediation Controller Cloud Native Deployment

Learn how to uninstall your Oracle Communications Offline Mediation Controller cloud native deployment.

Topics in this document:

Uninstalling Your Offline Mediation Controller Cloud Native Deployment

When you uninstall a Helm chart from your Offline Mediation Controller cloud native deployment, it removes only the Kubernetes objects that it created during installation.

Before you uninstall the Offline Mediation Controller Helm chart, back up all data inside mounted file systems.

To uninstall, enter this command:

```
helm delete ReleaseName -n Namespace
```

where:

- *ReleaseName* is the name you assigned to this installation instance.
- *NameSpace* is the namespace in which the Offline Mediation Controller Kubernetes objects reside.

11

Automated Scaling of Node Manager Pods Using HPA

Learn how to configure the automatic scaling of Node Manager pods in Oracle Communications Offline Mediation Controller cloud native.

Topics in this document:

About Automated Scaling of Node Manager Pods

Offline Mediation Controller cloud native supports the Kubernetes Horizontal Pod Autoscaler (HPA), enabling dynamic scaling of Node Manager pods to handle varying workloads. With this feature, the Node Manager pods are replicated as the application scales up, distributing the load evenly and ensuring optimal resource utilization during processing.

Offline Mediation Controller uses StatefulSets to run groups of Node Manager pods. You can configure Offline Mediation Controller to automatically scale Node Manager pods in StatefulSets at the following levels:

- **Global Level**: Provides a default, system-wide approach to HPA. These settings apply to all Node Manager StatefulSets unless explicitly overridden.
- **Node Manager Set Level**: Offers control for the pods in each Node Manager StatefulSet. Set-level configuration takes precedence over the global level.

Enabling Scaling Replication

When HPA scaling is enabled, new Node Manager instances are created, and the node chain is replicated to distribute the load across the scaled instances. You control replication using the **scaling.replication.enabled** flag, which ensures that new nodes share the load during scale-up events. Each new instance replicates the node chain and participates in load distribution. In addition to scaling up, you can also configure Node Manager pods to scale down when resource usage decreases. You control the scale-down behavior using the **scaling.hpa.hpaScaleDownEnabled** key.

Scaling a Node Manager up or down triggers a restart for nodes with routes linked to that Node Manager since the routes are modified.

> **Note**
>
> To enable REST Services Manager authentication, you must set the **ocomc.secrets.rsmOAuthToken** key in your **override-values.yaml** file.

Configuring HPA

The Node Manager scaling and resource configurations can be managed at both global and Node Manager set levels.

- **Global Configuration:** Provides a default, system-wide approach to HPA. When defined in the global **nodeManagerConfigurations** block, these settings apply to all Node Manager sets unless explicitly overridden.
- **Node Manager Set Configuration:** Offers control for each Node Manager set.

> **Note**
>
> RDM configurations, such as thread count, require appropriate tuning when HPA is enabled.

Configuring Global HPA Values

The **nodeManagerConfigurations** block in the **override-values.yaml** file defines the global settings for Node Managers. These settings can be overridden at the Node Manager set level if needed. For example, if the global log level configuration is set to **WARN**, you can configure the nm-voice-cc set to use the **DEBUG** log level instead.

To configure global HPA values:

1. Set the **scaling.hpa.enabled** key to **true**.
2. Configure the following global HPA parameters:
   - **scaling.replication.enabled**: Set to **true** to specify if the node chain needs to be replicated from the root Node Manager (that is, the first pod of the StatefulSet).
   - **scaling.hpa.maxReplicas**: Specify the maximum number of pod replicas allowed.
   - **scaling.hpa.metrics**: Define the scaling triggers based on resource utilization.
   - **scaling.serviceAccount.createServiceAccount**: Set to **true** to create a dedicated service account for scaling operations.
   - **scaling.serviceAccount.name**: Specify the name for the service account to be used for scaling operations.
   - **scaling.hpa.hpaScaleDownEnabled**: Set to **true** to enable scaling down of Node Manager pods when resource usage decreases.

   For example:

```
nodeManagerConfigurations:
  scaling:
    replication:
      enabled:
    hpa:
      enabled: true
      maxReplicas: 3
      metrics:
        - type: Resource
          resource:
            name: cpu
            target:
```

```
        type: Utilization
        averageUtilization: 50
    - type: Resource
      resource:
        name: memory
        target:
          type: Utilization
          averageUtilization: 70
```

This example specifies to scale resource utilization as follows:

- **CPU Utilization Metric**: The HPA monitors CPU usage for each Node Manager pod. If the average CPU utilization across all replicas exceeds 50%, the HPA initiates scaling by adding more instances (up to the maximum specified in **maxReplicas**).
- **Memory Utilization Metric**: The memory usage of each pod is monitored. If the average memory utilization reaches 70%, the HPA triggers scaling to ensure that enough instances are available to handle the workload.

3. Run the **helm upgrade** command to update the Offline Mediation Controller Helm release with the values you have set:

   `helm upgrade` *`ReleaseName`* **`oc-cn-ocomc-core-helm-chart --values`** *`OverrideValuesFile`* **`-n`** *`Namespace`*

   where:

   - *ReleaseName* is the release name, which is used to track the installation instance.
   - *OverrideValuesFile* is the path to a YAML file that overrides the default configurations in the chart's **values.yaml** file.
   - *Namespace* is the namespace in which to create Offline Mediation Controller Kubernetes objects.

Configuring Node Manager Set HPA Values

Each Node Manager set can have its own specific configuration. If a configuration is not explicitly defined for a set, it inherits the values from the global **nodeManagerConfigurations** block.

To configure HPA values for Node Manager sets:

1. In your **override-values.yaml** file, locate the **sets** block.
2. For each Node Manager set, use the **scaling.hpa.enabled** key to enable scaling.
3. Configure the following HPA parameters:
   - **name:** Specify the name of the Node Manager set.
   - **replicas:** Specify the maximum number of pod replicas for this set.
   - **resources:** Define the minimum and maximum CPU and memory resources that the Node Manager contains.

   For example:

```
sets
  nm-cc:
    name: "nm-cc"
    replicas: 1
    resources:
      requests:
```

```
      cpu: "800m"
      memory: "5Gi"
    limits:
      cpu: "800m"
      memory: "5Gi"
  scaling:
    hpa:
      enabled: true
  gcOptions: "${GLOBAL_OPTS}
  memoryOptions: "-Xms4g -Xmx4g"
```

4. Run the **helm upgrade** command to update the Offline Mediation Controller Helm release:

   helm upgrade *ReleaseName* **oc-cn-ocomc-core-helm-chart --values** *OverrideValuesFile* **-n** *Namespace*

12

Monitoring and Maintaining Offline Mediation Controller Cloud Native

Learn how to monitor and maintain your Oracle Communications Offline Mediation Controller cloud native deployment.

Topics in this document:

Using Prometheus Operator to Monitor Offline Mediation Controller Cloud Native

Offline Mediation Controller cloud native tracks and exposes the following metric data in Prometheus format:

- Node Manager-level statistics, which include:
  - The total network account records (NARs) processed
  - The current NARs processed
  - The current processing rate
  - The average processing rate

  Node Manager sets expose metrics through a configurable service port. In your **override-values.yaml** file, each set includes a **metrics** section where the port for metrics can be specified using the **metrics.service.port** key. For example:

```
sets:
    nm-cc
        name: "nm-cc"
        replicas: 1
        metrics:
            service:
                port: 31300
```

- JVM metrics for all Offline Mediation Controller components, which include:
  - Performance on the Node Manager level
  - JVM parameters

JVM metrics are exposed through the endpoint **http://***hostname***:***JVMport***/metrics**, where *JVMport* is the port number where the JVM metrics are exposed. They are exposed on the same port as the Node Manager metrics for each set, which is configured using **metrics.service.port** in your **override-values.yaml** file.

- REST Services Manager metrics, which include:
  - JVM metrics
  - Service status
  - Total uptime
  - Garbage collection events
  - Memory usage statistics
  - Thread count
  - Class loader statistics

  REST Services Manager metrics are exposed through the Helidon framework's **http://***RSM_hostname:RSM_port***/metrics** endpoint, where *RSM_hostname* is the host name of the machine on which REST Services Manager is installed. The endpoint exposes metrics information in both JSON format (according to the MicroProfile Metrics specification) and plain text format suitable for Prometheus.

To monitor Offline Mediation Controller more easily, you can configure an external centralized metrics service, such as Prometheus Operator, to scrape metrics from each endpoint and store them for analysis and monitoring. You can then set up a visualization tool, such as Grafana, to display your metric data in a graphical format.

For the list of compatible Prometheus Operator and Grafana software versions, see "Offline Mediation Controller Cloud Native Deployment Software Compatibility" in *Offline Mediation Controller Compatibility Matrix*.

Enabling the Automatic Scraping of Metrics

You can configure the Prometheus Operator ServiceMonitor to automatically scrape Offline Mediation Controller metrics and Offline Mediation Controller REST Services Manager metrics. For more information about Prometheus Operator and ServiceMonitors, see the prometheus-operator documentation on the GitHub website (https://github.com/prometheus-operator/prometheus-operator/tree/main/Documentation/getting-started).

To enable the automatic scraping of Offline Mediation Controller metrics:

1. Install Prometheus Operator on your cloud native environment.
2. In your **override-values.yaml** file for **oc-cn-ocomc**, set the following keys:
   - **ocomcCore.ocomc.nodeManagerConfigurations.serviceMonitor.enabled:** Set this key to **true**.
   - **ocomcCore.ocomc.nodeManagerConfigurations.serviceMonitor.interval:** Set this to the interval at which to scrape metrics. The default is **10s**.
   - **ocomcCore.ocomc.nodeManagerConfigurations.serviceMonitor.labels.app**: Set this to the app label you want applied to the ServiceMonitor resource. This label is used by Prometheus to discover the ServiceMonitor via label selectors.
   - **ocomcCore.ocomc.nodeManagerConfigurations.serviceMonitor.labels.release:** Set this to the release label you want applied to the ServiceMonitor resource. This label is used by Prometheus to discover the ServiceMonitor via label selectors.

- **ocomcCore.ocomc.nodeManagerConfigurations.metrics.service.type:** Set this to the service type, such as **NodePort** or **ClusterIP**. The default is **ClusterIP**.
- (For REST Services Manager) **ocomcRSM.rsm.serviceMonitor.enabled:** Set this key to **true**.

3. Run the **helm upgrade** command to update the Offline Mediation Controller Helm release:

```
helm upgrade ReleaseName oc-cn-ocomc --values OverridingValuesFile -n NameSpace
```

where:

- *ReleaseName* is the release name, which is used to track the installation instance.
- *OverrideValuesFile* is the path to a YAML file that overrides the default configurations in the chart's **values.yaml** file.
- *NameSpace* is the namespace in which to create Offline Mediation Controller Kubernetes objects.

Using the Sample Grafana Dashboards

The Offline Mediation Controller package includes sample Grafana Dashboard templates that you can use for visualizing metrics. To use the sample dashboards, import the following JSON files from the *OMC_home*/**sampleData/dashboards** directory into Grafana:

- **OCOMC_JVM_Dashboard.json**: This dashboard lets you view JVM-related metrics for Offline Mediation Controller.
- **OCOMC_Node_Manager_Summary.json**: This dashboard lets you view NAR processing metrics for the Node Manager.
- **OCOMC_Node_Summary.json**: This dashboard lets you view NAR processing metrics for all nodes.
- **OCOMC_Summary_Dashboard.json**: This dashboard lets you view NAR-related metrics for all Offline Mediation Controller components.
- **OCOMC_RSM_JVM_Dashboard.json**: This dashboard lets you view JVM-related metrics for Offline Mediation Controller RSM.

For information about importing dashboards, see "Manage Dashboards" in the *Grafana Dashboards* documentation.

Offline Mediation Controller REST Services Manager Metrics

You can use various metrics to monitor the performance and health of Offline Mediation Controller REST Services Manager.

Table 12-1 describes the metrics mapped to their variable names in the **/metrics** endpoint response.

Table 12-1 Metrics Mapped to Variable Names

Variable Name	Metric Description
up	The status of the service, indicating whether it is up and running.
base_jvm_uptime_seconds	The total uptime of the JVM in seconds since it was started.
base_gc_total	The total number of garbage collection events that have occurred using the copy and mark/sweep algorithms.

Table 12-1 (Cont.) Metrics Mapped to Variable Names

Variable Name	Metric Description
base_gc_time_seconds	The total time spent in garbage collection using the copy and mark/sweep algorithms., measured in seconds.
base_memory_usedHeap_bytes	The amount of memory currently used in the heap, measured in bytes.
base_memory_commitedHeap_bytes	The amount of memory that has been committed for use in the heap, measured in bytes.
base_memory_maxHeap_bytes	The maximum amount of memory that can be allocated for the heap, measured in bytes.
base_thread_count	The current number of threads that are actively running in the JVM.
base_classloader_loadedClasses_count	The number of classes currently loaded into memory by the class loader.
base_classloader_loadedClasses_total	The total number of classes that have been loaded into memory since the JVM started.
base_classloader_unloadedClasses_total	The total number of classes that have been unloaded from memory since the JVM started.

Automating Workflows Using RSM Request Automation

The RSM Request Automation feature enables response-driven execution of workflows within the Offline Mediation Controller cloud native environment. It is implemented as a Kubernetes job that dynamically executes workflows based on a structured JSON payload file. This file defines the sequence of API requests to run, response handling, and any conditional logic that determines the execution flow. This feature also allows users to execute any API exposed by the REST Services Manager.

With this feature, you can:

- Automatically define and execute a sequence of steps based on predefined rules and conditions.
- Use data from API responses to set environment variables for subsequent requests.
- Leverage predefined flows for error handling and retries.
- Support multipart file uploads as part of API requests.
- Apply conditions based on response data to control execution the process flow.
- Chain requests based on the output of previous ones.

Setting Up REST Services Manager Request Automation

To set up the REST Services Manager Request Automation feature in your Offline Mediation Controller cloud native deployment:

1. In your **override-values.yaml** file for **oc-cn-ocomc-rsm**, set the **rsm.jobs.requestAutomation.enabled** key to **true**.
2. Configure the following keys under **rsm.jobs.requestAutomation.config**:
   - **apiUrl**: Set this to the URL of your Offline Mediation Controller REST Services Manager service, such as **http://ocomc-rsm:8080**.
   - **payloadFile**: Specify the absolute path to your workflow payload JSON file.
   - **loggingDlr**: Define the directory where log files should be written.

- (Optional) **rsmOAuthToken**: Set this to the REST Services Manager OAuth 2.0 token.

The following shows an example configuration:

```
jobs:
  requestAutomation:
    enabled: true
    config:
      apiUrl: "http://ocomc-rsm:8080"
      payloadFile: "/app/config/workflow-payload.json"
      loggingDir: "/app/volumes/ocomc-ext/logs
```

3. Create a JSON workflow payload file following the workflow file syntax. See "Creating a Workflow Payload File" for details.
4. Move the JSON workflow payload file to the directory specified in the **payloadFile** key.
5. Run a **helm install** or **helm upgrade** command to deploy your changes.

Creating a Workflow Payload File

The workflow payload file defines your automation workflow in JSON format.

The following is an example of the workflow file structure:

```
{
  "workflow": {
    "flow": [
      {
        "id": "UPLOAD_IMPORT_FILE_ACTION",
        "request": "UPLOAD_IMPORT_FILE",
        "onResponse": {
          "environment": {
            "IMPORT_ID": "/importId"
          }
        }
      },
      {
        "id": "START_IMPORT_ACTION",
        "request": "START_IMPORT",
        "onResponse": {
          "statusMapping": {
            "2**": ["CHECK_IMPORT_COMPLETION_ACTION"]
          }
        }
      }
    ]
  },
  "headers": {},
  "environment": {},
  "requests": [
    {
      "id": "UPLOAD_IMPORT_FILE",
      "method": "POST",
      "uri": "/imports/uploadImportFile",
      "headers": {},
      "multipartForm": {
        "configurationFile": "FILE::/path/to/export.xml"
      }
    },
    {
      "id": "START_IMPORT",
      "method": "POST",
```

```
      "uri": "/imports/{{ENV:IMPORT_ID}}/tasks",
      "headers": {},
      "payload": {}
    }
  ]
}
```

The following is a breakdown of the syntax used within the workflow file, covering the key elements and options.

Table 12-2 describes the structure of the workflow section, which defines the sequence of actions to be executed.

Table 12-2 Workflow Section

Element	Description
flow	An array defining the sequence of actions to run.
id	A unique identifier for the action within the workflow.
request	The ID of the API request to run (defined in the **requests** section).
onResponse	How to process the API response and subsequent actions.

Table 12-3 describes the elements used in the OnResponse section to handle API responses and determine subsequent actions.

Table 12-3 OnResponse Section

Element	Description

environment	Specifies the environment variables to set based on the response. The value from **/importId** in the response is assigned to the environment variable **IMPORT_ID**. These variables can later be accessed in requests using the **{{ENV:VARIABLE_NAME}}** syntax.
	`"onResponse": {`
	`  "environment": {`
	`  "IMPORT_ID": "/importId"`
	`  }`
	`}`

statusMapping	Maps HTTP status codes to specific actions, such as ABORT. In this case, if a 5xx error is encountered, the transaction is halted and terminated.
	`"statusMapping": {`
	`  "5**": ["ABORT"]`
	`}`

retry	Configures automatic retries for a request.
	• **maxRetries**: The maximum number of retry attempts (integer).
	• **interval**: The time interval (in milliseconds) between retries.
	`"retry": {`
	`  "maxRetries": "5",`
	`  "interval": "10000"`
	`}`

maxRetrics	Specifies the maximum number of retry attempts.

Table 12-3 (Cont.) OnResponse Section

Element	Description
dataMapping	Specifies conditions based on values in the API response. • **path**: A specific field in the JSON response, such as **/importTaskItem/taskFinished**. • **equals**: The value to compare against (string). • **then**: An array of actions to take when the condition is true, such as retry the request if the task is not finished. `"dataMapping": [ { "path": "/importTaskItem/taskFinished", "equals": "true", "then": [] }, { "path": "/importTaskItem/taskFinished", "equals": "false", "then": ["RETRY"] } ]`

Table 12-4 describes the elements required to define individual API requests that can be referenced within a workflow.

Table 12-4 Requests Section

Element	Description
id	A unique identifier for the request.
method	The HTTP method, such as **GET**, **POST**, **PUT**, or **DELETE**.
uri	The URI or endpoint for the API request. The **{{ENV:VARIABLE_NAME}}** syntax injects values from environment variables.
headers	Any required HTTP headers.
payload	The data to be sent in the request body (for methods like **POST**).
multipartForm	Defines files to be uploaded as part of the request. The **FILE::** prefix indicates sending a file from a local path.

Managing a Helm Release

After you install a Helm chart, Kubernetes manages all of its objects and deployments. All pods created through **oc-cn-ocomc** are wrapped in a Kubernetes controller, which creates and manages the pods and performs health checks. For example, if a node fails, a controller can automatically replace a pod by scheduling an identical replacement on a different node.

Administrators can perform these maintenance tasks on a Helm chart release:

- Tracking a Release's Status
- Updating a Release
- Checking a Release's Revision

- Rolling Back a Release to a Previous Revision

Tracking a Release's Status

When you install a Helm chart, it creates a release. A release contains Kubernetes objects, such as ConfigMap, Secret, Deployment, Pod, PersistentVolume, and so on. Not every object is up and running immediately. Some objects have a start delay, but the Helm install command completes immediately.

To track the status of a release and its Kubernetes objects, enter this command:

```
helm status ReleaseName -n Namespace
```

where:

- *ReleaseName* is the name you assigned to this installation instance.
- *NameSpace* is the namespace in which the Offline Mediation Controller Kubernetes objects reside.

Updating a Release

To update any key value after a release has been created, enter this command. This command updates or re-creates the impacted Kubernetes objects, without impacting other objects in the release. It also creates a new revision of the release.

```
helm upgrade ReleaseName oc-cn-ocomc-helm-chart --values OverridingValueFile
--values NewOverridingValueFile -n Namespace
```

where:

- *ReleaseName* is the name you assigned to this installation instance.
- *OverridingValueFile* is the path to the YAML file that overrides the default configurations in the **oc-cn-ocomc/values.yaml** file.
- *NewOverridingValueFile* is the path to the YAML file that has updated values. The values in this file are newer than those defined in **values.yaml** and *OverridingValueFile*.
- *Namespace* is the namespace in which the Offline Mediation Controller Kubernetes objects reside.

Checking a Release's Revision

Helm keeps track of the revisions you make to a release. To check the revision for a particular release, enter this command:

```
helm history ReleaseName -n Namespace
```

where:

- *ReleaseName* is the name you assigned to this installation instance.
- *Namespace* is the namespace in which the Offline Mediation Controller Kubernetes objects reside.

Rolling Back a Release to a Previous Revision

To roll back a release to any previous revision, enter this command:

```
helm rollback ReleaseName RevisionNumber -n Namespace
```

where:

- *ReleaseName* is the name you assigned to this installation instance.
- *RevisionNumber* is the value from the Helm history command.
- *Namespace* is the namespace in which the Offline Mediation Controller Kubernetes objects reside.

Rolling Back an Offline Mediation Controller Cloud Native Upgrade

If you encounter errors after upgrading, you can roll back to a previous version of Offline Mediation Controller.

The following procedure assumes that you have upgraded Offline Mediation Controller from 12.0 Patch Set 5 (Revision 1), to 12.0 Patch Set 6 (Revision 2), and then to 15.1 (Revision 3). To roll back your upgrade from 15.1 to 12.0 Patch Set 6, you would do this:

1. Check the revision history of the Offline Mediation Controller release:

   ```
   helm history ReleaseName -n Namespace
   ```

   You should see something similar to this:

   ```
   REVISION   UPDATED                    STATUS        CHART
   APP                 VERSION        DESCRIPTION
   1          Thu May 30 07:12:46 2030   superseded    oc-cn-ocomc-helm-
   chart    12.0.0.5.0     Initial install
   2          Thu May 30 08:32:09 2030   superseded    oc-cn-ocomc-helm-
   chart    12.0.0.6.0     Upgraded successfully
   3          Thu May 30 09:50:00 2030   deployed      oc-cn-ocomc-helm-
   chart    15.1.0.0.0     Upgraded successfully
   ```

2. Roll back the release to Offline Mediation Controller 12.0 Patch Set 6:

   ```
   helm rollback ReleaseName 2 -n BrmNamespace
   ```

   If successful, you will see this:

   ```
   Rollback was a success! Happy Helming!
   ```

3. Check the revision history of the Offline Mediation Controller release:

   ```
   helm history ReleaseName -n BrmNamespace
   ```

If successful, you should see something similar to this:

```
REVISION   UPDATED                    STATUS        CHART
APP                VERSION       DESCRIPTION
1          Thu May 30 07:12:46 2030   superseded    oc-cn-ocomc-helm-
chart    12.0.0.5.0     Initial install
2          Thu May 30 08:32:09 2030   superseded    oc-cn-ocomc-helm-
chart    12.0.0.6.0     Upgraded successfully
3          Thu May 30 09:50:00 2030   superseded    oc-cn-ocomc-helm-
chart    15.1.0.0.0     Upgraded successfully
4          Thu May 30 11:25:00 2030   deployed      oc-cn-ocomc-helm-
chart    12.0.0.6.0     Roll back to 2
```

Integrating Oracle Unified Directory with Offline Mediation Controller Cloud Native

After verifying the Oracle Unified Directory deployment, follow these steps to integrate it with Offline Mediation Controller cloud native:

1. Open a terminal session within the running pod:

   ```
   kubectl exec -it oud-ds-rs-0 -n oudns -- /bin/bash
   ```

2. Inside the Oracle Unified Directory container, create a temporary directory (*tempdir*) to hold configuration files.
3. From the Offline Mediation Controller installation directory, copy the **oudConfig** and **populateDir.ldif** files to *tempdir*.
4. Inside *tempdir*, create a file named **populateDirTemp.ldif**. This file updates the user information in Oracle Unified Directory to match the Offline Mediation Controller requirements. Add the following content:

   ```
   dn: uid=Admin,ou=People,dc=ocomcexample.com
   changetype: modify
   replace: userpassword
   userpassword: adminpassword
   ```

5. Run the *tempdir*/**oudConfig** script with the Oracle Unified Directory container:

   ```
   sh oudConfig -i oud_instance_path -h oud_host -p oud_admin_port -d "oud_binddn" -w
   oud_admin_password -b "dc=ocomcexample.com" -l oud_ldapport
   ```

   > **Note**
   >
   > If the command fails with a "host not found" error, replace the host name with **localhost** in both the above and below commands.

6. Locate the **values.yaml** file in your Offline Mediation Controller installation directory, such as **/scratch/username/ocomc**. Under the **ocomcCore/lcmc.configEnv** section, add the following fields, adjusting values if necessary to match your setup:

   ```
   adminsvrAuthMode: true

   adminsvrAuthuser: "true"

   adminsvrLdapurl: "ldap://oud-ds-rs-lbr-ldap.oudns.svc.cluster.local:1389"
   ```

```
oudRootUserDn: cn=Directory Manager

oudPath: /u01/oracle/user_projects/oud-ds-rs-0/OUD

oudLdapPort: 1389

oudBaseDn: dc=ocomcexample.com

adminConnectPort: 4444

hostName: oud-ds-rs-0.oud-ds-rs.oudns.svc.cluster.local
```

> **Note**
>
> Upgrade the Offline Mediation Controller Helm installation after configuring the **values.yaml** file.

7. Log in to Offline Mediation Controller through Administration Client.

Common Problems and Their Solutions

Although Offline Mediation Controller is designed to be trouble free, you might encounter a problem. You can view descriptions of some common problems along with their solutions:

- Problem: NullPointerException While Creating IMS CC Cartridge in Offline Mediation Controller GUI

Problem: NullPointerException While Creating IMS CC Cartridge in Offline Mediation Controller GUI

After installing the IMS CDF/CGF Cartridge Pack, attempting to create the Diameter CC from the Offline Mediation Controller GUI results in a **NullPointerException**. The output displayed in the console window includes the following errors:

```
NodeInfoGUI.getConfigGUI() : Unable to instantiate config GUI for node of
type: EI : Diameter
ClassNotFoundException caught :
com.metasolv.nm.diameter.cc.DiameterEINodeConfigGUI
...
Exception in thread "AWT-EventQueue-0" java.lang.NullPointerException
at
com.nt.udc.admin.client.gui.nodegui.NodeInfoGUI.allFieldsValid(NodeInfoGUI.jav
a:210)
at
com.nt.udc.admin.client.gui.nodegui.NodeGUIFrame.actionPerformed(NodeGUIFrame.
java:541)
at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:2022)
```

Possible Cause

The **cartridges** folder in the Offline Mediation Controller GUI Virtual Machine (VM) contains unrelated or incompatible JAR files. These interfere with class loading during cartridge creation, resulting in GUI initialization failures and runtime exceptions.

Solution

1. Ensure that the *OMC_home*/**cartridges** directory of the Offline Mediation Controller GUI VM contains only the required JAR files for the IMS, Syslog, and Netflow cartridge packs. Remove any unnecessary files.

   For more information, see "List of Cartridge Packs and JAR Files" in Offline Mediation Controller Cartridge Packs.
2. Restart the pods to apply the changes.
3. Reattempt the cartridge creation using the Offline Mediation Controller GUI. The process should now complete without errors.

13

Deploying into Oracle Cloud Infrastructure

Learn how to deploy Oracle Communications Offline Mediation Controller cloud native services into Oracle Cloud Infrastructure.

Topics in this document:

Deploying into Oracle Cloud Infrastructure

Oracle Cloud Infrastructure is a set of complementary cloud services that enable you to run a wide range of applications and services in a highly available hosted environment. It offers high-performance compute capabilities (as physical hardware instances) and storage capacity in a flexible overlay virtual network that is securely accessible from your on-premises network. Among many of its services, the Offline Mediation Controller cloud native deployment is tested in an Oracle Cloud Infrastructure environment using its database and container engine for Kubernetes services on a bare metal instance.

Deploying the Offline Mediation Controller cloud native services into Oracle Cloud Infrastructure involves these high-level steps:

> **Note**
>
> These are the bare minimum tasks for deploying Offline Mediation Controller cloud native services in Oracle Cloud Infrastructure. Your steps may vary from the ones listed below.

1. Sign up for Oracle Cloud Infrastructure.
2. Create a Kubernetes cluster and deselect the **Tiller (Helm) Enabled** option. The version of Helm used by Oracle Cloud Infrastructure isn't compatible with the Offline Mediation Controller cloud native software requirements.
3. Install and configure the Oracle Cloud Infrastructure Command Line Interface (CLI).

   CLI is a small footprint tool that you can use on its own or with the Console to complete OCI tasks. It's needed here to download the **kubeconfig** file.
4. Install and configure **kubectl** on your system to perform operations on your cluster in Oracle Cloud Infrastructure.
5. The **kubeconfig** file (by default named **config** and stored in the **$HOME/.kube** directory) provides the necessary details to access the cluster using **kubectl** and the Kubernetes Dashboard.

   Download **kubeconfig** to access your cluster on Oracle Cloud Infrastructure by entering this command:

   ```
   oci ce cluster create-kubeconfig --cluster-id ClusterId --file $HOME/.kube/
   config --region RegionId
   ```

where *ClusterId* is the Oracle Cloud Identifier (OCID) of the cluster, and *RegionId* is the region identifier such as us-phoenix-1 and us-ashburn-1.

6. Set the **$KUBECONFIG** environment variable to the downloaded **kubeconfig** file by entering this command:

   ```
   export KUBECONFIG=$HOME/.kube/config
   ```

7. Verify access to your cluster. You can enter this command and then match the output Internal IP Addresses and External IP Addresses against the nodes in your cluster in the Oracle Cloud Infrastructure Console.

   ```
   kubectl get node -o wide
   ```

8. Download and configure Helm in your local system. To install Tiller on your cluster in Oracle Cloud Infrastructure, enter this command:

   ```
   helm init
   ```

9. If you are using a password-protected registry for Docker images, Kubernetes can't pull the images unless the authentication details are provided.

   There are many ways to enable Kubernetes to pull images from a password-protected Docker registry. For example, you could do this on each worker node:

   a. Log in to the Docker registry by entering this command:

      ```
      docker login -u UserName RepoHost:RepoPort
      ```

   b. Copy the **config.json** file where Docker has stored the authentication details to **/var/lib/kubelet**.

10. Place the Offline Mediation Controller cloud native Helm chart on your system where you have downloaded and configured **kubectl** and Helm. Then, follow the instructions in "Installing the Offline Mediation Controller Cloud Native Deployment Package".

14

Building Your Own Images

You can build your own images of Oracle Communications Offline Mediation Controller using the guidance provided in this chapter.

The Docker build commands in this chapter reference Dockerfile and related scripts as is from the **oc-cn-ocomc-docker-files-15.1.0.*x*.0.tgz** package. Ensure that you use your own version of Dockerfile and related scripts before running the build command.

Topics in this document:

- Building Offline Mediation Controller Images

Sample Dockerfiles included in the Offline Mediation Controller cloud native deployment package (**oc-cn-ocomc-docker-files-15.1.0.*x*.0.tgz**) are examples that depict how default images are built for Offline Mediation Controller. If you want to build your own images, refer to the sample Dockerfiles shipped with the product as a reference. Create your own Dockerfiles and then build your images.

> ⚠ **Caution**
>
> The Dockerfiles and related scripts are provided for reference only. You can refer to them to build or extend your own Docker images. Support is restricted to core product issues only and no support will be provided for custom Dockerfiles and scripts.

Building Offline Mediation Controller Images

To build images for Offline Mediation Controller, unpack **oc-cn-ocomc-docker-files-15.1.0.*x*.0.tgz** to create the directory structure in **docker_files/**.

Building your own Offline Mediation Controller images involves these high-level steps:

1. You build the Offline Mediation Controller base image. See "Building the Offline Mediation Controller Base Image".
2. You build custom images for Offline Mediation Controller. See "Building Your Offline Mediation Controller Image".

Building the Offline Mediation Controller Base Image

All images from the Offline Mediation Controller cloud native deployment package use Oracle Linux, JDK 1.8, and a few utilities as the base image. Oracle Linux is available from Oracle Container Registry (http://container-registry.oracle.com). You can pull the image from it. To build the base Offline Mediation Controller image, do this:

1. Extract the Docker file package (**oc-cn-ocomc-docker-files-15.1.0.*x*.0.tgz**).

   ```
   tar xvzf oc-cn-ocomc-docker-files-15.1.0.x.0.tgz
   ```

2. Place the **jdk*.tar.gz** in the **docker_files/jdk/** directory.

3. Build the Offline Mediation Controller base image by entering this command from the **docker_files/jdk/** directory:

   `docker build -t oc-cn-oraclelinuxjdk:15.1.0.`*`x`*`.0 -f Dockerfile.jdk --build-arg PROXY=`*`ProxyHost`***`:`***`Port`*** **`.`**

   For example:

   ```
   docker build -t oc-cn-oraclelinuxjdk:15.1.0.x.0 -f Dockerfile.jdk --build-arg
   PROXY=http://www-proxy.example.com:80 .
   ```

Building Your Offline Mediation Controller Image

To build your Offline Mediation Controller image:

1. Update the Offline Mediation Controller base image (**oc-cn-oraclelinuxjdk:15.1.0.*x*.0**) in the **docker_files/OCOMC/Dockerfile** directory.
2. Move the Offline Mediation Controller 15.1*x*.0 package (**OCOMC-15.1.0.*x*.0_generic_full.jar**) to the **docker_files/OCOMC/container-scripts** directory.
3. Build the Offline Mediation Controller image by entering this command from the **docker_files/OCOMC/** directory:

   `docker build -t `*`Image`***`:`***`Tag`*** **`-f Dockerfile .`**

   For example:

   ```
   docker build -t oc-cn-ocomc:15.1.0.x.0 -f Dockerfile .
   ```

4. Tag and push the image to your private registry server, if required.