Oracle® Communications Billing and Revenue Management

Cloud Native System Administrator's Guide

Release 12.0
F42583-06
August 2023

Oracle Communications Billing and Revenue Management Cloud Native System Administrator's Guide, Release 12.0

F42583-06

Contents

Preface

This guide describes how to install and administer Oracle Communications Billing and Revenue Management (BRM) Cloud Native Deployment Option.

Audience

This document is intended for DevOps administrators and those involved in installing and maintaining an Oracle Communications Billing and Revenue Management (BRM) 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 customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

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

Part I

Basic System Administration of BRM Cloud Native

This part describes basic administration tasks in an Oracle Communications Billing and Revenue Management (BRM) cloud native system. It contains the following chapters:

1 Managing Pods in BRM Cloud Native

Learn how to manage the pods in your Oracle Communications Billing and Revenue Management (BRM) cloud native environment.

Topics in this document:

Note:

This documentation uses the **override-values.yaml** file name for ease of use, but you can name the file whatever you want.

Setting up Autoscaling of BRM Pods

You can use the Kubernetes Horizontal pod Autoscaler to automatically scale up or scale down the number of BRM pod replicas in your deployment based on a pod's CPU or memory utilization. For more information, see "Horizontal Pod Autoscaler" in the *Kubernetes Tasks* documentation.

In BRM cloud native deployments, the Horizontal Pod Autoscaler monitors and scales these BRM pods:

- cm
- dm-oracle
- dm-aq
- dm-eai
- dm-ifw-sync
- batch-controller
- rel-daemon
- realtime-pipe
- brm-rest-services-manager

To set up autoscaling for BRM pods:

1. Open your **override-values.yaml** file for **oc-cn-helm-chart**.
2. Enable the Horizontal Pod Autoscaler by setting the **ocbrm.isHPAEnabled** key to **true**.
3. Enable the modification of resource limits by setting the **ocbrm.isResourceLimitEnabled** key to **true**.

4. Specify how often, in seconds, the Horizontal Pod Autoscaler checks a BRM pod's memory usage and scales the number of replicas. To do so, set the **ocbrm.refreshInterval** key to the number of seconds between each check. For example, set it to **60** for a one-minute interval.
5. For each BRM pod, set these keys to the appropriate values for your system:
   - **ocbrm.***BRMPod***.resourceLimits.limitsCpu**: Set this to the maximum number of CPU cores that the pod can utilize.

     If the pod's CPU utilization exceeds this value, Kubernetes terminates the pod.
   - **ocbrm.***BRMPod***.resourceLimits.requestCpu**: Set this to the minimum number of CPU cores that must be available in a Kubernetes node to deploy a pod. The default is 2 for the cm pod, 3.5 for the dm_oracle pod, 2 for the rel_daemon pod, and 1 for all other pods.

     If the minimum CPU amount is not available, the pod is set to **Pending**.

     Note:

     Because the CM contains two containers, its node must have twice the minimum CPU available (that is, 2 * **requestCpu**) to deploy the cm pod.
   - **ocbrm.***BRMPod***.resourceLimits.limitsMemory**: Set this to the maximum amount of memory that a pod can utilize.

     If a pod's memory utilization exceeds this value, Kubernetes terminates the pod.
   - **ocbrm.***BRMPod***.resourceLimits.requestMemory**: Set this to the minimum amount of memory required for a Kubernetes node to deploy a pod.

     If the minimum amount is not available, the pod is set to **Pending** due to insufficient memory.
   - **ocbrm.***BRMPod***.hpaValues.minReplica**: Set this to the minimum number of pod replicas that can be deployed in a cluster.

     If a pod's utilization metrics drop below **targetCPU** or **targetMemory**, the Horizontal Pod Autoscaler scales down the number of pod replicas to this minimum count. If the number of pod replicas are already at the minimum, no changes are made.
   - **ocbrm.***BRMPod***.hpaValues.maxReplica:** Set this to the maximum number of pod replicas to deploy when scale up is triggered.

     If a pod's metrics utilization goes above **targetCPU** or **targetMemory**, the Horizontal Pod Autoscaler scales up the number of pods to this maximum count.
   - **ocbrm.***BRMPod***.hpaValues.targetCpu**: Set this to the percentage of **requestCpu** at which to scale up or scale down a pod.

     If a pod's CPU utilization exceeds **targetCpu**, the Horizontal Pod Autoscaler increases the pod replica count to **maxReplica**. If a pod's CPU utilization drops below **targetCpu**, the Horizontal Pod Autoscaler decreases the pod replica count to **minReplica**.

- **ocbrm.***BRMPod***.hpaValues.targetMemory**: Set this to the percentage of **requestMemory** at which to scale up or scale down a pod.

  If a pod's memory utilization exceeds **targetMemory**, the Horizontal Pod Autoscaler increases the pod replica count to **maxReplica**. If memory utilization drops below **targetMemory**, the Horizontal Pod Autoscaler decreases the pod replica count to **minReplica**.

6. Save and close your **override-values.yaml** file.
7. Run the **helm upgrade** command to update your Helm release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
n BrmNameSpace
```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

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 "Automatically Roll Deployments" in *Helm Chart Development Tips and Tricks*.

Restarting BRM Pods

You may occasionally need to restart a BRM pod, such as when an error occurs that you cannot fix or a pod is stuck in a terminating status. You restart a BRM pod by deleting it with **kubectl**.

To restart a BRM pod:

1. Retrieve the names of the BRM pods by entering this command:

   ```
   kubectl -n NameSpace get pods
   ```

   where *NameSpace* is the namespace in which Kubernetes objects for the BRM Helm chart reside.

   The following provides sample output:

   ```
   NAME                              READY    STATUS       RESTARTS    AGE
   cm-6f79d95887-lp7qs               1/1      Running      0           6d17h
   dm-oracle-5496bf8d94-vjgn7        1/1      Running      0           6d17h
   dm-kafka-d5ccf6dbd-l968b          1/1      Running      0           6d17h
   ```

2. Delete a pod by entering this command:

   ```
   kubectl delete pod PodName -n NameSpace
   ```

   where *PodName* is the name of the pod

   For example, to delete and restart the cm pod, you would enter:

   ```
   kubectl delete pod cm-6f79d95887-lp7qs -n NameSpace
   ```

2

Running Applications and Utilities Outside Pods

Learn how to run applications, utilities, and scripts on demand in Oracle Communications Billing and Revenue Management (BRM) cloud native without entering a pod by running configurator jobs and brm-apps jobs.

Topics in this document:

Running Load Utilities through Configurator Jobs

You can run BRM load utilities on demand without entering into a pod by running a configurator job. For a list of utilities supported by the configurator job, see "Supported Load Utilities for Configurator Jobs".

To run BRM load utilities through configurator jobs:

1. Update the **oc-cn-helm-chart/config_scripts/loadme.sh** script with the list of load utilities that you want to run. The input will follow this general syntax:

   ```
   #!/bin/sh

   cd runDirectory; utilityCommand configFile
   exit 0;
   ```

   where:

   - *runDirectory* is the directory from which to run the utility.
   - *utilityCommand* is the utility command to run at the command line.
   - *configFile* is the file name and path to any input files the utility requires.

2. Move any required input files to the **oc-cn-helm-chart/config_scripts** directory.

   If the input file is an XML file with an XSD path, modify the XML file to refer to the container path. If the XML has just an XSD file name, move the XSD file along with the XML file.

3. Enable the configurator job. In your **override-values.yaml** file for **oc-cn-helm-chart**, set **ocbrm.config_jobs.run_apps** to **true**:

   ```
   ocbrm:
      config_jobs:
         run_apps: true
   ```

4. Run the **helm upgrade** command to update the release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values
   OverrideValuesFile -n BrmNameSpace
   ```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

   The utilities specified in the **loadme.sh** script are run.

5. If the utility requires the CM to be restarted, do this:

   a. Update these keys in the **override-values.yaml** file for **oc-cn-helm-chart**:

      - **ocbrm.config_jobs.restart_count**: Increment the existing value by **1**
      - **ocbrm.config_jobs.run_apps**: Set this to **false**

   b. Update the Helm release again:

      ```
      helm upgrade BrmReleaseName oc-cn-helm-chart --values
      OverrideValuesFile -n BrmNameSpace
      ```

Running pin_bus_params and load_pin_device_state

This example shows how to set up the configurator job to run the **pin_bus_params** and **load_pin_device_state** utilities.

To run **pin_bus_params** and then run **load_pin_device_state**:

1. Add the following lines to the **oc-cn-helm-chart/config_scripts/loadme.sh** script:

   ```
   #!/bin/sh

   cd /oms/sys/data/config; pin_bus_params -v /oms/load/
   bus_params_billing_flow.xml
   cd /oms/sys/data/config; load_pin_device_state -v /oms/sys/data/
   config/pin_device_state_num
   exit 0;
   ```

2. Move the **bus_params_billing_flow.xml** and **pin_device_state_num** input files to the **oc-cn-helm-chart/config_scripts** directory.

3. In the **override-values.yaml** file for **oc-cn-helm-chart**, set **ocbrm.config_jobs.run_apps** to **true**.
4. Run the **helm upgrade** command to update the release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
   n BrmNameSpace
   ```

5. Restart the CM because it is required by **pin_bus_params**.
   a. Set these keys in the **override-values.yaml** file:
      - **ocbrm.config_jobs.restart_count**: Increment the existing value by 1
      - **ocbrm.config_jobs.run_apps**: Set this to **false**
   b. Update the Helm release again:

      ```
      helm upgrade BrmReleaseName oc-cn-helm-chart --values
      OverrideValuesFile -n BrmNameSpace
      ```

Running Load Utilities on Multischema Systems

When you use the configurator job to load configuration data into a multischema system, you load the configuration data into the primary schema.

To load configuration data on a multischema system:

1. Update the **oc-cn-helm-chart/config_scripts/loadme.sh** script with the list of load utilities that you want to run. The input will follow this general syntax:

   ```
   #!/bin/sh

   cd runDirectory; utilityCommand configFile
   exit 0;
   ```

2. Move any required input files to the **oc-cn-helm-chart/config_scripts** directory.

   If the input file is an XML file with an XSD path, modify the XML file to refer to the container path. If the XML has just an XSD file name, move the XSD file along with the XML file.
3. Enable the configurator job, and disable multischema in the configurator job.

   In your **override-values.yaml** file for **oc-cn-helm-chart**, set these keys:

   ```
   ocbrm:
      config_jobs:
         run_apps: true
         isMultiSchema: false
   ```

4. Run the **helm upgrade** command to update the BRM Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
   n BrmNameSpace
   ```

   The utilities specified in the **loadme.sh** script are run.

5. If the utility requires the CM to be restarted, do this:
   a. Update these keys in the **override-values.yaml** file for **oc-cn-helm-chart**:
      - **ocbrm.config_jobs.restart_count**: Increment the existing value by **1**
      - **ocbrm.config_jobs.run_apps**: Set this to **false**
   b. Update the BRM Helm release again:

      ```
      helm upgrade BrmReleaseName oc-cn-helm-chart --values
      OverrideValuesFile -n BrmNameSpace
      ```

Running Applications and Utilities through brm-apps Jobs

You can run applications and utilities on demand without entering a pod through a brm-apps job. For a list of utilities and applications supported by the brm-apps job, see "Supported Utilities and Applications for brm-apps Jobs".

To run BRM applications through a brm-apps job:

1. Update the **oc-cn-helm-chart/brmapps_scripts/loadme.sh** script to include the applications and utilities that you want to run. The input will follow this general syntax:

   ```
   #!/bin/sh

   cd runDirectory; utilityCommand configFile
   exit 0;
   ```

   where:
   - *runDirectory* is the directory from which to run the application or utility.
   - *utilityCommand* is the utility or application command to run at the command line.
   - *configFile* is the file name and path to any input files the application or utility requires.
2. Move any required input files to the **oc-cn-helm-chart/brmapps_scripts** directory.
3. Enable the brm-apps job. In your **override-values.yaml** file for **oc-cn-helm-chart**, set **ocbrm.brm_apps.job.isEnabled** to **true**.
4. If you are running a multithreaded application (MTA), configure the performance parameters in your **override-values.yaml** file. For more information, see "Configuring MTA Performance Parameters".
5. Run the **helm upgrade** command to update the BRM Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values
   OverrideValuesFile -n BrmNameSpace
   ```

   where:
   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.

- *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
- *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

The applications and utilities specified in the **loadme.sh** script are run.

Configuring MTA Performance Parameters

You can configure the performance of multithreaded (MTA) applications, such as **pin_bill_accts** and **pin_export_price**, outside of the Kubernetes cluster. To do so, you edit these MTA-related keys in your **override-values.yaml** file for **oc-cn-helm-chart**:

- **mtaChildren**: Governs how many child threads process data in parallel. Each child thread fetches and processes one account from the queue before it fetches the next one.

  You can increase the number of child threads to improve application performance when the database server remains under-utilized even though you have a large number of accounts. If you increase the number of children beyond the optimum, performance suffers from context switching. This is often indicated by a higher system time with no increase in throughput. Performance is best when the number of children is nearly equal to the number of DM back ends and most back ends are dedicated to processing transactions.
- **mtaPerBatch**: Specifies the number of payment transactions that the **pin_collect** utility sends to dm_fusa in a batch. For example, if you have 20,000 payments to process and the **mtaPerBatch** key is set to 5000, the **pin_collect** utility would send four batches to dm_fusa (with each batch containing 5,000 payment transactions).

  Note:

  This key impacts the performance of the **pin_collect** application only. It has very little impact on other applications.
- **mtaPerStep**: Specifies how much data to store in dm_oracle when the application is performing a step search. It does not have a large impact on performance, but it does govern memory usage in dm_oracle. It also prevents BRM from using all of its memory for one large search.

  A 64-bit dm_oracle can use reasonably large values. A typical **mtaPerStep** value for invoice utilities would be between 10,000 and 50,000.
- **mtaFetchSize**: Specifies the number of account records to retrieve from the database and hold in memory before the utility starts processing them. In general, this value should be as large as possible to reduce the number of fetches from the database.

  The maximum possible fetch size depends on the complexity of the application's search results. When running applications on parent accounts (**pay_type 10001**), the **mtaFetchSize** value refers to the number of parent accounts to retrieve. For example, if you have 10,000 parent accounts and each account has an average of 50 children, you would set **mtaFetchSize** to 10,000 to retrieve all of the parent accounts. When running applications on only the children (**pay_type 10007**), you would set **mtaFetchSize** to 500,000 to retrieve all of the child accounts.

The MTA-related keys are nested under the **ocbrm.brm_apps.deployment.***DirectoryName* section in your **override-values.yaml** file:

```
ocbrm:
   brm_apps:
      deployment:
         DirectoryName
            mtaChildren: 5
            mtaPerBatch: 500
            mtaPerStep: 1000
            mtaFetchSize: 5000
```

where *DirectoryName* is the name of the directory in which the application resides, such as **pin_collections** for the **pin_collect** application or **pin_billd** for the **pin_bill_day** application. The directory name for each application is listed in "Supported Utilities and Applications for brm-apps Jobs".

If you modify these keys, you must run the **helm upgrade** command for the changes to take effect. See "Updating a Helm Release".

Running Custom Applications and Utilities through brm-apps

You can configure your BRM cloud native environment to run custom applications and utilities through a brm-apps job. To do so:

1. Identify all binaries, libraries, and configuration files required for your custom utility.
2. Layer the binaries and libraries on top of the brm-apps image.

   If any configuration needs to be done when the container starts, modify the **entrypoint.sh** script and layer it while building the brm-apps image.
3. Convert any configuration files into ConfigMaps.

Example: Running pin_billing_custom

This example shows how to set up a custom utility named **pin_billing_custom** to run through a brm-apps job.

1. Convert the utility's **pin.conf** configuration file into a ConfigMap, which will be mounted inside the container in the path **/oms/custom_pin.conf**.

   For information about converting a **pin.conf** file into a ConfigMap, refer to any **configmap_pin_conf** file in the **oc-cn-helm-chart/template** directory.
2. Copy the **entrypoint.sh** script from the **oc-cn-docker-files** directory to the **/oms** directory.
3. In the **entrypoint.sh** script, under the brm-apps section, add a line for copying the **/oms/custom_pin.conf** file to the **apps/pin_billing_custom** directory.
4. Layer the **pin_billing_custom** binary, the modified **entrypoint.sh** script, and the **apps/pin_billing_custom** directory into a brm-apps image by creating this **dockerfile_custom_brm_apps** file:

Note:

Ensure that the scripts and binaries have execute permission.

```
vi dockerfile_custom_brm_apps
    FROM brm_apps:12.0.0.x.0
    USER root
    COPY pin_billing_custom /oms/bin/
    RUN mkdir /oms/apps/pin_billing_custom
    COPY entrypoint.sh /oms/
    RUN chown -R omsuser:oms /oms/bin/pin_billing_custom  /oms/apps/
pin_billing_custom  /oms/entrypoint.sh && \
      chmod -R 755 /oms/bin/pin_billing_custom  /oms/apps/
pin_billing_custom  /oms/entrypoint.sh
    USER omsuser
```

5. Build the Docker image by entering this command:

```
docker build -t brm_apps:12.0.0.x.0-custom -f dockerfile_custom_brm_apps .
```

6. Update the **oc-cn-helm-chart/template/brm_apps_job.yaml** file to mount the ConfigMap in the container:

```
volumeMounts:
- name: brm-apps-custom-pin-conf
  mountPath: /oms/custom_pin.conf
    subPath: pin.conf
volumes:
- name: brm-apps-custom-pin-conf
  configMap:
    name: brm-apps-custom-conf
```

7. Add the **pin.conf** file entries to the ConfigMap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: brm-apps-custom-conf
  namespace: {{ .Release.Namespace }}
  labels:
    application: {{ .Chart.Name }}
data:
  pin.conf: |

#************************************************************************
   pin.conf content here

#************************************************************************
```

8. Update the Docker image tag in your **override-values.yaml** file.

Running Business Operations through pin_job_executor Service

You can run business operations, such as billing and payment collections, in BRM cloud native environments in the following ways:

- Using the brm-apps pod to run the **pin_job_executor** utility as a service named **pje** in the pje pod. The **pje** service processes business operations jobs or runs the **pin_virtual_time** utility. The **pin_job_executor** service port is exposed as ClusterIP, and the host name and service name of the brm-apps pod is **pje**.
- Using the boc pod or another client application to call the PCM_OP_JOB_EXECUTE opcode. In this case, the opcode request goes to the CM, which connects to the pje pod through the **pin_job_executor** service. The **pin_job_executor** service processes the opcode request and calls the appropriate BRM application.

  For more information, see "Job Opcode Workflows" in *BRM Opcode Guide*.

3

Exposing Directories as ConfigMaps

Learn how to expose any directory as a ConfigMap in your Oracle Communications Billing and Revenue Management (BRM) cloud native environment. This decouples environment-specific configuration from your container images.

Topics in this document:

- Configuring a CM ConfigMap Directory
- Configuring an EAI Publisher ConfigMap

Configuring a CM ConfigMap Directory

You can expose the CM directory as a ConfigMap so that your BRM cloud native deployment can access custom input files.

To expose the **oc-cn-helm-chart/cm_custom_files** directory as a ConfigMap, do this:

1. Move your custom input files to the **oc-cn-helm-chart/cm_custom_files** directory.
2. In your **override-values.yaml** file for **oc-cn-helm-chart**, set these keys:
   - **ocbrm.cm.custom_files.enable**: Set this to **true**.
   - **ocbrm.cm.custom_files.path**: Set this to the location of your custom input files, such as **/oms/load**.
3. In the CM ConfigMap file (**configmap_pin_conf_cm.yaml**), set the path to your custom input files.
4. Run the **helm upgrade** command to update your Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
   n BrmNameSpace
   ```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

Exposing the taxcode_map File Example

This example shows how to expose the **taxcodes_map** file using the CM ConfigMap.

1. Edit the **taxcodes_map** file and move it to the **oc-cn-helm-chart/cm_custom_files** directory.

2. Set these keys in your **override-values.yaml** file for **oc-cn-helm-chart**:

   ```
   ocbrm.cm.custom_files.enable=true
   ocbrm.cm.custom_files.path=/oms/load
   ```

3. In the CM ConfigMap (**configmap_pin_conf_cm.yaml**), set the path to the **taxcodes_map** file:

   ```
   - fm_rate taxcodes_map /oms/load/taxcodes_map
   ```

4. Run the **helm upgrade** command to update your Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values
   OverrideValuesFile -n BrmNameSpace
   ```

Configuring an EAI Publisher ConfigMap

The payload configuration file used by the EAI Java Server (eai_js) process can be loaded as a Kubernetes ConfigMap and consumed by **eai_js** from **/oms/payload**.

The following payload configuration files are included in the BRM Helm chart and can be mounted as a Kubernetes ConfigMap:

- **payloadconfig_ece_sync.xml**
- **payloadconfig_ifw_sync.xml**
- **payloadconfig_kafka_sync.xml**

By default, the EAI Java Server uses the **payloadconfig_ifw_sync.xml** file. To configure it to use a different payload configuration XML file, do the following:

1. Configure your payload configuration file.
2. Copy your payload configuration file to the **oc-cn-helm-chart/payload_xml** directory.
3. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **ocbrm.eai_js.deployment.eaiConfigFile** key to the name of your payload configuration file.
4. Run the **helm upgrade** command to update your Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values
   OverrideValuesFile -n BrmNameSpace
   ```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

4

Managing a Helm Release

Learn how to manage your Helm releases in Oracle Communications Billing and Revenue Management (BRM) cloud native.

Topics in this document:

About Helm Releases

After you install a Helm chart, Kubernetes manages all of its objects and deployments. All pods created through **oc-cn-helm-chart** and **oc-cn-ece-helm-chart** 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.

As part of maintaining a Helm release, administrators can check a release's status or revision, update a release, or roll back the 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 ConfigMaps, Secrets, deployments, and pods. 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, run this command:

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

where:

- *ReleaseName* is the name you assigned to this installation instance.
- *NameSpace* is the name space in which the BRM Kubernetes objects reside.

Updating a Helm Release

To update any **override-values.yaml** key value after a release has been created, run the following 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.

Note:

- Before updating the release, you can check for issues by running the **helm upgrade** command and appending the **--dry-run** parameter.
- If you are updating an ECE Patch Set 7 or earlier release, you must delete any configLoader jobs before you do the Helm update.

```
helm upgrade ReleaseName Chart --values OverrideValuesFile --values
NewOverrideValuesFile -n Namespace
```

where:

- *ReleaseName* is the name you assigned to this installation instance.
- *Chart* is the name and location of the chart: **oc-cn-helm-chart** for BRM cloud native services, **oc-cn-ece-helm-chart** for ECE cloud native services, or **oc-cn-init-db-helm-chart** for initializing the BRM database schema.
- *OverrideValuesFile* is the path to the YAML file that overrides the default configurations in the **values.yaml** file.
- *NewOverrideValuesFile* 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 *OverrideValuesFile*.
- *Namespace* is the name space in which the BRM 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, run this command:

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

where:

- *ReleaseName* is the name you assigned to this installation instance.
- *Namespace* is the name space in which the BRM Kubernetes objects reside.

Rolling Back a Release To a Previous Revision

To roll back a release to any previous revision, run 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 name space in which the BRM Kubernetes objects reside.

5

Managing Passwords in BRM Cloud Native

Learn how to manage passwords in your Oracle Communications Billing and Revenue Management (BRM) cloud native environment.

Topics in this document:

- Rotating the BRM Root Password
- Rotating the BRM Root Key
- Rotating the BRM Password

Rotating the BRM Root Password

The BRM root password is the password of service with the login ID **root.0.0.0.1**, which is used by all clients to connect to the Connection Manager (CM). For security reasons, you should change this password at regular intervals.

When you change the BRM root password, it impacts all clients that connect to the CM service: Billing Care, the Billing Care REST API, Business Operations Center, and BRM Web Services. Therefore, you must provide the new password to your clients so they can continue to connect to the CM service.

This shows the procedure for changing the current BRM root password (*RootPassword1*) to a new root password (*RootPassword2*) and then providing *RootPassword2* to all of your clients:

1. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the keys in Table 5-1.

Table 5-1 Initial Key Values

Key	Value	Description
ocbrm.rotate_password	**true**	Specify that the password is being changed.
ocbrm.new_brm_root_password	*RootPassword2*	Set a new password for the **root.0.0.0.1** service.
ocbrm.cm.deployment.load_localized	**0**	Specify to not reload the localized strings into the database. This was already done during installation.
ocbc.bc.wop.serverStartPolicy	**NEVER**	Specify to shut down the WebLogic servers for Billing Care.
ocbc.bcws.wop.serverStartPolicy	**NEVER**	Specify to shut down the WebLogic servers for the Billing Care REST API.

Table 5-1 (Cont.) Initial Key Values

Key	Value	Description
ocboc.boc.wop.serverStartPolicy	**NEVER**	Specify to shut down the WebLogic servers for Business Operations Center.

2. Specify to shut down the WebLogic servers for BRM Web Services. In the **oc-cn-helm-chart/templates/domain_brm_wsm.yaml** file, set the **serverStartPolicy** key to **NEVER**.
3. Run the **helm upgrade** command to update the Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values
   OverrideValuesFile -n BrmNameSpace
   ```

   where:

   - *BrmReleaseName* is the release name assigned to your existing **oc-cn-helm-chart** installation.
   - *OverrideValuesFile* is the file name and path of your **override-values.yaml** file.
   - *BrmNameSpace* is the name space for your existing BRM deployment.

   Updating the release changes the password for service **root.0.0.0.1**, spins off new pods for the CM and a few other services, and stops services for Billing Care, the Billing Care REST API, Business Operations Center, and BRM Web Services.
4. Specify to turn off the password rotation indicator and to update the password. In the same **override-values.yaml** file, set the keys in Table 5-2.

   Table 5-2 Turn Off Password Rotation

Key	Value	Description
ocbrm.rotate_password	**false**	Turn off password rotation. This specifies that the password is not being changed.
ocbrm.brm_root_password	*RootPassword2*	Provide the updated password for the **root.0.0.0.1** service.

5. Update the password in the **Infranet.properties** file and **wallet** for Billing Care, the Billing Care REST API, and Business Operations Center by either reinstalling **oc-cn-op-job-helm-chart** or updating the wallet in-place in the persistent volume (PV).

   To reinstall **oc-cn-op-job-helm-chart**, do this:

   a. Delete the release of **oc-cn-op-job-helm-chart**:

      ```
      helm delete --namespace NameSpace OpJobReleaseName
      ```

      where *OpJobReleaseName* is the name of the **oc-cn-op-job-helm-chart** release.

b. Clean up the domain home from the PV for Billing Care, Billing Care REST, and Business Operations Center:

```
rm -rf DomainHome/domains/DomainUID
```

where:

- *DomainHome* is the location specified in the **domainVolHostPath** key under groups **ocbc.bc.wop**, **ocbc.bcws.wop**, and **ocboc.boc.wop**.
- *DomainUID* is the domain name specified in the **domainUID** key under groups **ocbc.bc.wop**, **ocbc.bcws.wop**, and **ocboc.boc.wop**. Typically, the defaults are **billingcare-domain**, **bcws-domain**, and **boc-domain** respectively.

c. Clean up the application home from the PV for Billing Care and the Billing Care REST API:

```
rm -rf ApplicationHome/billingcare
```

where *ApplicationHome* is the location specified in the **appVolHostPath** key under groups **ocbc.bc.wop** and **ocbc.bcws.wop**.

d. Clean up the application home from the PV for Business Operations Center:

```
rm -rf ApplicationHome/BOC
```

where *ApplicationHome* is the location specified in the **appVolHostPath** key under group **ocboc.boc.wop**.

e. Install **oc-cn-op-job-helm-chart** again:

```
helm install OpJobReleaseName oc-cn-op-job-helm-chart --namespace NameSpace --
values OverrideValuesFile
```

Wait for the jobs to complete their tasks.

f. Delete the policy job for Billing Care, the Billing Care REST API, and Business Operations Center:

```
kubectl --namespace NameSpace delete job DomainUID-policy-job
```

where *DomainUID* is the domain name specified in the **domainUID** key under groups **ocbc.bc.wop**, **ocbc.bcws.wop**, and **ocboc.boc.wop** in the **override-values.yaml** file. Typically, the defaults are **billingcare-domain**, **bcws-domain**, and **boc-domain** respectively.

To update the wallet in-place in the PV, do this:

a. For Billing Care and the Billing Care REST API, update the password in the wallet by following the instructions in "Storing Configuration Entries in the Billing Care Wallet" in *BRM Security Guide*. The wallet for these clients is located at *ApplicationHome*/**billingcare/wallet/client**.

b. For Business Operations Center, update the password in the wallet by following the instructions in "Storing Configuration Entries in the Business Operations Center Wallet" in *BRM Security Guide*. The wallet for Business Operations Center is located at *ApplicationHome*/**BOC/wallet/client**.

where *ApplicationHome* is the location specified in the **appVolHostPath** key under groups **ocbc.bc.wop**, **ocbc.bcws.wop**, and **ocboc.boc.wop**.

6. Delete the PDC and PCC deployments:

```
kubectl --namespace NameSpace delete deploy pdc-deployment pcc-deployment
```

7. Specify to start the WebLogic servers for BRM Web Services. In the **oc-cn-helm-chart/templates/domain_brm_wsm.yaml** file, set the **serverStartPolicy** key to **IF_NEEDED**.
8. Update the release of **oc-cn-helm-chart** to bring up all client services with the updated CM connection details:

   ```
   helm upgrade --namespace NameSpace ReleaseName oc-cn-helm-chart --values
   OverrideValuesFile
   ```

9. Update the BRM root password in your ECE pods by doing this:
   a. Connect via JMX to any of the charging server (ecs) pods.
   b. Navigate to the BRM Connection MBean.
   c. Navigate to the **Operations** section.
   d. Enter the new BRM root password (*RootPassword2*) along with the existing wallet password in the **setPassword** method and then run it.
   e. Perform a test connection to validate that the connection is successful.
   f. Rebounce the brmgateway pods for the new password to take effect and for the connection pool to BRM to be re-created.

Rotating the BRM Root Key

You should rotate your root keys on a regular basis to increase security.

To rotate the BRM root key:

1. Ensure that the cm and dm_oracle pods are up and running.
2. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **ocbrm.root_key_rotate** key to **true**.
3. Run the **helm upgrade** command to update your Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values
   OverrideValuesFile -n BrmNameSpace
   ```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.
4. Restart the cm and dm_oracle pods.

If successful, the root key is rotated and a new root key is generated in the Oracle wallet (the **/oms/root-key-wallet/key_rotated** file mounted in the PVC). If a **key_rotated** file already exists in the PVC, you can update the Helm release to avoid rotating the root key again.

After you rotate the root key once, use one of the following methods to rotate the root key again:

- Rotating the Root Key Method 1
- Rotating the Root Key Method 2

Rotating the Root Key Method 1

One method for rotating the root key after you have rotated it once:

1. Delete the dm_oracle deployment.
2. Remove the **/oms/root-key-wallet/key_rotated** file from the PVC.
3. In your **override-values.yaml** file for **oc-cn-helm-chart**, ensure that the **ocbrm.root_key_rotate** key is set to **true**.
4. Run the **helm upgrade** command to update your Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
   n BrmNameSpace
   ```

Rotating the Root Key Method 2

Another method for rotating the root key after you have rotated it once:

1. Copy the Oracle wallet from the PVC to the **oc-cn-helm-chart/existing_wallet/** directory.
2. In your **override-values.yaml** file for **oc-cn-helm-chart**, do the following:
   - **ocbrm.root_key_rotate**: Set this key to **false**.
   - **ocbrm.existing_rootkey_wallet**: Set this key to **true**.
3. Run the **helm upgrade** command and ensure that the new dm_oracle pod is created:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
   n BrmNameSpace
   ```

4. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **ocbrm.root_key_rotate** key to **true**.
5. Delete the dm_oracle deployment.
6. Run the **helm upgrade** command again:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
   n BrmNameSpace
   ```

7. Restart the dm_oracle and cm pods.

Rotating the BRM Password

To rotate the BRM password, you must stop and then restart your pods.

To rotate the BRM password:

1. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **ocpdc.labels.isEnabled** key to **false**.

2. Run the **helm upgrade** command to update the Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values
   OverrideValuesFile -n BrmNameSpace
   ```

   where:

   - *BrmReleaseName* is the release name assigned to your existing **oc-cn-helm-chart** installation.
   - *OverrideValuesFile* is the file name and path of your **override-values.yaml** file.
   - *BrmNameSpace* is the name space for your existing BRM deployment.

3. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **ocpdc.labels.isEnabled** key to **true**.
4. Run the **helm upgrade** command to update the Helm release.

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values
   OverrideValuesFile -n BrmNameSpace
   ```

6

Managing Database Partitions

Learn how to organize your Oracle Communications Billing and Revenue Management (BRM) cloud native database by using partitioned tables.

Topics in this document:

- Converting Nonpartitioned Classes to Partitioned Classes
- Adding Partitions to Your Database

Converting Nonpartitioned Classes to Partitioned Classes

If you did not enable partitioning for one or more storable classes when you deployed BRM cloud native, you can do so after deployment. The partitioning conversion feature splits a storable class's table in the BRM database into the following partitions:

- **partition_migrate**: Holds all objects created *before* the nonpartitioned storable classes were converted to partitioned storable classes. The BRM purge utility, **partition_utils**, cannot purge objects in this partition. To purge them, you must develop your own tools based on sound Oracle database management principles.
- **partition_historic**: Holds *nonpurgeable events* created *after* the nonpartitioned storable classes were converted to partitioned storable classes. Nonpurgeable events should not be purged from the database.
- **partition_last**: A *spillover* partition that is not intended to store objects you want to purge or preserve. If you do not add purgeable partitions to your tables *before* BRM resumes generating objects, purgeable objects created after the upgrade are stored in this partition.

To convert nonpartitioned storable classes to partitioned storable classes, perform these tasks:

1. Add the following lines to the **oc-cn-helm-chart/brmapps_scripts/loadme.sh** script:

   ```
   #!/bin/sh

   cd /oms/apps/partition; perl partitioning.pl ClassName
   exit 0;
   ```

   where *ClassName* is the name of the storable class that you want to partition, such as **/product** or **/bill**.
2. The brm-apps-partition-cfg ConfigMap (**configmap_partition_cfg.yaml**) controls your conversion parameters, such as your database's name and the partition logging directory. If necessary, edit the parameters in the file and then run the **helm upgrade** command.
3. Enable the brm-apps job. In your **override-values.yaml** file for **oc-cn-helm-chart**, set **ocbrm.brm_apps.job.isEnabled** to **true**.
4. Run the **helm upgrade** command to update the BRM Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -n
   BrmNameSpace
   ```

where:

- *BrmReleaseName* is the release name for oc-cn-helm-chart and is used to track this installation instance.
- *OverrideValuesFile* is the file name and path to your override-values.yaml file.
- *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

The brm_apps job runs a series of partitioning scripts that perform the conversion.

Check the **log** and **pinlog** files in the directory specified by the $PARTITION_LOG_DIR parameter in your **configmap_partition_cfg.yaml** file. These log files show how long each script took to run and list any errors that occurred. If any errors are reported, fix them and rerun the script.

Adding Partitions to Your Database

You can add partitions to your database by using the **partition_utils** utility. For information about the utility's syntax and parameters, see "partition_utils" in *BRM System Administrator's Guide.*

To add partitions to the database in your BRM cloud native environment:

1. Stop the following BRM pods:
   - dm-oracle
   - cm
   - realtime-pipeline
   - batch-controller
   - rel-daemon
   - Other pods
2. Ensure that all jobs are stopped in your BRM cloud native environment. This includes Configurator jobs, brm-apps jobs, **ImportExportPricing** jobs, and **SyncPDC** jobs.
3. Create a restore point in your BRM database.

   For more information, see CREATE RESTORE POINT in Oracle Database SQL Language Reference.
4. Run the **partition_utils** utility in *test mode* to check the command for enabling delayed-event partitions:

   ```
   partition_utils -o enable -t delayed -c /event/delayed/session% -p
   ```

   The utility writes the operation's SQL statement to a **partition_utils.log** file without performing any action on the database.
5. Verify that the generated SQL statement is correct in the **partition_utils.log** file before proceeding.
6. Enable delayed-event partitions by running this command:

   ```
   partition_utils -o enable -t delayed -c /event/delayed/session%
   ```
7. Run the **partition_utils** utility in *test mode* to check the command for adding partitions for 12 months:

```
partition_utils -o add -t delayed -s StartDate -u month -q 12 -f -p
```

where *StartDate* specifies the starting date for the new partitions in the format *MMDDYYYY*. The start date must be the day after tomorrow or later. You cannot create partitions starting on the current day or the next day. For example, if the current date is January 1, the earliest start date for the new partition is January 3.

The utility writes the operation's SQL statement to a **partition_utils.log** file without performing any action on the database.

8. Verify that the generated SQL statement is correct in the **partition_utils.log** file before proceeding.
9. Add delayed-event partitions for 12 months by running this command:

   ```
   partition_utils -o add -t delayed -s StartDate -u month -q 12 -f
   ```

10. Restart any Configurator, brm-apps, **ImportExportPricing**, or **SyncPDC** jobs in your BRM cloud native environment.
11. Start the following BRM pods:
    - dm-oracle
    - cm
    - realtime-pipeline
    - batch-controller
    - rel-daemon

7

Improving Performance in BRM Cloud Native

Learn how to improve performance in your Oracle Communications Billing and Revenue Management (BRM) cloud native environment.

Topics in this document:

- Deploying the CM and DM Containers in the Same Pod
- Tuning Your Application Connection Pools
- Configuring Multiple Replicas of Batch Controller
- Deploying Paymentech Data Manager in HA Mode

Deploying the CM and DM Containers in the Same Pod

You can improve system performance by deploying the CM and Oracle DM containers in the same pod.

To deploy the CM and DM in the same pod:

1. In the **oc-cn-helm-chart/templates** directory, rename the **dm_oracle.yaml** file to **_dm_oracle.yaml**.
2. Copy the dm_oracle **containers** and **VolumeMounts** entries from the **oc-cn-helm-chart/templates/dm_oracle.yaml** file into the **oc-cn-helm-chart/templates/cm.yaml** file. For example:

```
containers:
- name: dm-oracle
  image: "{{ .Values.imageRepository }}
{{ .Values.ocbrm.dm_oracle.deployment.imageName }}:
{{ .Values.ocbrm.dm_oracle.deployment.imageTag }}"
  ports:
    - name: dm-pcp-port
      containerPort: 12950
  env:
  - name: ROTATE_PASSWORD
    value: "{{ .Values.ocbrm.rotate_password }}"
  {{ if eq .Values.ocbrm.rotate_password true }}
  - name: NEW_BRM_ROOT_PASSWORD
    valueFrom:
      secretKeyRef:
        name: oms-schema-password
        key: new_brm_root_password
  {{ end }}
  {{- if eq .Values.ocbrm.existing_rootkey_wallet true }}
  - name: BRM_WALLET
    value: "/oms/client"
  {{- end }}
  - name: USE_ORACLE_BRM_IMAGES
    value: "{{ .Values.ocbrm.use_oracle_brm_images }}"
  - name: TZ
    value: "{{ .Values.ocbrm.TZ }}"
  - name: NLS_LANG
    value: "{{ .Values.ocbrm.db.nls_lang }}"
  - name: PIN_LOG_DIR
    value: "/oms_logs"
  - name: TNS_ADMIN
    value: "/oms/ora_k8"
  - name: DM_DEBUG
    value: "{{ .Values.ocbrm.dm_oracle.deployment.dm_debug }}"
  - name: DM_DEBUG2
    value: "{{ .Values.ocbrm.dm_oracle.deployment.dm_debug2 }}"
  - name: DM_DEBUG3
    value: "{{ .Values.ocbrm.dm_oracle.deployment.dm_debug3 }}"
  - name: SERVICE_FQDN
    value: "localhost"
  {{ if eq .Values.ocbrm.cmSSLTermination true }}
  - name: ENABLE_SSL
    value: "0"
  {{ else }}
  - name: ENABLE_SSL
    valueFrom:
      configMapKeyRef:
        name: oms-common-config
        key: ENABLE_SSL
  {{ end }}
  - name: ORACLE_CHARACTERSET
    valueFrom:
      configMapKeyRef:
        name: oms-common-config
        key: ORACLE_CHARACTERSET
  - name: DM_ORACLE_SERVICE_PORT
    value: "12950"
  - name: OMS_SCHEMA_USERNAME
    valueFrom:
      configMapKeyRef:
        name: oms-common-config
        key: OMS_SCHEMA_USERNAME
  {{ if .Values.ocbrm.brm_crypt_key }}
  - name: BRM_CRYPT_KEY
    valueFrom:
      secretKeyRef:
        name: oms-schema-password
        key: brm_crypt_key
  {{ end }}
  - name: OMS_DB_SERVICE
    valueFrom:
      configMapKeyRef:
        name: oms-common-config
        key: OMS_DB_SERVICE
  - name: OMS_DB_ALIAS
    value: "pindb"
  - name: LOG_LEVEL
    valueFrom:
```

```
            configMapKeyRef:
              name: oms-common-config
              key: LOG_LEVEL
        - name: DM_NO_FRONT_ENDS
          valueFrom:
            configMapKeyRef:
              name: oms-dm-oracle-config
              key: DM_NO_FRONT_ENDS
        - name: DM_NO_BACK_ENDS
          valueFrom:
            configMapKeyRef:
              name: oms-dm-oracle-config
              key: DM_NO_BACK_ENDS
        - name: DM_SHM_BIGSIZE
          valueFrom:
            configMapKeyRef:
              name: oms-dm-oracle-config
              key: DM_SHM_BIGSIZE
        - name: DM_MAX_PER_FE
          valueFrom:
            configMapKeyRef:
              name: oms-dm-oracle-config
              key: DM_MAX_PER_FE
        - name: DM_SHM_SEGMENT_SIZE
          valueFrom:
            configMapKeyRef:
              name: oms-dm-oracle-config
              key: DM_SHM_SEGMENT_SIZE
        - name: DM_NO_TRANS_BE_MAX
          valueFrom:
            configMapKeyRef:
              name: oms-dm-oracle-config
              key: DM_NO_TRANS_BE_MAX
        - name: DM_STMT_CACHE_ENTRIES
          valueFrom:
            configMapKeyRef:
              name: oms-dm-oracle-config
              key: DM_STMT_CACHE_ENTRIES
        - name: DM_SEQUENCE_CACHE_SIZE
          valueFrom:
            configMapKeyRef:
              name: oms-dm-oracle-config
              key: DM_SEQUENCE_CACHE_SIZE
        - name: VIRTUAL_TIME_SETTING
          valueFrom:
            configMapKeyRef:
              name: oms-common-config
              key: VIRTUAL_TIME_SETTING
        - name: VIRTUAL_TIME_ENABLED
          valueFrom:
            configMapKeyRef:
              name: oms-common-config
              key: VIRTUAL_TIME_ENABLED
        - name: SHARED_VIRTUAL_TIME_FILE
          value: /oms/virtual_time/shared/pin_virtual_time_file
```

```
 - name: BRM_LOG_STDOUT
   value: "FALSE"
 - name: SYNC_PVT_TIME
   value: "{{ .Values.ocbrm.virtual_time.sync_pvt_time }}"
 imagePullPolicy: {{ .Values.ocbrm.imagePullPolicy }}
 terminationMessagePolicy: FallbackToLogsOnError
 livenessProbe:
   exec:
     command:
     - /bin/sh
     - -c
     - sh /oms/test/is_dm_ready.sh
   initialDelaySeconds: 10
   periodSeconds: 10
   failureThreshold: 50
 readinessProbe:
   exec:
     command:
     - /bin/sh
     - -c
     - sh /oms/test/is_dm_ready.sh
   initialDelaySeconds: 15
   periodSeconds: 10
   timeoutSeconds: 1
 volumeMounts:
 - name: secret-volume
   mountPath: /etc/secret
 {{- if eq .Values.ocbrm.existing_rootkey_wallet true }}
 - name: wallet-pvc
   mountPath: /oms/client
 {{- end }}
 - name: dm-oracle-pin-conf-volume
   mountPath: /oms/pin.conf.tmpl
   subPath: pin.conf
 - name: dm-oracle-tnsnames-ora-volume
   mountPath: /oms/ora_k8
 - name: oms-logs
   mountPath: /oms_logs
 - name: virtual-time-volume
   mountPath: /oms/virtual_time/shared

- name: dm-oracle-pin-conf-volume
  configMap:
    name: dm-oracle-pin-conf-config
- name: dm-oracle-tnsnames-ora-volume
  configMap:
    name: db-config
    items:
      - key: tnsnames.ora
        path: tnsnames.ora
      - key: sqlnet.ora
        path: sqlnet.ora
```

3. Copy the dm_oracle **annotations** entries from the **oc-cn-helm-chart/templates/dm_oracle.yaml** file into the **oc-cn-helm-chart/templates/cm.yaml** file. For example:

   ```
   annotations:
      configmap_pin_conf_dm_oracle.yaml
      configmap_env_dm_oracle.yaml
   ```

4. In the **cm-pin-conf-config** ConfigMap, update the **dm_pointer** entry to point to **localhost** rather than **dm-oracle**. For example:

   ```
   - cm dm_pointer databaseNumber ip localhost 12950
   ```

5. Run the **helm upgrade** command to update your Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
   n BrmNameSpace
   ```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

Tuning Your Application Connection Pools

You can improve an application's performance by tuning the number of threads that are available for its connection with the CM.

When the CM sends a request, it is assigned a thread from the application's connection pool for performing operations. When the CM completes its operation, the thread is returned to the pool.

If an incoming request cannot be assigned a thread immediately, the request is queued. The request waits for a thread to become available for a configurable period of time. If a thread does not become available during this time, an exception is thrown indicating that the request timed out.

To tune the number of threads in an application's connection pool:

1. Open the application's ConfigMap. For example:
   - For Web Services Manager with Tomcat, the **wsm-infranet-properties** ConfigMap.
   - For Web Services Manager with WebLogic Server, the **wsm-wl-infranet-properties** ConfigMap.
2. Edit the parameters shown in Table 7-1.

Table 7-1 Connection Pool Parameters

Entry	Description
infranet.connectionpool.minsize	The minimum number of threads that the application spawns when it starts. The default is **1**.
infranet.connectionpool.maxsize	The maximum number of threads that the application can spawn for accepting requests from the CM. The default is **8**.
infranet.connectionpool.timeout	The time, in milliseconds, that a connection request will wait in the pending request queue for a free thread before it times out. If a pending request is not assigned a thread during this time, an exception is thrown. The default is **30000**.
infranet.connectionpool.maxidletime	The time, in milliseconds, that an unused thread remains in the connection pool before it is removed. The default is **10000**. **Important**: If the value is set too low, threads might be removed and restored too frequently. This can degrade system performance.
infranet.connectionpool.maxrequestlistsize	The maximum number of requests that can be held in the pending request queue. The default is **50**.

3. Save and close the file.
4. Run the **helm upgrade** command to update your Helm release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

Configuring Multiple Replicas of Batch Controller

If you load event files into your BRM cloud native deployment through Universal Event (UE) Loader, you can improve throughput by running multiple replicas of the batch-controller pod. In this case, each pod can select a file from those available in the UE Loader input PersistentVolumeClaim (PVC). When an individual pod copies the file into its local file system for processing, the other input files are distributed among the remaining batch-controller pod replicas. The time a file arrives in the input PVC determines which pod gets to process the file.

To configure the number of replicas:

1. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **ocbrm.batch_controller.deployment.replicaCount** key to the number of replicas to create of the batch-controller pod.

2. Run the **helm upgrade** command to update your Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
   n BrmNameSpace
   ```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

For more information about UE Loader, see "About Rating Events Created by External Sources" in *BRM Loading Events*.

Deploying Paymentech Data Manager in HA Mode

Paymentech supports only one connection to its batch port at any one time. To support high availability and increase throughput to the Paymentech server, you can deploy two Paymentech Data Manager (dm-fusa) images, with each image using a different batch port for connecting to the Paymentech server.

Deploying two images provides failover support for dm-fusa. If one dm-fusa deployment goes down, the traffic from CM to dm-fusa will be redirected to the other dm-fusa deployment. The load is also distributed among all dm-fusa deployments.

To deploy two dm-fusa images:

1. Edit the keys in the **configmap_pin_conf_dm_fusa.yaml** file for your system.
2. Edit these keys in the **configmap_env_dm_fusa.yaml** file:

   ```
   DMF_BATCH_PORT_2: "8781"
   DMF_BATCH_SRVR_2: fusa-simulator-2
   DMF_ONLINE_PORT_2: "9781"
   DMF_ONLINE_SRVR_2: fusa-simulator-2
   ```

   Note:

   Unlike the batch port, simultaneous transactions can be sent to the Paymentech online port. Thus, the values of **DMF_ONLINE_PORT_2** and **DMF_ONLINE_SRVR_2** can be the same as or different from that of the first dm-fusa deployment.

3. Rename the **_dm_fusa_2.yaml** file to **dm_fusa_2.yaml**.
4. Run the **helm upgrade** command to update the Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
   n BrmNameSpace
   ```

   where:

- *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
- *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
- *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

Using the FUSA Simulator

For testing purposes, a second deployment of the FUSA simulator is provided in the **templates** directory. To deploy this second version, rename the **_fusa_simulator_2.yaml** file to **fusa_simulator_2.yaml** and then update the Helm release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

The deployment scripts and configuration files for the FUSA simulator are provided for testing purposes only. In a production environment, remove these files:

- **fusa_simulator.yaml**
- **fusa_simulator_2.yaml**
- **configmap_pin_conf_fusa_simulator.yaml**
- **configmap_env_fusa_simulator.yaml**

8
Managing a BRM Cloud Native Multischema System

Learn how to perform basic tasks, such as migrating accounts, adding schemas, or setting a schema's status, in an Oracle Communications Billing and Revenue Management (BRM) cloud native multischema system.

Topics in this document:

Running Billing Against a Specified Schema

You generate bills for your customers' accounts by running the **pin_bill_accts** utility through the brm-apps job. By default, the utility is run against all schemas in your database, but you can configure BRM cloud native to run the utility against a specific schema. For more information about generating bills, see "Billing Accounts By Using the pin_bill_accts Utility" in *BRM Configuring and Running Billing*.

To run billing against a particular database schema using the brm-apps job:

1. In your **override-values.yaml** file for **oc-cn-helm-chart**, set these keys:
   - **ocbrm.brm_apps.job.isEnabled**: Set this to **true**.
   - **ocbrm.brm_apps.job.isMultiSchema**: Set this to **false**.
2. Update the **oc-cn-helm-chart/brmapps_scripts/loadme.sh** script to run **pin_bill_accts** commands on the specified schema:

   ```
   if [ "${DB_NUMBER}" = "0.0.0.x" ]; then
   cd /oms/apps/pin_billd; pin_bill_accts -verbose
   exit 0;
   ```

   where *x* is the schema number.
3. Run the **helm upgrade** command to update your Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
   n BrmNameSpace
   ```

where:

- *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
- *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
- *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

The **pin_bill_accts** utility generates bills for the accounts in the specified schema.

Adding Schemas to a Multischema System

To add one or more schemas to your existing BRM cloud native multischema system:

1. Initialize the new secondary schemas in your BRM database.
   a. Open your **override-values.yaml** file for **oc-cn-init-db-helm**.
   b. Set the **ocbrm.db.skipPrimary** key to **true**.
   c. For each existing secondary schema in your system, set the **ocbrm.db.multiSchemas.secondary***N***.deploy** key to **false**.
   d. For each new schema, add a **ocbrm.db.multiSchemas.secondary***N* block, where *N* is **3** for the third secondary schema, **4** for the next secondary schema, and so on.
   e. In the new **ocbrm.db.multiSchemas.secondary***N* block, set these keys:
      - **deploy**: Set this to **true** to deploy this secondary schema.
      - **host**: Set this to the hostname of the secondary schema. This key is optional.
      - **port**: Set this to the port number for the secondary schema. This key is optional.
      - **service**: Set this to the service name for the secondary schema. This key is optional.
      - **schemauser**: Set this to the schema user name.
      - **schemapass**: Set this to the schema password.
      - **schematablespace**: Set this to the name of the schema tablespace, such as pin01.
      - **indextablespace**: Set this to the name of the index tablespace, such as pinx01.

      This shows sample **override-values.yaml** entries for adding a third secondary schema to an existing multischema system.

```
ocbrm:
   isAmt: true
   db:
      skipPrimary: true
      multiSchemas:
         secondary1:
            deploy: false
            schemauser: pin02
            schemapass: password
            schematablespace: pin02
            indextablespace: pinx02
         secondary2:
            deploy: false
            schemauser: pin03
            schemapass: password
            schematablespace: pin03
            indextablespace: pinx03
         secondary3:
            deploy: true
            schemauser: pin04
            schemapass: password
            schematablespace: pin04
            indextablespace: pinx04
```

f. Save and close your **override-values.yaml** file.

g. Run the **helm install** command for **oc-cn-init-db-helm-chart**.

```
helm install InitDbReleaseName oc-cn-init-db-helm-chart --values
OverrideValuesFile -n InitDbNameSpace
```

where:

- *InitDbReleaseName* is the release name for **oc-cn-init-db-helm-chart** and is used to track this installation instance.
- *OverrideValuesFile* is the path to a YAML file that overrides the default configurations in the **values.yaml** file for **oc-cn-init-db-helm-cart**.
- *InitDbNameSpace* is the namespace for **oc-cn-init-db-helm-chart**.

2. Specify the details for connecting the BRM server to your new secondary schemas.

a. Open your **override-values.yaml** file for **oc-cn-helm-chart**.

b. Enable account migration by setting the **ocbrm.isAmt** key to **true**.

c. Set the **ocbrm.db.skipPrimary** key to **false**.

d. For each secondary schema you are adding to your system, add a **ocbrm.db.multiSchemas.secondary***N* block, where *N* is **3** for the third secondary schema, **4** for the next secondary schema, and so on.

e. In each **ocbrm.db.multiSchemas.secondary***N* block, set the following keys:

- **deploy**: Set this to **true**.
- **host**: Set this to the hostname of the secondary schema. This key is optional.
- **port**: Set this to the port number for the secondary schema. This key is optional.
- **service**: Set this to the service name for the secondary schema. This key is optional.
- **schemauser**: Set this to the schema user name.
- **schemapass**: Set this to the schema password.
- **schematablespace**: Set this to the name of the schema tablespace, such as pin01.

- **indextablespace**: Set this to the name of the index tablespace, such as pinx01.

This shows sample **override-values.yaml** entries for adding a third secondary schema to an existing multischema system.

```
ocbrm:
   isAmt: true
   db:
      skipPrimary: false
      multiSchemas:
         secondary1:
            deploy: true
            schemauser: pin02
            schemapass: password
            schematablespace: pin02
            indextablespace: pinx02
         secondary2:
            deploy: true
            schemauser: pin03
            schemapass: password
            schematablespace: pin03
            indextablespace: pinx03
         secondary3:
            deploy: true
            schemauser: pin04
            schemapass: password
            schematablespace: pin04
            indextablespace: pinx04
```

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

```
helm install BrmReleaseName oc-cn-helm-chart --namespace
BrmNameSpace --values OverrideValuesFile
```

where:

- *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance. It must be different from the one used for **oc-cn-init-db-helm-chart**.
- *BrmNameSpace* is the namespace in which to create BRM Kubernetes objects for the BRM Helm chart.
- *OverrideValuesFile* is the path to a YAML file that overrides the default configurations in the **values.yaml** file for **oc-cn-helm-chart**.

The BRM Helm chart deploys new dm-oracle, amt, and rel-dameon pods, Rated Event (RE) Loader PVCs, services, ConfigMaps, and secrets. It also updates their corresponding schema entries in the primary CM and Oracle DM, and deploys multiple containers for the batch-wireless-pipe pod.

3. Run the **pin_multidb.pl** and **pin_amt_install.pl** scripts through the brm-apps job.

   a. Add these lines to the **oc-cn-helm-chart/brmapps_scripts/loadme.sh** script:

```
#!/bin/sh
```

```
cd /oms/setup/scripts
perl pin_multidb.pl -i
perl pin_multidb.pl -f
perl pin_amt_install.pl
exit 0;
```

b. In your **override-values.yaml** file for **oc-cn-helm-chart**, set these keys:
   - **ocbrm.brm_apps.job.isEnabled**: Set this to **true**.
   - **ocbrm.brm_apps.job.isMultiSchema**: Set this to **false**.

c. Run the **helm upgrade** command to update the Helm release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

4. Set each database schema's status and priority. BRM cloud native assigns accounts to an open schema with the highest priority.

   a. Open the **configmap_pin_conf_testnap.yaml** file.

   b. Under the **config_dist.conf** section, add the following entries for each new secondary schema:

```
DB_NO = "schema_number" ;             # database config. block
PRIORITY = priority ;
MAX_ACCOUNT_SIZE = 100000 ;
STATUS = "status" ;
SCHEMA_NAME = "pin111x" ;
```

   c. Set the **STATUS** and **PRIORITY** entries for each new secondary schema:

```
DB_NO = "0.0.0.1" ;             # Primary schema configuration block
PRIORITY = priority;
MAX_ACCOUNT_SIZE = 100000 ;
STATUS = "status" ;
SCHEMA_NAME = "pin112x" ;

DB_NO = "0.0.0.2" ;             # Secondary schema configuration block
PRIORITY = priority;
MAX_ACCOUNT_SIZE = 50000 ;
STATUS = "status" ;
SCHEMA_NAME = "pin113x" ;
```

   where:

   - *priority* is a number representing the schema's priority, with the highest number having the most priority. For example, 5 indicates a greater priority than a value of 1. For more information, and "Modifying Database Schema Priorities".
   - *status* specifies whether the schema is **open**, **closed**, or **unavailable**. For more information, see "Modifying Database Schema Status".

d. Set up the configurator job to run the **load_config_dist** utility by adding the following lines to the **oc-cn-helm-chart/config_scripts/loadme.sh** script:

```
#!/bin/sh

cp /oms/config_dist.conf /oms/sys/test/config_dist.conf
cd /oms/sys/test ; load_config_dist
exit 0;
```

e. In the **override-values.yaml** file for **oc-cn-helm-chart**, set this key:

   ocbrm.config_jobs.run_apps: Set this to **true**.

f. Run the **helm upgrade** command to update the Helm release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

   The distribution information is loaded into the primary schema.

g. Update these keys in the **override-values.yaml** file for **oc-cn-helm-chart**:
   - **ocbrm.config_jobs.restart_count**: Increment the existing value by 1.
   - **ocbrm.config_jobs.run_apps**: Set this to **false**.

h. Update the **oc-cn-helm-chart** release again:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

   The CM is restarted.

5. Reset POID_IDS2 sequences as part of the brm-apps job.

   a. Add these lines to the **oc-cn-helm-chart/brmapps_scripts/loadme.sh** script:

```
#!/bin/sh

java -cp $ORACLE_HOME/lib/ojdbc8.jar:$PIN_HOME/jars/
pin_reset_seq.jar:$PIN_HOME/jars/pcm.jar:$PIN_HOME/jars/
oraclepki.jar PinResetSeq /oms/pin_confs2/
pin_reset_seq.properties
exit 0;
```

   b. In your **override-values.yaml** file for **oc-cn-helm-chart**, set these keys:
      - **ocbrm.brm_apps.job.isEnabled**: Set this to **true**.
      - **ocbrm.brm_apps.job.isMultiSchema**: Set this to **false**.

   c. Update the **oc-cn-helm-chart** release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

6. Set up the configuration job to run the **load_pin_uniqueness** utility.

See "Synchronizing the Database Schema /uniqueness Objects" in *BRM System Administrator's Guide* for more information about the utility.

a. Add the following lines to the **oc-cn-helm-chart/config_scripts/loadme.sh** script:

```
#!/bin/sh

cd /oms/sys/test ; load_pin_uniqueness
exit 0;
```

b. In the **override-values.yaml** file for **oc-cn-helm-chart**, set this key:

   ocbrm.config_jobs.run_apps: Set this to **true**.

c. Update the **oc-cn-helm-chart** release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

   The **/uniqueness** objects are synchronized between the schemas.

d. Update these keys in the **override-values.yaml** file for **oc-cn-helm-chart**:

   - **ocbrm.config_jobs.restart_count**: Increment the existing value by 1.
   - **ocbrm.config_jobs.run_apps**: Set this to **false**.

e. Update the **oc-cn-helm-chart** release again:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

   The CM is restarted.

7. Configure the account-router Pipeline Manager to route CDRs to pipelines based on the database schema POID. To do so, edit the ConfigMap file **configmap_acc_router_reg.yaml**.

   Based on the configuration, the account router Pipeline Manager does the following:

   - Moves input files to the **data** PVC directory. The input file names have a prefix of **router** and a suffix of **.edr**.
   - Moves the rated output files to the input of the Rating pipeline.
   - Replicates the Rating pipeline based on the multischema entry. The Range function is used to replicate the rating pipeline.
   - Moves the output files from the Rating pipeline to the **outputcdr** PVC directory.

Migrating Accounts from One Schema to Another

You migrate accounts from one schema to another schema in the same database by configuring the account search configuration file and then running the **pin_amt** utility through the brm-apps job. For more information, see "Understanding Account Migration" in *BRM Moving Accounts between Database Schemas*.

To migrate accounts from one schema to another:

1. Ensure that Account Migration Manager (**ocbrm.isAmt**) is enabled in your BRM database.
2. In your **override-values.yaml** file for **oc-cn-helm-chart**, set these keys:
   - **ocbrm.brm_apps.job.isEnabled**: Set this to **true**.
   - **ocbrm.brm_apps.job.isMultiSchema**: Set this to **false**.
   - **ocbrm.isAmt**: Set this to **true**.
3. Update the **oc-cn-helm-chart/brmapps_scripts/loadme.sh** script to run **pin_amt** commands:

   ```
   cd /oms/apps/amt; pin_amt -s /oms/apps/amt/account_search.cfg
   exit 0;
   ```

4. In the **configmap_infranet_properties_brm_apps.yaml** file, do this:
   a. Under the **Infranet.properties** section, set the **controller_1_hold_period** key to the amount of time, in minutes, that the AMM Controller waits before migrating accounts. This provides time for your pipelines to flush any EDRs targeted for accounts in the migration job. The default is **120**.

      ```
      controller_1_hold_period=Value
      ```

   b. Under the **account_search.cfg** section, specify the account search criteria by editing the parameters in Table 8-1.

Table 8-1 Account Search Parameters

Parameter	Description	Required
src_database	Specifies the source schema, which is the schema from which you are migrating accounts. The default is **0.0.0.1**.	YES
dest_database	Specifies the destination schema, which is the schema to which you are migrating accounts. The default is **0.0.0.2**.	YES
batch_size	Specifies the number of accounts in each batch. You can specify any amount from 1 through 1,000. However, for optimal performance, set this to an integer between 50 and 100. The default is **100**. **Important:** • Using a batch size of more than 50 accounts does not improve performance. • If you set this to a number greater than 100, you must increase the size of your Oracle rollback segments.	YES
start_creation_date	Use this parameter to migrate accounts that were created in a specific date range. AMM migrates accounts created between midnight (00:00:00) on the start date and 23:59:59 on the end date. For example, to migrate accounts created after midnight on August 1, 2030, enter **08/01/2030**. **Important:** If you set this parameter, you must also set the **end_creation_date** parameter.	no

Table 8-1 (Cont.) Account Search Parameters

Parameter	Description	Required
end_creation_date	Use this parameter to migrate accounts that were created in a specific date range. AMM migrates accounts created between midnight (00:00:00) on the start date and 23:59:59 on the end date. For example, to migrate accounts created on or before 11:59:59 p.m. on August 10, 2030, enter **08/10/2030**. **Important:** If you set this parameter, you must also set the **start_creation_date** parameter.	no
product_name	Migrates accounts that purchased the specified charge offer. For example, **Offer 1b - Email Account**.	no
account_status	Migrates accounts based on the specified account status: • **Active**: Migrates only active accounts. This is the default. • **Inactive**: Migrates only inactive accounts. • **Closed**: Migrates only closed accounts.	no
bill_day_of_month	Migrates accounts that have the specified billing day of month (DOM). You can specify any number from 1 through 31. For example, enter **4** to migrate all accounts that are billed on the 4th of the month.	no
max_accounts	Specifies the maximum number of accounts to move in a job. The default is **200**.	no
poid_list	Migrates accounts based on the POID. Use comma separators, for example, **22860, 22861, 22862**. Limit the number of accounts to 1,000 or less.	no
migration_mode	Specifies whether to migrate account groups. When AMM finds an account that belongs to a hierarchical account, charge sharing group, or discount sharing group, AMM migrates all accounts related to that account. • **IncludeAccountGroup** specifies to migrate accounts groups. • **ExcludeAccountGroup** specifies to exclude account groups from migrations. This is the default. **Important:** If you set this parameter, you must also set the **max_group_size** parameter.	no
max_group_size	Specifies the maximum size of an account group that AMM can migrate. If an account group exceeds the maximum number of accounts, AMM excludes the account group from the job. The default is **100**.	no
cross_schema_group	Specifies whether **pin_amt** migrates accounts that belong to a cross-schema sharing group. A cross-schema sharing group has members in multiple database schemas. • **Enabled**: Does not migrate account members of a cross-schema sharing group. • **Disabled**: Migrates account members of a cross-schema sharing group. This is the default. **Note:** When this parameter is enabled, AMM performs validation for an account and only its immediate child account. You should perform extra validation to ensure accounts picked up by AMM are not part of a cross-schema sharing group.	no

For more information, see "Creating the Account Search Configuration File" in *BRM Moving Accounts between Database Schemas*.

5. Run the **helm upgrade** command to update the release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values
   OverrideValuesFile -n BrmNameSpace
   ```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

   The accounts meeting your search criteria are migrated from the source schema to the destination schema.
6. Verify the brm-apps and controller log files.

Migrating Accounts Using Custom Search Criteria

Account Migration Manager (AMM) allows you to migrate accounts from one schema to another using custom search criteria. For example, you can create custom criteria for finding and migrating accounts for customers living in a certain American state or belonging to a particular service provider.

To migrate accounts using custom search criteria:

1. Ensure that Account Migration Manager (**ocbrm.isAmt**) is enabled in your BRM database.
2. In your **override-values.yaml** file for **oc-cn-helm-chart**, set these keys:
   - **ocbrm.brm_apps.job.isEnabled**: Set this to **true**.
   - **ocbrm.brm_apps.job.isMultiSchema**: Set this to **false**.
   - **ocbrm.isAmt**: Set this to **true**.
3. Update the **oc-cn-helm-chart/brmapps_scripts/loadme.sh** script to run **pin_amt** commands:

   ```
   cd /oms/apps/amt; pin_amt -s /oms/apps/amt/account_search.cfg
   exit 0;
   ```

4. Open the **configmap_infranet_properties_brm_apps.yaml** file.
5. Under the **Infranet.properties** section, set the **controller_1_hold_period** key to the amount of time, in minutes, that the AMM Controller waits before migrating accounts. This provides time for your pipelines to flush any EDRs targeted for accounts in the migration job. The default is **120**.

   ```
   controller_1_hold_period=Value
   ```

6. Under the **custom_account_search.properties** section, add SQL fragments for your search criteria using this syntax:

   ```
   criteria_name=AND SQL_condition \n
   ```

   where:

   - *criteria_name* is the name of your selection criteria.
   - *SQL_condition* is a valid SQL condition that searches a BRM table and references one or more search variables as shown below. Search variables must be surrounded by curly braces "**{ }**" and match an entry under the **account_search.cfg** section.

     ```
     condition_text '{SearchVariable}'...
     ```

   - *SearchVariable* must use a unique name and must not match one of the BRM-defined search variable names under the **account_search.cfg** section.

   For example, this SQL fragment enables AMM to search for accounts located in a particular state. AMM searches the ACCOUNT_NAME_INFO_T table for objects with the **state** field set to a specified value.

   ```
   # select accounts based on state
   cust_acct_search_account_state_constraint=\
   AND EXISTS \n\
   (SELECT an.obj_id0 FROM account_nameinfo_t an \n\
   WHERE an.obj_id0 = a.poid_id0 and an.state = '{account_state}') \n
   ```

7. Under the **account_search.cfg** section, add your *SearchVariable* entry set to the appropriate value.

   For example:

   ```
   # - Migrates accounts located in a specific state. Valid values
   # are California and Oregon.
   account_state=California
   ```

8. Under the **account_search.cfg** section, specify the source and destination schema as well as any additional account search criteria by editing the parameters in Table 8-1.
9. Save and close the **configmap_infranet_properties_brm_apps.yaml** file.
10. For each custom search variable, create a corresponding Java implementation of the **Conversion** interface.
    a. Run the appropriate profile script for your shell. This script sets your CLASSPATH and PATH environment variables to the appropriate values.

       For example, for the c shell:

       ```
       cd BRM_home/apps/amt
       source profile.csh
       ```

    b. Create a class that implements the **Conversion** interface.

The following sample class, **account_state.class**, allows users to search for accounts from California or Oregon.

```
package com.portal.amt;
public class account_state implements Conversion {
  public String convert(String stateName) throws
ConversionException {
    String stateCode = null;
    if(stateName.equals("California")) {
      stateCode = "CA";
    } else if(stateName.equals("Oregon")) {
      stateCode = "OR";
    } else {
      throw new
        ConversionException("Error: account_state " + stateName
+ " unknown.");
    }
    return(stateCode);
  }
}
```

c. Save and compile your *SearchVariable***.java** source file in the *BRM_home***/ apps/amt/com/portal/amt** directory.

```
cd BRM_home/apps/amt/com/portal/amt
javac SearchVariable.java
```

This creates a *SearchVariable***.class** file in the same directory.

11. Run the **helm upgrade** command to update the release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

The accounts meeting your custom search criteria are migrated from the source schema to the destination schema.

12. Verify the brm-apps and controller log files.

Modifying Database Schema Priorities

Database schema priority determines when customer accounts are created in a particular schema relative to other schemas. Multidatabase Manager assigns accounts to an open schema with the highest priority.

If all schemas have the same priority, Multidatabase Manager chooses an open schema at random in which to create the account. This distributes accounts evenly across all schemas. However, BRM locates accounts as follows:

- All accounts with nonpaying child units in the same schema as their paying parent bill units
- All sponsored accounts in the same schema as their sponsoring accounts

To limit the number of accounts in your primary database schema, set your primary database schema to a *lower* priority than the secondary database schemas. Accounts will be created in the secondary database schemas when possible.

You set each schema's priority by editing the **configmap_pin_conf_testnap.yaml** file and then running the **load_config_dist** utility through the configurator job.

Note:

The **load_config_dist** utility overwrites all distributions already in the database. When adding or updating distributions, be aware that you cannot load only new and changed distributions.

To modify database schema priorities:

1. Open the **configmap_pin_conf_testnap.yaml** file.
2. Under **config_dist.conf**, set the **PRIORITY** entries to the schema's priority with the highest number having the most priority. For example, 5 indicates a greater priority than a value of 1.

   In this example, BRM cloud native would create accounts on schema 0.0.0.2 because it has the highest priority setting of all open schemas.

   ```
   DB_NO = "0.0.0.1" ;               # 1st database config. block
   PRIORITY = 1 ;
   MAX_ACCOUNT_SIZE = 100000 ;
   STATUS = "OPEN" ;
   SCHEMA_NAME = "schema_name"

   DB_NO = "0.0.0.2" ;               # 2nd database config. block
   PRIORITY = 3;
   MAX_ACCOUNT_SIZE = 50000 ;
   STATUS = "OPEN" ;
   SCHEMA_NAME = "schema_name"

   DB_NO = "0.0.0.3" ;               # 3rd database config. block
   PRIORITY = 5;
   MAX_ACCOUNT_SIZE = 50000 ;
   STATUS = "CLOSED" ;
   SCHEMA_NAME = "schema_name"
   ```

3. Save and close the file.
4. Set up the configurator job to run the **load_config_dist** utility by adding the following lines to the **oc-cn-helm-chart/config_scripts/loadme.sh** script:

   ```
   #!/bin/sh

   #cp /oms/config_dist.conf /oms/sys/test/config_dist.conf
   cd /oms/sys/test ; load_config_dist
   exit 0;
   ```

5. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **ocbrm.config_jobs.run_apps** key to **true**.

6. Run the **helm upgrade** command to update the Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values
   OverrideValuesFile -n BrmNameSpace
   ```

   The distribution information is loaded into the primary schema.

7. Restart the CM.

   a. Update these keys in the **override-values.yaml** file for **oc-cn-helm-chart**:

      - **ocbrm.config_jobs.restart_count**: Increment the existing value by 1
      - **ocbrm.config_jobs.run_apps**: Set this to **false**

   b. Update the Helm release again:

      ```
      helm upgrade BrmReleaseName oc-cn-helm-chart --values
      OverrideValuesFile -n BrmNameSpace
      ```

      The CM is restarted.

Modifying Database Schema Status

Database schema status determines whether a schema is available for account creation. Schemas can be set to the following statuses:

- **Open:** Open schemas are available for account creation.
- **Closed:** Closed schemas are not used for account creation under most circumstances. Accounts are created in a closed schema only if a sponsoring account belongs to that schema or if all schemas are closed. If all schemas are closed, Multidatabase Manager chooses a closed schema at random in which to create accounts and continues to create accounts in that schema until a schema becomes open. To limit the number of accounts created in a schema, you can manually change the schema's status to closed, or you can have Multidatabase Manager automatically change it to closed when the schema reaches a predefined limit.
- **Unavailable:** Unavailable schemas are not used for account creation unless the schema contains an account's parent or sponsoring account.

You set each schema's status by editing the **configmap_pin_conf_testnap.yaml** file and then running the **load_config_dist** utility through the configurator job.

Note:

The **load_config_dist** utility overwrites all distributions already in the database. When adding or updating distributions, be aware that you cannot load only new and changed distributions.

To modify a schema's status:

1. Open the **configmap_pin_conf_testnap.yaml** file.

2. Under **config_dist.conf**, set the value of each schema's **STATUS** entry to **OPEN**, **CLOSED**, or **UNAVAILABLE**. For example:

```
DB_NO = "0.0.0.1" ;               # 1st database config. block
PRIORITY = 1 ;
MAX_ACCOUNT_SIZE = 100000 ;
STATUS = "OPEN" ;
SCHEMA_NAME = "schema_name" ;

DB_NO = "0.0.0.2" ;               # 2nd database config. block
PRIORITY = 3;
MAX_ACCOUNT_SIZE = 50000 ;
STATUS = "OPEN" ;
SCHEMA_NAME = "schema_name" ;
```

3. Save and close the file.
4. Set up the configurator job to run the **load_config_dist** utility by adding the following lines to the **oc-cn-helm-chart/config_scripts/loadme.sh** script:

```
#!/bin/sh

#cp /oms/config_dist.conf /oms/sys/test/config_dist.conf
cd /oms/sys/test ; load_config_dist
exit 0;
```

5. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **ocbrm.config_jobs.run_apps** key to **true**.
6. Run the **helm upgrade** command to update the Helm release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
n BrmNameSpace
```

   The distribution information is loaded into the primary schema.
7. Restart the CM.
   a. Update these keys in the **override-values.yaml** file for **oc-cn-helm-chart**:
      - **ocbrm.config_jobs.restart_count**: Increment the existing value by 1
      - **ocbrm.config_jobs.run_apps**: Set this to **false**
   b. Update the Helm release again:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

Synchronizing /uniqueness Objects Between Schemas

In a multischema environment, BRM cloud native uses the **/uniqueness** object to locate subscribers. It contains a cache of services and must stay synchronized with the service cache in the primary schema. During normal multischema operations, the **/uniqueness** objects in the primary and secondary database schemas are updated automatically.

To determine whether the **/uniqueness** object in a secondary database schema is out of synchronization, use **sqlplus** to compare the entries in the **uniqueness_t** database table with those in the **service_t** database table. There should be a one-to-one relationship.

If the database tables are not synchronized, run the **load_pin_uniqueness** utility through the configurator job. This utility updates the **/uniqueness** object with the current service data.

To synchronize **/uniqueness** objects between database schemas:

1. Set up the configurator job to run the **load_pin_uniqueness** utility by adding the following lines to the **oc-cn-helm-chart/config_scripts/loadme.sh** script:

   ```
   #!/bin/sh

   cd /oms/sys/test ; load_pin_uniqueness
   exit 0;
   ```

2. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **ocbrm.config_jobs.run_apps** key to **true**.
3. Run the **helm upgrade** command to update the Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values
   OverrideValuesFile -n BrmNameSpace
   ```

   The **load_pin_uniqueness** utility is run.
4. Restart the CM.
   a. Update these keys in the **override-values.yaml** file for **oc-cn-helm-chart**:
      - **ocbrm.config_jobs.restart_count**: Increment the existing value by 1
      - **ocbrm.config_jobs.run_apps**: Set this to **false**

   b. Update the Helm release again:

      ```
      helm upgrade BrmReleaseName oc-cn-helm-chart --values
      OverrideValuesFile -n BrmNameSpace
      ```

      The CM is restarted.
5. Verify that the **/uniqueness** object was loaded by using one of the following to display the **/uniqueness** object:
   - Object Browser.
   - **robj** command with the **testnap** utility.

9

Migrating Legacy Data to BRM Cloud Native

Learn how to migrate data from your legacy database to the Oracle Communications Billing and Revenue Management (BRM) cloud native database.

Topics in this document:

About Migrating Legacy Data

You migrate legacy data to the BRM cloud native database using Conversion Manager. Conversion Manager can migrate the following types of data: account data, service data, product offering data, billing data, account hierarchy data, and balance data. See "Understanding Conversion Manager" in *BRM Migrating Accounts to the BRM Database* for more information.

The high-level steps for migrating legacy data to the BRM cloud native database include the following:

1. Understanding the data in your legacy system and deciding how to convert it to the database.
2. Mapping the data in your legacy database to the BRM database. To do so, you create XML files that are validated by the Conversion Manager XSD schema files.

   See "Mapping Legacy Data to the BRM Data Schema" in *BRM Migrating Accounts to the BRM Database*.
3. Migrating the data to the BRM database by running the **pin_cmt** utility through a brm-apps job.

   See "Loading Legacy Data into the BRM Database".

Loading Legacy Data into the BRM Database

You load legacy data into the BRM cloud native database in a multistep process:

- Import your legacy data into a staged area of the BRM database
- If necessary, recover and reload any failed load processes
- Deploy the data from the staged area to the production area of the BRM database

You load legacy data by running the **pin_cmt** utility through the brm-apps job. For more information about the utility's parameters and syntax, see "pin_cmt" in *BRM Migrating Accounts to the BRM Database*.

To load legacy data into the BRM database, do the following:

1. Ensure that BRM cloud native is running.

2. (Optional) Modify the **pin_cmt** utility's connection and performance parameters. To do so, edit the infranet-properties-brm-apps ConfigMap (**configmap_infranet_properties_brm_apps.yaml**):

   a. Under the file's **cmt_Infranet.properties** section, edit the **pin_cmt** parameters.

   b. Run the **helm upgrade** command to update the BRM Helm release:

      ```
      helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile
      -n BrmNameSpace
      ```

      where:

      - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
      - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
      - *BrmNameSpace* is the namespace in which to create BRM Kubernetes objects for the BRM Helm chart.

3. Import your legacy data into a staged area of the BRM database:

   a. Add the following lines to the **oc-cn-helm-chart/brmapps_scripts/loadme.sh** script:

      ```
      #!/bin/sh

      cd /oms/apps/pin_cmt; pin_cmt -import -file XML_input_data_file  stage_ID
      cd /oms/apps/pin_cmt; pin_cmt -import_custom -file XML_custom_data_file
      stage_ID
      exit 0;
      ```

      where:

      - *XML_input_data_file* is the file name and path to the XML file containing the mapping between the legacy and BRM databases.
      - *stage_ID* is the identity of the staging area.
      - *XML_custom_data_file* is the file name and path to the XML file containing the mapping between your legacy database and new storable classes in the BRM database.

   b. Move the *XML_input_data_file* and *XML_custom_data_file* files to the **oc-cn-helm-chart/brmapps_scripts** directory.

   c. Enable the **pin_cmt** utility and brm-apps job. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the following keys:

      - **ocbrm.cmt.enabled**: Set this to **true**.
      - **ocbrm.brm_apps.job.isEnabled**: Set this to **true**.

   d. Run the **helm upgrade** command to update the BRM Helm release:

      ```
      helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile
      -n BrmNameSpace
      ```

4. Check for load processes that failed and, if any did, recover and reload the processes:

   a. Check the **cmt.pinlog** file for load failures.

   b. In the **cmt.pinlog** file, retrieve the batch ID for each failed load process.
   c. Add the following lines to the **oc-cn-helm-chart/brmapps_scripts/loadme.sh** script:

      ```
      #!/bin/sh

      cd /oms/apps/pin_cmt; pin_cmt -recovery load batch_ID
      exit 0;
      ```

      where *batch_ID* is the batch ID you retrieved from **cmt.pinlog**.
   d. Run the **helm upgrade** command to update the BRM Helm release:

      ```
      helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -n
      BrmNameSpace
      ```

5. Deploy your data from the staged area to the production area of the BRM database:
   a. Add the following lines to the **oc-cn-helm-chart/brmapps_scripts/loadme.sh** script:

      ```
      #!/bin/sh

      cd /oms/apps/pin_cmt; pin_cmt -deploy DOM stage_ID
      exit 0;
      ```

      where *DOM* is the billing cycle's day of the month. Only those accounts with the specified stage ID and DOM are deployed.
   b. Run the **helm upgrade** command to update the BRM Helm release:

      ```
      helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -n
      BrmNameSpace
      ```

After accounts are deployed, BRM cloud native starts their billing cycles, applies any cycle fees, and, in multischema systems, updates the uniqueness table in the primary database schema.

Part II
Monitoring BRM Cloud Native Services

This part describes how to monitor Oracle Communications Billing and Revenue Management (BRM) cloud native services. It contains the following chapters:

For information about monitoring Elastic Charging Engine (ECE) and Pricing Design Center (PDC), see "Monitoring ECE in a Cloud Native Environment" and "Monitoring PDC in a Cloud Native Environment".

10 Monitoring BRM Cloud Native Services

Learn how to monitor your Oracle Communications Billing and Revenue Management (BRM) cloud native services by using Prometheus and Grafana.

Topics in this document:

- About Monitoring BRM Cloud Native Services
- Setting Up Monitoring for BRM Cloud Native Services
- BRM Opcode Metric Group

About Monitoring BRM Cloud Native Services

You can set up monitoring for the following BRM cloud native services:

- CM
- Oracle DM
- Oracle DM shared memory, front-end processes, and back-end processes
- Account Synchronization DM
- Synchronization Queue DM
- BRM Java Applications: RE Loader Daemon, Batch Controller, and EAI Java Server (JS)
- Web Services Manager
- BRM database

The metrics for the database are generated by OracleDB_exporter, and the metrics for all other BRM services are generated directly by BRM cloud native. You use Prometheus to scrape and store the metric data, and then use Grafana to display the data in a graphical dashboard.

Setting Up Monitoring for BRM Cloud Native Services

To set up monitoring for BRM cloud native services:

1. Deploy Prometheus in your Kubernetes Cluster in one of the following ways:
   - Deploy a standalone version of Prometheus in your cloud native environment. See "Installation" in the Prometheus documentation.
   - Deploy Prometheus Operator. See "prometheus-operator" on the GitHub website.

   For the list of compatible software versions, see "BRM Cloud Native Deployment Software Compatibility" in *BRM Compatibility Matrix*.
2. Install Grafana. See "Install Grafana" in the Grafana documentation.

   For the list of compatible software versions, see "BRM Cloud Native Deployment Software Compatibility" in *BRM Compatibility Matrix*.

3. Configure BRM cloud native to collect metrics for its components and export them to Prometheus. See "Configuring BRM Cloud Native to Collect Metrics".
4. Configure how Perflib generates metric data for BRM opcodes. See "Configuring Perflib for BRM Opcode Monitoring".
5. Configure OracleDB_exporter to scrape metrics from your Oracle database and export them to Prometheus. See "Configuring OracleDB_Exporter to Scrape Database Metrics".
6. Create Grafana Dashboards for viewing your metric data. See "Configuring Grafana for BRM Cloud Native".

Configuring BRM Cloud Native to Collect Metrics

To configure BRM cloud native to collect metrics for its components and then expose them in Prometheus format:

1. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **monitoring.prometheus.operator.enable** key to one of the following:
   - **true** if you are using Prometheus Operator.
   - **false** if you are using a standalone version of Prometheus. This is the default.
2. To collect metrics for the CM, do the following:
   a. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **ocbrm.cm.deployment.perflib_enabled** key to **true**.
   b. In the **oms-cm-perflib-config** ConfigMap, review and update the Perflib configuration. For information about the possible values, see "Configuring Perflib for BRM Opcode Monitoring".
   c. In the **oms-cm-config** ConfigMap, review and update the Perflib configuration. For information about the possible values, see "Configuring Perflib for BRM Opcode Monitoring".
3. To collect metrics for Oracle DM shared memory, front-end processes, and back-end processes, do the following:

   In the **oms-cm-perflib-config** ConfigMap, set the **data.ENABLE_STATUS_DM_METRICS** key to **true**.
4. To collect metrics for the dm-oracle pod, do the following:
   - In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **ocbrm.dm_oracle.deployment.perflib_enabled** key to **true**.
   - In the **oms-dm-oracle-perflib-config** ConfigMap, review and update the Perflib configuration. For information about the possible values, see "Configuring Perflib for BRM Opcode Monitoring".
   - In the **oms-dm-oracle-config** ConfigMap, review and update the Perflib configuration. For information about the possible values, see "Configuring Perflib for BRM Opcode Monitoring".
5. To collect metrics for dm_ifw_sync and dm_aq, do the following:

   In your **override-values.yaml** file for **oc-cn-helm-chart**, set these keys:
   - **ocbrm.dm_ifw_sync.deployment.perflib_enabled**: Set this key to **true**.
   - **ocbrm.dm_aq.deployment.perflib_enabled**: Set this key to **true**.

6. To collect metrics for the BRM Java applications, REL Daemon, Batch Controller, and EAI Java Server, do the following:

   In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **monitoring.prometheus.jmx_exporter.enable** key to **true**.

7. To collect metrics for Web Services Manager, do the following:

   In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **ocbrm.wsm.deployment.monitoring.isEnabled** key to **true**.

8. To persist the Perflib timing files in your BRM database, do the following:

   a. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **ocbrm.perflib.deployment.persistPerlibLogs** key to **true**.

   b. Check the values of these Perflib timing-related environment variables in your **oms-cm-perflib-config** and **oms-dm-oracle-perflib-config** ConfigMaps: PERFLIB_VAR_TIME, PERFLIB_VAR_FLIST, and PERFLIB_VAR_ALARM. See Table 10-1 for more information.

9. Run the **helm upgrade** command to update the BRM Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
   n BrmNameSpace
   ```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

After you update the Helm release, metrics will be exposed to Prometheus through the CM pod at the **/metrics** endpoint with the following ports:

- CM: Port 11961
- Oracle DM shared memory, back-end processes, and front-end processes: Port 11961 or Port 31961
- Oracle DM: Port 12951
- dm_ifw_sync and dm_aq: Port 12951

Example: Enabling Monitoring for All BRM Components

This shows sample **override-values.yaml** entries for enabling the collection of the following metrics for Prometheus:

- CM
- Oracle DM
- Oracle DM shared memory, front-end processes, and back-end processes
- Account Synchronization DM (dm_ifw_sync)
- Synchronization Queue DM (dm_aq)
- Web Services Manager

- BRM Java applications: REL Daemon, Batch Controller, and EAI Java Server

It also configures BRM to persist the Perflib timing files in your BRM database.

```
monitoring:
    prometheus:
    operator:
        enable: false
    jmx_exporter:
        enable: true
ocbrm:
    cm:
        deployment:
            perflib_enabled: true
    dm_oracle:
        deployment:
            perflib_enabled: true
    perflib:
        deployment:
            persistPerflibLogs: true
    wsm:
        deployment:
            monitoring:
                isEnabled: true
    dm_ifw_sync:
        deployment:
            perflib_enabled: true
    dm_aq:
        deployment:
            perflib_enabled: true
```

Configuring Perflib for BRM Opcode Monitoring

The BRM cloud native deployment package includes the BRM Performance Profiling Toolkit (Perflib), which the Connection Manager (CM), Oracle Data Manager (DM), Synchronization Queue DM, and Account Synchronization DM depend on for generating and exposing BRM opcode metrics.

You configure how Perflib generates the metric data by setting environment variables in the following:

- For the CM: The **oms-cm-perflib-config** ConfigMap
- For the DMs: The **oms-dm-oracle-perflib-config** ConfigMap

Table 10-1 describes the environment variables you can use to configure Perflib for the CM and DMs.

Table 10-1 Perflib Environment Variables

Environment Variable	Description
PERFLIB_ENABLED	Whether to enable opcode monitoring with Perflib. • **0**: Disables Perflib. • **1**: Enables Perflib. This is the default.

Table 10-1 (Cont.) Perflib Environment Variables

Environment Variable	Description
PERFLIB_HOME	The location of the Perflib Toolkit.
PERFLIB_DEBUG	The debug log level for Perflib. • **0**: Turn off debugging. This is the default. • **1**: Log summary information to **stderr**. • **2**: Log detailed opcode execution information to **stderr**. • **4**: Log trace information to **stderr**.
PERFLIB_MAX_LOG_SIZE	The maximum number of opcodes that can be logged in any one log file. You can use this to prevent excessively large log files if detailed tracing is used for long periods. • **0**: A single file is created with no limits. This is the default. • *Number*: Defines the maximum number of opcodes that can be tracked before a new file is opened.
PERFLIB_AGGREGATION_PERIOD	The amount of time that data is recorded into a bucket, in minutes or hours. When the amount of time expires, Perflib creates a new bucket. For example, each bucket could record an hour's worth of data, 2 hours of data, or 5 minutes of data. The allowed values for hours: **1h**, **2h**, **3h**, **4h**, **6h**, **8h**, **12h**, or **24h**. The allowed values for minutes: **1m**, **2m**, **3m**, **4m**, **5m**, **6m**, **10m**, **12m**, **15m**, **30m**, or **60m**. The default is **1h**.
PERFLIB_FLUSH_FREQUENCY	How frequently, in seconds, to flush in-memory aggregation data to trace files on disk. The default is **3600** (1 hour).
PERFLIB_LOG_SINGLE_FILE	The prefix to use for tracing filenames, such as **cm_batch**, **cm_aia**, or **cm_rt**. This allows you to distinguish the trace files for each type of application. The default is **perf_log**.
PERFLIB_PIN_SHLIB	The full path of the shared library that contains the BRM opcode functions being interposed. This environment variable is used for the CM only. The default is **/oms/lib/libcmpin.so**.
PERFLIB_DATA_FILE	The full path name of the memory-mapped data file that is used by Perflib to store control variables and real-time trace data. The following special formatting characters can be used as part of the data file name and is substituted by Perflib when the data file is created: • **%p**: Substituted with the process ID of the process using Perflib. • **%t**: Substituted with the current time stamp (as a UNIX time number). • **%T**: Substituted with the current timestamp (as a *YYYYMMDDHHMMSS* string). The default is **/oms_logs/perflib_data.dat**.

Table 10-1 (Cont.) Perflib Environment Variables

Environment Variable	Description
PERFLIB_LOG_DIR	The directory where trace output is written. The default is **/oms_logs**.
PERFLIB_DATA_FILE_RESET	Whether real-time tracing data and variable settings are maintained between application executions. This enables statistics to continue to accumulate across an application restart. • **Y**: All variables and trace data are destroyed when the application starts. This is the default. • **N**: The existing data is retained.
PERFLIB_VAR_TIME	Whether the Perflib tracing is activated immediately. • **0**: Timing is disabled at startup. • **1**: Real-time timing is enabled at startup. This is the default. • **2**: File-based timing is enabled at startup. • **3**: File-based and real-time timing is enabled at startup.
PERFLIB_VAR_FLIST	Whether the Perflib flist tracing is activated immediately. • **0**: Flist logging is disabled. This is the default. • **1**: Summary logging is enabled at startup. • **2**: Full flist logging is enabled at startup.
PERFLIB_VAR_ALARM	Whether the Perflib alarm functionality is activated immediately. • **0**: Alarms are disabled at startup. • **1**: Alarms are enabled at startup. This is the default.
PERFLIB_AUTO_FLUSH	Whether the CM flushes data regularly (with the frequency set by PERFLIB_FLUSH_FREQUENCY). • **0**: Disables flushing. In this case, if a CM does not receive any opcode requests, flushing is not performed until the CM terminates or an opcode arrives. This is the default. • **1**: Enables intra-opcode flushing. That is, flushing occurs between different top-level opcodes. • **2**: Enables full flushing. That is, flushing occurs within an opcode without waiting for it to complete. This can be useful when tracing very long-running opcodes. This environment variable is used for the CM only.
PERFLIB_COLLECT_CPU_USAGE	Whether user and system CPU usage is tracked at the opcode level, allowing CPU hogs to be identified more easily. • **0**: Collection is disabled. • *Positive value*: CPU data is collected for opcodes down to that level. For example, setting it to **1** would collect CPU data for the top-level opcodes, while setting it to **2** would collect data for both the top-level opcodes and all the children.
PERFLIB_LOCK_METHOD	The method used to lock between processes. • **0**: Use POSIX shared-memory mutexes. This is the default. • **1**: Use file-based advisory locks.

Table 10-1 (Cont.) Perflib Environment Variables

Environment Variable	Description
PERFLIB_ASYNC_FLUSHING	Whether flushing to the trace file from memory is done within the opcode execution, or asynchronously in a separate thread. • **0**: Flush data to the trace file within the opcode execution. • **1**: Flush data to the trace file in a separate processing thread. This is the default.
PERFLIB_TRACE_OBJECT_TYPE	Whether Perflib records the BRM object type associated with different database operations, such as PCM_OP_SEARCH, PCM_OP_READ_FLDS, PCM_OP_WRITE_FLDS, and so on. This can help you understand which objects are being read or written most frequently and how much time is being spent for different objects. For PCM_OP_EXEC_SPROC, the latest versions of Perflib will record the stored procedure name that was run. • **0**: Do not collect object type data. • **1**: Collect object type data and record it in realtime or batch trace files. This is the default.
PERFLIB_GROUP_TRANSACTIONS	Whether Perflib tracks BRM transactions as a single unit. The opcodes run as part of a transaction are grouped under a virtual opcode, TRANSACTION_GROUP. • **0**: Do not group transactions. This is the default. • **1**: Group transactions.
PERFLIB_LOG_MAX_SINGLE_FILE_SIZE	The threshold file size at which a new single log file is created (it only works with the **PERFLIB_LOG_SINGLE_FILE** parameter). Whenever a flush of aggregate timing data causes the configured size to be exceeded, the log file is renamed and a new file is created for any subsequent data. The size is expressed in bytes. For example, 5242880 is equivalent to 512 Mb. If the parameter is not defined or set to 0, the file size defaults to 1 Gb.
PERFLIB_ALARM_CONFIG_FILE	How Perflib handles alarms. Perflib provides an example alarm file, **alarm_config.lst**, which shows how operation-specific configurations may be done.
PERFLIB_ALARM	The general alarm that triggers the logging of information regarding any opcode call that exceeds a particular elapsed time.
ENABLE_STATUS_DM_METRICS	Whether Prometheus generates metrics for the Oracle DM shared memory, front-end processes, and back-end processes. • **true**: Enables DM shared memory, front end, and back end metrics in Prometheus format. • **false**: Disables DM shared memory, front end, and back end metrics. This is the default.

Table 10-1 (Cont.) Perflib Environment Variables

Environment Variable	Description
PERFLIB_LOG_CORRELATION_IN_CALL_STACK	Whether Perflib adds the BRM correlation ID to call-stack logs. • **0**: Do not add correlation IDs to call-stack traces. • **1**: Add correlation IDs to call-stack traces. This is the default.
PERFLIB_FLIST_LOG_TO_STDOUT	Instructs Perflib to generate flist logs to standard output. • **0**: Writes opcode flists and stack trace logs to files. This is the default. • **1**: Writes opcode flists and stack trace logs to STDOUT.

Configuring OracleDB_Exporter to Scrape Database Metrics

You use OracleDB_Exporter to scrape metrics from your BRM database and export them to Prometheus. Prometheus can then read the metrics and display them in the a graphic format in Grafana.

To configure OracleDB_Exporter to scrape and export metrics from your BRM database:

1. Download and install the following external applications:
   - OracleDB_exporter. See https://github.com/iamseth/oracledb_exporter on the GitHub website.
   - Oracle database client.

   For the list of compatible software versions, see "BRM Cloud Native Deployment Software Compatibility" in *BRM Compatibility Matrix*.
2. Specify the BRM database metrics to scrape and export in the *Exporter_home***/default-metrics.toml** file, where *Exporter_home* is the directory in which you deployed OracleDB_Exporter.

   For more information, see https://github.com/iamseth/oracledb_exporter/blob/master/README.md on the GitHub website.
3. Open your **override-values.yaml** file for Prometheus.
4. Configure Prometheus to fetch performance data from OracleDB_exporter.

   To do so, copy and paste the following into your **override-values.yaml** file, replacing *hostname* with the host name of the machine on which OracleDB_exporter is deployed:

   ```
   static_configs:
   - targets: [hostname:33775']
   - job_name: 'oracledbexporter'
   static_configs:
   - targets: ['hostname:9161']
   ```
5. Save and close your file.
6. Run the **helm upgrade** command to update your Prometheus Helm chart release.

The metrics for your BRM database are available at **http://***hostname***:9161/metrics**.

Configuring Grafana for BRM Cloud Native

You can create a dashboard in Grafana for displaying the metric data for your BRM cloud native services.

Alternatively, you can use the sample dashboards that are included in the **oc-cn-docker-files-12.0.0.*x*.0.tgz** package. To use the sample dashboards, import the dashboard files from the **oc-cn-docker-files/samples/monitoring/** directory into Grafana. See "Export and Import" in the *Grafana Dashboards* documentation for more information.

Table 10-2 describes each sample dashboard.

Table 10-2 Sample Grafana Dashboards

File Name	Description
oc-cn-applications-dashboard.json	Provides a high-level view of all BRM components that have been installed, grouped by whether they are running or have failed.
ocbrm-batch-controller-dashboard.json	Allows you to view JVM-related metrics for the Batch Controller.
ocbrm-cm-dashboard.json	Allows you to view CPU and opcode-level metrics for the CM.
ocbrm-dm-ifw-dashboard.json	Allows you to view opcode-level, CPU usage, and memory usage metrics for the Account Synchronization DM.
ocbrm-dm-oracle-dashboard.json	Allows you to view opcode-level, CPU usage, and memory usage metrics for the Oracle DM.
ocbrm-dm-oracle-shm-dashboard.json	Allows you to view shared memory, front-end process, and back-end process metrics for the Oracle DM.
ocbrm-eai-js-dashboard.json	Allows you to view JVM and opcode-related metrics for the EAI JS.
ocbrm-overview-dashboard.json	Allows you to view metrics for BRM services at the pod, container, network, and input-output level.
ocbrm-rel-dashboard.json	Allows you to view JVM-related metrics for Rated Event (RE) Loader.
ocbrm-wsm-weblogic-server-dashboard.json	Allows you to view metrics for Web Services Manager.

Note:

For the sample dashboard to work properly, the datasource name for the WebLogic Domain must be **Prometheus**.

You can also configure Grafana to send alerts to your dashboard, an email address, or Slack when a problem occurs. For example, you could configure Grafana to send an alert when an opcode exceeds a specified number of errors. For information about setting up alerts, see "Grafana Alerts" in the Grafana documentation.

BRM Opcode Metric Group

Use the BRM opcode metric group to retrieve runtime information for BRM opcodes. Table 10-3 lists the metrics in this group.

Table 10-3 BRM Opcode Metrics

Metric Name	Metric Type	Metric Description	Pod
brm_opcode_calls_total	Counter	The total number of calls for a BRM opcode.	cm dm-oracle dm-ifw-sync dm-aq
brm_opcode_errors_total	Counter	The total number of errors when executing a BRM opcode.	cm dm-oracle dm-ifw-sync dm-aq
brm_opcode_exec_time_total	Counter	The total time taken to run a BRM opcode.	cm dm-oracle dm-ifw-sync dm-aq
brm_opcode_user_cpu_time_total	Counter	The total CPU time taken to run the BRM opcode in user space.	cm dm-oracle dm-ifw-sync dm-aq
brm_opcode_system_cpu_time_total	Counter	The total CPU time taken to run the BRM opcode in OS Kernel space.	cm dm-oracle dm-ifw-sync dm-aq
brm_opcode_records_total	Counter	The total number of records returned by the BRM opcode execution.	cm dm-oracle dm-ifw-sync dm-aq
brm_dmo_shared_memory_used_current	Gauge	The total number of shared memory blocks currently used by dm_oracle.	cm
brm_dmo_shared_memory_used_max	Counter	The maximum number of shared memory blocks currently used by dm_oracle.	cm
brm_dmo_shared_memory_free_current	Gauge	The total number of free shared memory blocks available to dm_oracle.	cm
brm_dmo_shared_memory_hwm	Gauge	The shared memory high water mark for dm_oracle.	cm
brm_dmo_shared_memory_bigsize_used_max	Counter	The maximum big size shared memory used by dm_oracle in bytes.	cm

Table 10-3 (Cont.) BRM Opcode Metrics

Metric Name	Metric Type	Metric Description	Pod
brm_dmo_shared_memory_bigsize_used_current	Gauge	The total big size shared memory used by dm_oracle in bytes.	cm
brm_dmo_shared_memory_bigsize_hwm	Gauge	Big size shared memory high water mark for dm_oracle in bytes.	cm
brm_dmo_front_end_connections_total	Gauge	The total number of connections for a dm_oracle front-end process.	cm
brm_dmo_front_end_max_connections_total	Counter	The maximum number of connections for a dm_oracle front-end process.	cm
brm_dmo_front_end_trans_done_total	Counter	The total number of transactions handled by the dm_oracle front-end process.	cm
brm_dmo_front_end_ops_done_total	Counter	The total number of operations handled by the dm_oracle front-end process.	cm
brm_dmo_back_end_ops_done_total	Counter	The total number of operations done by the dm_oracle back-end process.	cm
brm_dmo_back_end_ops_error_total	Counter	The total number of errors encountered by the dm_oracle back-end process.	cm
brm_dmo_back_end_trans_done_total	Counter	The total number of transactions handled by the dm_oracle back-end process.	cm
brm_dmo_back_end_trans_error_total	Counter	The total number of transaction errors in the dm_oracle back-end process.	cm
com_portal_js_JSMetrics_CurrentConnectionCount	Counter	The current count of concurrent connections to the Java Server from the CM.	cm (eai-java-server)
com_portal_js_JSMetrics_MaxConnectionCount	Counter	The maximum concurrent connections to the Java Server from the CM.	cm (eai-java-server)
com_portal_js_JSMetrics_SuccessfulOpcodeCount	Counter	The count of opcodes called from the CM, the execution of which succeeded in the Java Server.	cm (eai-java-server)
com_portal_js_JSMetrics_FailedOpcodeCount	Counter	The count of opcodes called from the CM, the execution of which failed in the Java Server.	cm (eai-java-server)
com_portal_js_JSMetrics_TotalOpcodeCount	Counter	The total count of opcodes called from the CM.	cm (eai-java-server)

Table 10-3 (Cont.) BRM Opcode Metrics

Metric Name	Metric Type	Metric Description	Pod
com_portal_js_JSMetrics_TotalOpcodeExecutionTime	Counter	The total time taken in, milliseconds, across all opcodes.	cm (eai-java-server)

11 Monitoring and Autoscaling Business Operations Center Cloud Native

Learn how to use external applications, such as Prometheus and Grafana, to monitor and autoscale Oracle Communications Business Operations Center in a cloud native environment.

Topics in this document:

About Monitoring and Autoscaling in Business Operations Center Cloud Native

You set up the monitoring of Business Operations Center and the autoscaling of its managed-server pods by using the following external applications:

- **WebLogic Monitoring Exporter**: Use this Oracle web application to scrape runtime information from Business Operations Center cloud native and then expose the metric data in Prometheus format. It exposes different WebLogic Mbeans metrics, such as memory usage and sessions count, that are required for monitoring and maintaining the Business Operations Center application.
- **Prometheus**: Use this open-source toolkit to scrape Business Operations Center metric data from WebLogic Monitoring Exporter and then store it in a time-series database. It can also be used to scale up or scale down your Business Operations Center pods based on memory and CPU usage.

  You can use a standalone version of Prometheus or Prometheus Operator.
- **Grafana**: Use this open-source tool to view on a graphical dashboard all Business Operations Center metric data that is stored in Prometheus.

Setting Up Monitoring and Autoscaling in Business Operations Center

To set up monitoring and autoscaling in Business Operations Center cloud native:

1. Deploy Prometheus in one of the following ways:
   - Deploy a standalone version of Prometheus in your cloud native environment. See "Installation" in the Prometheus documentation.
   - Deploy Prometheus Operator. See "prometheus-operator" on the GitHub website.

For the list of compatible software versions, see "BRM Cloud Native Deployment Software Compatibility" in *BRM Compatibility Matrix*.

2. Install Grafana. See "Install Grafana" in the Grafana documentation.

   For the list of compatible software versions, see "BRM Cloud Native Deployment Software Compatibility" in *BRM Compatibility Matrix*.

3. Configure WebLogic Monitoring Exporter to scrape metric data from Business Operations Center in your cloud native environment. See "Configuring WebLogic Monitoring Exporter to Scrape Metric Data".
4. Configure the Prometheus webhook to autoscale the Business Operations Center pods in your cloud native environment. See "Configuring webhook to Enable Autoscaling".
5. Configure one of the following to collect metric data and send alerts:
   - Standalone version of Prometheus. See "Configuring Standalone Prometheus for Business Operations Center".
   - Prometheus Operator. See "Configuring Prometheus Operator for Business Operations Center".
6. Configure Grafana for displaying Business Operations Center metric data. See "Creating Grafana Dashboards for Business Operations Center".

Configuring WebLogic Monitoring Exporter to Scrape Metric Data

You configure WebLogic Monitoring Exporter to scrape metric data for Business Operations Center by enabling monitoring of the application and by specifying whether to use it with Prometheus or Prometheus Operator.

When monitoring is enabled, WebLogic Monitoring Exporter scrapes WebLogic Server MBean metrics such as server status, web application session metrics, servlet metrics, JVM runtime metrics, and so on. See "WebLogic-Based Application Metrics" for a full list of metrics that are scraped. However, you can configure WebLogic Monitoring Exporter to scrape additional WebLogic Server MBeans to meet your business requirements.

To configure WebLogic Monitoring Exporter to scrape metric data for Business Operations Center cloud native:

1. Open your **override-values.yaml** file for **oc-cn-helm-chart**.
2. Set the **ocboc.boc.monitoring.isEnabled** key to **true**.
3. Set the **ocboc.boc.monitoring.operator.isEnabled** key to one of the following:
   - **true** if you are using Prometheus Operator.
   - **false** if you are using a standalone version of Prometheus. This is the default.
4. Optionally, configure WebLogic Monitoring Exporter to scrape additional metrics for Business Operations Center. To do so, set the **ocboc.boc.monitoring.queries** key to the full array of WebLogic Server MBeans to monitor, in YAML structure. For the list of possible MBeans, see MBean Reference for Oracle WebLogic Server in the Oracle WebLogic Server documentation.

Note:

Set the **queries** key to the full list of MBeans to scrape, including the default MBeans. That is, if you want to add one new metric, you must copy the default list from the domain's YAML file, add the new metric to that list, and then copy the full list to the **queries** key.

5. Set the other optional keys under **ocboc.boc.monitoring** as needed.

   For information about the other keys under **ocboc.boc.monitoring**, read the descriptions in the **oc-cn-helm-charts/values.yaml** file.

6. Save and close the file.
7. Run the **helm upgrade** command to update the BRM Helm release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
n BrmNameSpace
```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

   WebLogic Monitoring Exporter is started within the Business Operations Center WebLogic Server pod and begins scraping metric data for Business Operations Center.

   If you enabled Prometheus Operator, a ServiceMonitor is also deployed. The ServiceMonitor specifies how to monitor groups of services. Prometheus Operator automatically generates the scrape configuration based on this definition.

Configuring webhook to Enable Autoscaling

You can configure the webhook application to autoscale your Business Operations Center pods. When configured to do so, the webhook application waits for alerts from Prometheus Alertmanager. When it receives a specific alert status, the webhook application calls a script that performs the scaling action.

You can optionally configure the webhook application to monitor for additional alert statuses that trigger calls to your custom scripts.

To configure webhook to autoscale your Business Operations Center pods:

1. Open your **override-values.yaml** file for **oc-cn-helm-chart**.
2. Set the following keys to enable autoscaling:
   - **webhook.isEnabled**: Set this to **true**.
   - **webhook.logPath**: Set this to the path in which to write log files for the webhook application.
   - **webhook.scripts.mountpath**: Set this to the directory in which you will store any custom scripts to be run by the webhook application. The default is **/u01/script**.

- **webhook.wop.namespace**: Set this to the namespace for WebLogic Kubernetes Operator. See "Installing WebLogic Kubernetes Operator" in *BRM Cloud Native Deployment Guide*.
- **webhook.wop.sa**: Set this to the service account for the WebLogic Kubernetes Operator. The default is **default**.
- **webhook.wop.internalOperatorCert**: Set this to the WebLogic Kubernetes Operator certificate. To retrieve the certificate for this key, run the following command:

  ```
  kubectl -n operator describe configmap
  ```

  where *operator* is the namespace for WebLogic Kubernetes Operator.

For information about the other optional keys under the **webhook** section, read the descriptions in the **oc-cn-helm-charts/values.yaml** file.

3. If you want the webhook application to monitor for additional alert statuses and call your custom scripts, do the following:

   a. Copy your custom scripts to the **oc-cn-helm-chart/webhook_scripts** directory.

   b. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **webhook.jsonConfig** key to include the additional alerts to monitor and the scripts that are triggered when they occur. Use the following format:

   ```
   jsonConfig: {"alertName":"value", "alertStatus":["value"],
   "args":["arg1","arg2"], "script":"path/customScript",
   "workDirectory":"path"}
   ```

   Table 11-1 lists the possible values for each parameter.

   Table 11-1 Webhook Alerts

Alert Parameter	Description
alertName	Set this to the name of the alert to monitor, such as **clusterScaleUp**.
alertStatus	Set this to the alert's status that triggers a call to your custom script. For example: **firing**.

Table 11-1 (Cont.) Webhook Alerts

Alert Parameter	Description
args	Set this to the list of arguments to pass to your custom script. The arguments must be listed in the order in which they will appear in the script's command line. There are three types of arguments: • **static**: These arguments can be directly mapped while calling your script. For example: **"operator"** or **"operator-sa"**. • **custom labels**: Use the format **@@LABEL:***key-name***@@**, where *key-name* is an alert label passed in the alert notification. For example, to include the "domain_uid=boc-domain" argument, you would enter **"--domain_uid=@@LABEL:domain_uid@@"**. • **environment variables**: Use the format **@@ENV:***env-name***@@**, where *env-name* is the environment variable that is looked up. For example, to include the "--wls_domain_namespace=oc-cn-brm" argument, you would enter **"--wls_domain_namespace=@@ENV:NAMESPACE@@"**.
script	The name of the script to run along with its fully qualified path. For example: **/u01/script/scalingAction.sh**.
workDirectory	The script's current working directory. For example: **/u01/oracle/app**.

4. Save and close your **override-values.yaml** file.
5. Run the **helm upgrade** command to update your BRM Helm release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
n BrmNameSpace
```

The webhook application starts waiting for alerts from Prometheus Alertmanager.

Example: Configuring webhook to Autoscale Business Operations Center Pods

The following shows sample **override-values.yaml** entries for setting up the webhook application to perform autoscaling on your Business Operations Center pods:

```
webhook:
    isEnabled: true
    logPath: /u01/logs
    logLevel: INFO
    deployment:
        imageName: webhook
        imageTag: $BRM_VERSION
        imagePullPolicy: IfNotPresent
    scripts:
        mountPath: /u01/script
    wop:
        namespace: WME_Namespace
        sa: default
        internalOperatorCert: certificate
    jsonConfig: {"alertName":"clusterAlert", "alertStatus":["firing"],
"args":["arg1","arg2"], "script":"/u01/script/customAction.sh",
"workDirectory":"/u01/oracle/app"}
```

Configuring Standalone Prometheus for Business Operations Center

To configure a standalone version of Prometheus for Business Operations Center cloud native:

1. Open your **override-values.yaml** file for Prometheus.
2. Configure Prometheus to collect your Business Operations Center metrics exposed by WebLogic Monitoring Exporter.

   To do so, copy and paste the following into your file, replacing the variables with the appropriate values for your system:

```
extraScrapeConfigs: |
    - job_name: 'wls-domain1'
      kubernetes_sd_configs:
      - role: pod
      relabel_configs:
      - source_labels: [__meta_kubernetes_namespace]
        action: replace
        target_label: namespace
      - source_labels:
[__meta_kubernetes_pod_label_weblogic_domainUID,
__meta_kubernetes_pod_label_weblogic_clusterName]
        action: keep
        regex: boc-domain
      - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_path]
        action: replace
        target_label: __metrics_path__
        regex: (.+)
      - source_labels: [__address__,
__meta_kubernetes_pod_annotation_prometheus_io_port]
        action: replace
        regex: ([^:]+)(?::\d+)?;(\d+)
        replacement: $1:$2
        target_label: __address__
      - action: labelmap
        regex: __meta_kubernetes_pod_label_(.+)
      - source_labels: [__meta_kubernetes_pod_name]
        action: replace
        target_label: pod_name
      basic_auth:
        username: WebLogic_UserName
        password: WebLogic_Password
```

3. Configure the alert rules in Prometheus.

   To do so, copy and paste the following into your file, replacing the variables with the appropriate values for your system. However, do not change the alert names **clusterScaleUp** and **clusterScaleDown**.

The **clusterScaleUp** rule specifies to scale up the number of Business Operations Center managed server pods when the number of servers goes below two for two minutes. The **clusterScaledown** rule specifies to scale down the number of Business Operations Center managed servers pods when the number of servers goes below two for two minutes. For examples of other expressions you can use, see "Sample Prometheus Alert Rules for Business Operations Center".

```
serverFiles:
  alerts:
    groups:
      - name: node_rules
        rules:
          - alert: clusterScaleUp
            for: 2m
            expr: sum by(weblogic_domainUID, weblogic_clusterName)
(up{weblogic_domainUID="boc-domain"}) < 2
            labels:
              domain_uid: boc-domain
              severity: critical
            annotations:
              description: 'Server count is less than 2'
              summary: 'Some wls cluster is in warning state.'
          - alert: clusterScaleDown
            for: 2m
            expr: sum by(weblogic_domainUID, weblogic_clusterName)
(up{weblogic_domainUID="boc-domain"}) > 3
            labels:
              domain_uid: boc-domain
              severity: critical
            annotations:
              description: 'Server count is greater 3'
              summary: 'Some wls cluster is in warning state.'
```

4. Configure Alertmanager to send alerts to the webhook application.

   To do so, copy and paste the following into your file, replacing the variables with the appropriate values for your system. However, do not change the alert names **clusterScaleUp** and **clusterScaleDown**.

   For the **url** key, use the following syntax: **http://webhook.***WLS_NameSpace***.svc.cluster.local:8080/action**, where *WLS_NameSpace* is the namespace for your WebLogic Server domain.

```
alertmanagerFiles:
  alertmanager.yml:
    global:
      resolve_timeout: 5m
    route:
      group_by: ['alertname']
      receiver: 'null'
      group_wait: 10s
      group_interval: 10s
      repeat_interval: 5m
      routes:
      - match:
          alertname: clusterScaleUp
```

```
        receiver: 'web.hook'
      - match:
          alertname: clusterScaleDown
        receiver: 'web.hook'
    receivers:
    - name: 'web.hook'
      webhook_configs:
      - send_resolved: false
        url: 'http://webhook.oc-cn-brm.svc.cluster.local:8080/
action'
    - name: 'null'
```

5. Save and close your **override-values.yaml** file for Prometheus.
6. Run the **helm upgrade** command to update your Prometheus Helm chart.

Configuring Prometheus Operator for Business Operations Center

To configure Prometheus Operator for Business Operations Center cloud native:

1. Open your **override-values.yaml** file for Prometheus Operator.
2. Configure the alert rules for Prometheus Operator.

   To do so, copy and paste the following **additionalPrometheusRulesMap** section into your file, replacing the variables with the appropriate values for your system. However, do not change the alert names **clusterScaleUp** and **clusterScaleDown**.

   The **clusterScaleUp** rule specifies to scale up the number of Business Operations Center managed server pods when the number of servers goes below two for two minutes. The **clusterScaledown** rule specifies to scale down the number of Business Operations Center managed servers pods when the number of servers goes below two for two minutes. For examples of other expressions you can use, see "Sample Prometheus Alert Rules for Business Operations Center".

```
## Provide custom recording or alerting rules to be deployed into
the cluster.
##
additionalPrometheusRulesMap:
  - rule-name: Custom-rule
    groups:
  - name: custom-alert.rules
    rules:
  - alert: clusterScaleUp
    annotations:
      message: WLS cluster has less than 2 running server for more
than 2 minutes.
    expr: sum by(weblogic_domainUID)
(up{serviceType="SERVER",weblogic_clusterName="cluster-1",weblogic_d
omainUID="boc-domain"}) < 2
    for: 2m
    labels:
      domain_uid: boc-domain
      severity: critical
  - alert: clusterScaleDown
    annotations:
      message: WLS cluster has more than 3 running servers for more
```

```
than 2 minutes.
    expr: sum by(weblogic_domainUID)
(up{serviceType="SERVER",weblogic_clusterName="cluster-1",weblogic_domainU
ID="boc-domain"}) > 3
    for: 2m
    labels:
      domain_uid: boc-domain
      severity: critical
```

3. Configure Prometheus Operator to send alerts to the webhook application in WebLogic Monitoring Exporter.

   To do so, copy and paste the following **alertmanager** section into your file, replacing the variables with the appropriate values for your system. However, do not change the alert names **clusterScaleUp** and **clusterScaleDown**.

   For the **url** key, use the following syntax: **http://webhook.***BrmNameSpace***.svc.cluster.local:8080/action**, where *BrmNameSpace* is the namespace for your BRM Kubernetes objects.

```
alertmanager:
  config:
    global:
      resolve_timeout: 5m
    route:
      group_by: ['alertname']
      group_wait: 10s
      group_interval: 10s
      repeat_interval: 5m
      receiver: 'null'
      routes:
      - match:
          alertname: clusterScaleUp
        receiver: 'web.hook'
      - match:
          alertname: clusterScaleDown
        receiver: 'web.hook'
    receivers:
    - name: 'null'
    - name: 'web.hook'
      webhook_configs:
      - send_resolved: false
        url: 'http://webhook.oc-cn-brm.svc.cluster.local:8080/action'
```

4. Save and close your **override-values.yaml** file for Prometheus Operator.
5. Run the **helm upgrade** command to update your Prometheus Operator Helm chart.

Creating Grafana Dashboards for Business Operations Center

Create a dashboard in Grafana for displaying your Business Operations Center metric data. You can alternatively use the sample dashboard JSON model that is included in the **oc-cn-docker-files-12.0.0.***x***.0.tgz** package.

Note:

For the sample dashboard to work properly, the datasource name for the WebLogic Domain must be **Prometheus**.

To use the sample dashboard, import the **oc-cn-docker-files/samples/monitoring/ocboc-boc-dashboard.json** dashboard file into Grafana. See "Export and Import" in the *Grafana Dashboards* documentation for more information.

Sample Prometheus Alert Rules for Business Operations Center

You can use custom expressions for your Prometheus alert rules when setting up autoscaling in Business Operations Center.

Sample Cluster Scale Up Expressions

To raise an alert when the average CPU usage across managed servers is greater than 70% for more than two minutes:

```
avg(avg_over_time(wls_jvm_process_cpu_load{weblogic_clusterName=~".+",w
eblogic_domainUID="boc-domain",weblogic_serverName=~".+"}[2m]))*100 >
70
```

To raise an alert when the average memory usage across managed servers is greater than 70% for more than two minutes:

```
100 -
avg(avg_over_time(wls_jvm_heap_free_percent{weblogic_domainUID="boc-
domain",weblogic_clusterName=~".+",weblogic_serverName=~".+"}[2m])) >
70
```

To raise an alert when the CPU usage is greater than 70% and memory usage is greater than 70%:

```
avg(avg_over_time(wls_jvm_process_cpu_load{weblogic_clusterName=~".+",w
eblogic_domainUID="boc-domain",weblogic_serverName=~".+"}[2m])) * 100
> 70 and on() 100 -
avg(avg_over_time(wls_jvm_heap_free_percent{weblogic_clusterName=~".+",
weblogic_domainUID="boc-domain",weblogic_serverName=~".+"}[2m])) > 70
```

Sample Cluster Scale Down Expressions

To raise an alert when the CPU usage is less than 40%, memory usage is less than 40%, and the number of managed servers is equal to 5:

```
avg(avg_over_time(wls_jvm_process_cpu_load{weblogic_clusterName=~".+",w
eblogic_domainUID="boc-domain",weblogic_serverName=~".+"}[2m])) * 100
< 40 and on() 100 -
```

```
avg(avg_over_time(wls_jvm_heap_free_percent{weblogic_clusterName=~".+",weblog
ic_domainUID="boc-domain",weblogic_serverName=~".+"}[2m])) < 40 and on() sum
by(weblogic_domainUID)
(up{weblogic_clusterName="cluster-1",weblogic_domainUID="boc-domain"}) == 5
```

12

Monitoring and Autoscaling Billing Care Cloud Native

Learn how to use external applications, such as Prometheus and Grafana, to monitor and autoscale Oracle Communications Billing Care in a cloud native environment.

Topics in this document:

- About Monitoring and Autoscaling in Billing Care Cloud Native
- Setting Up Monitoring and Autoscaling in Billing Care and Billing Care REST API
- Sample Prometheus Alert Rules for Billing Care and Billing Care REST API

About Monitoring and Autoscaling in Billing Care Cloud Native

You set up the monitoring of Billing Care and the Billing Care REST API and the autoscaling of their managed-server pods by using the following external applications:

- **WebLogic Monitoring Exporter**: Use this Oracle web application to scrape runtime information from Billing Care and the Billing Care REST API and then expose the metric data in Prometheus format. It exposes different WebLogic Mbeans metrics, such as memory usage and sessions count, that are required for monitoring and maintaining the Billing Care and Billing Care REST API applications.
- **Prometheus**: Use this open-source toolkit to scrape metric data from WebLogic Monitoring Exporter and then store it in a time-series database. It can also be used to scale up or scale down your Billing Care managed server pods based on memory and CPU usage.

  You can use a standalone version of Prometheus or Prometheus Operator.
- **Grafana**: Use this open-source tool to view on a graphical dashboard all Billing Care and Billing Care REST API metric data stored in Prometheus.

Setting Up Monitoring and Autoscaling in Billing Care and Billing Care REST API

To set up the monitoring and autoscaling of Billing Care and the Billing Care REST API in a cloud native environment:

1. Deploy Prometheus in one of the following ways:
   - Deploy a standalone version of Prometheus in your cloud native environment. See "Installation" in the Prometheus documentation.
   - Deploy Prometheus Operator. See "prometheus-operator" on the GitHub website.

   For the list of compatible software versions, see "BRM Cloud Native Deployment Software Compatibility" in *BRM Compatibility Matrix*.
2. Install Grafana. See "Install Grafana" in the Grafana documentation.

For the list of compatible software versions, see "BRM Cloud Native Deployment Software Compatibility" in *BRM Compatibility Matrix*.

3. Configure WebLogic Monitoring Exporter to scrape metric data from Billing Care in your cloud native environment. See "Configuring WebLogic Monitoring Exporter to Scrape Metric Data".
4. Configure webhook to enable the autoscaling of Billing Care and Billing Care REST API pods in your cloud native environment. See "Configuring Webhook to Enable Autoscaling".
5. Configure one of the following to collect metric data and send alerts:
   - Standalone version of Prometheus. See "Configuring Standalone Prometheus for Billing Care".
   - Prometheus Operator. See "Configuring Prometheus Operator for Billing Care".
6. Configure Grafana for displaying Billing Care metric data. See "Creating Grafana Dashboards for Billing Care and Billing Care REST API".

Configuring WebLogic Monitoring Exporter to Scrape Metric Data

You configure WebLogic Monitoring Exporter to scrape metric data for Billing Care and the Billing Care REST API by enabling monitoring in each application and by specifying whether to use each application with Prometheus or Prometheus Operator.

When monitoring is enabled, WebLogic Monitoring Exporter scrapes WebLogic Server MBean metrics such as server status, web application session metrics, servlet metrics, JVM runtime metrics, and so on. See "WebLogic-Based Application Metrics" for a full list of metrics that are scraped. However, you can configure WebLogic Monitoring Exporter to scrape additional WebLogic Server MBeans to meet your business requirements.

To configure WebLogic Monitoring Exporter to scrape metric data for Billing Care and the Billing Care REST API in a cloud native environment:

1. Open your **override-values.yaml** file for **oc-cn-helm-chart**.
2. Configure monitoring for Billing Care cloud native:
   - Set the **ocbc.bc.monitoring.isEnabled** key to **true**.
   - Set the **ocbc.bc.monitoring.operator.isEnabled** key to **true** if you are using Prometheus Operator, or **false** if you are using a standalone version of Prometheus. The default is **false**.
3. Configure monitoring for the Billing Care REST API:
   - Set the **ocbc.bcws.monitoring.isEnabled** key to **true**.
   - Set the **ocbc.bcws.monitoring.operator.isEnabled** key to **true** if you are using Prometheus Operator, or **false** if you are using a standalone version of Prometheus. The default in **false**.
4. Optionally, configure WebLogic Monitoring Exporter to scrape additional metrics. To do so, set the following keys to the full array of WebLogic Server MBeans to monitor, in YAML structure. For the list of possible MBeans, see MBean Reference for Oracle WebLogic Server in the Oracle WebLogic Server documentation.
   - For Billing Care: **ocbc.bc.monitoring.queries**

- For the Billing Care REST API: **ocbc.bcws.monitoring.queries**

> **Note:**
>
> Set the **queries** key to the full list of MBeans to scrape, including the default MBeans. That is, if you want to add one new metric, you must copy the default list from the domain's YAML file, add the new metric to that list, and then copy the full list to the **queries** key.

5. Set the other optional monitoring keys as needed.

   For information about the other keys, read the descriptions in the **oc-cn-helm-charts/values.yaml** file.
6. Save and close the file.
7. Run the **helm upgrade** command to update the BRM Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
   n BrmNameSpace
   ```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

   WebLogic Monitoring Exporter is started within the Billing Care and Billing Care REST API WebLogic Server pods and begins scraping metric data for Billing Care and the Billing Care REST API.

   If you enabled Prometheus Operator, a ServiceMonitor is also deployed. The ServiceMonitor specifies how to monitor groups of services. Prometheus Operator automatically generates the scrape configuration based on this definition.

Configuring Webhook to Enable Autoscaling

You can configure the webhook application to autoscale your Billing Care and Billing Care REST API pods. When configured to do so, the webhook application waits for alerts from Prometheus Alertmanager. When it receives a specific alert status, the webhook application calls a script that performs the scaling action.

You can optionally configure the webhook application to monitor for additional alert statuses that trigger calls to your custom scripts.

To configure WebLogic Monitoring Exporter to autoscale your Billing Care pods:

1. Open your **override-values.yaml** file for **oc-cn-helm-chart**.
2. Set the following keys to enable autoscaling:
   - **webhook.isEnabled**: Set this to **true**.
   - **webhook.logPath**: Set this to the path in which to write log files for the webhook application.

- **webhook.scripts.mountpath**: Set this to the directory in which you will store any custom scripts to be run by the webhook application. The default is **/u01/script**.
- **webhook.wop.namespace**: Set this to the namespace for WebLogic Kubernetes Operator. See "Installing WebLogic Kubernetes Operator" in *BRM Cloud Native Deployment Guide*.
- **webhook.wop.sa**: Set this to the service account for the WebLogic Kubernetes Operator. The default is **default**.
- **webhook.wop.internalOperatorCert**: Set this to the WebLogic Kubernetes Operator certificate. To retrieve the certificate for this key, run the following command:

  ```
  kubectl -n operator describe configmap
  ```

  where *operator* is the namespace for WebLogic Kubenetes Operator.

For information about the other optional keys under the **webhook** section, read the descriptions in the **oc-cn-helm-charts/values.yaml** file.

3. If you want the webhook application to monitor for additional alert statuses and call your custom scripts, do the following:

   a. Copy your custom scripts to the **oc-cn-helm-chart/webhook_scripts** directory.

   > **Note:**
   >
   > You can configure the mount path for your custom scripts by using the **webhook.scripts.mountPath** key.

   b. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the **webhook.jsonConfig** key to include the additional alerts to monitor and the scripts that are triggered when they occur. Use the following format:

   ```
   jsonConfig: {"alertName":"value", "alertStatus":["value"],
   "args":["arg1","arg2"], "script":"path/customScript",
   "workDirectory":"path"}
   ```

   Table 12-1 lists the possible values for each parameter.

Table 12-1 Webhook Alerts

Alert Parameter	Description
alertName	Set this to the name of the alert to monitor, such as **clusterScaleUp**.
alertStatus	Set this to the alert's status that triggers a call to your custom script. For example: **firing**.

Table 12-1 (Cont.) Webhook Alerts

Alert Parameter	Description
args	Set this to the list of arguments to pass to your custom script. The arguments must be listed in the order in which they will appear in the script's command line. There are three types of arguments: • **static**: These arguments can be directly mapped while calling your script. For example: **"operator"** or **"operator-sa"**. • **custom labels**: Use the format @@**LABEL:***key-name*@@, where *key-name* is an alert label passed in the alert notification. For example, to include the "domain_uid=bc-domain" argument, you would enter **"--domain_uid=@@LABEL:domain_uid@@"**. • **environment variables**: Use the format @@**ENV:***env-name*@@, where *env-name* is the environment variable that is looked up. For example, to include the "--wls_domain_namespace=oc-cn-brm" argument, you would enter **"--wls_domain_namespace=@@ENV:NAMESPACE@@"**.
script	The name of the script to run along with its fully qualified path. For example: **/u01/script/scalingAction.sh**.
workDirectory	The script's current working directory. For example: **/u01/oracle/app**.

4. Save and close your **override-values.yaml** file.
5. Run the **helm upgrade** command to update your BRM Helm release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
n BrmNameSpace
```

The webhook application starts waiting for alerts from Prometheus Alertmanager.

Example: Configuring webhook to Autoscale Billing Care Pods

The following shows sample **override-values.yaml** entries for setting up the webhook application to perform autoscaling on your Billing Care and Billing Care REST API pods:

```
webhook:
    isEnabled: true
    logPath: /u01/logs
    logLevel: INFO
    deployment:
        imageName: webhook
        imageTag: $BRM_VERSION
        imagePullPolicy: IfNotPresent
    scripts:
        mountPath: /u01/script
    wop:
        namespace: WebLogicKubernetesOperator_Namespace
        sa: default
        internalOperatorCert: certificate
    jsonConfig: {"alertName":"clusterAlert", "alertStatus":["firing"],
"args":["arg1","arg2"], "script":"/u01/script/customAction.sh",
"workDirectory":"/u01/oracle/app"}
```

Configuring Standalone Prometheus for Billing Care

To configure a standalone version of Prometheus for Billing Care and the Billing Care REST API:

1. Open your **override-values.yaml** file for Prometheus.
2. Configure Prometheus to scrape the required metrics exposed by WebLogic Monitoring Exporter.

   To do so, copy and paste the following into your file, replacing the variables with the appropriate values for your system:

```
extraScrapeConfigs: |
  - job_name: 'wls-domain1'
    kubernetes_sd_configs:
  - role: pod
    relabel_configs:
  - source_labels: [__meta_kubernetes_namespace]
    action: replace
    target_label: namespace
  - source_labels: [__meta_kubernetes_pod_label_weblogic_domainUID,
__meta_kubernetes_pod_label_weblogic_clusterName]
    action: keep
    regex: billingcare-domain
  - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_path]
    action: replace
    target_label: __metrics_path__
    regex: (.+)
  - source_labels: [__address__,
__meta_kubernetes_pod_annotation_prometheus_io_port]
    action: replace
    regex: ([^:]+)(?::\d+)?;(\d+)
    replacement: $1:$2
    target_label: __address__
  - action: labelmap
    regex: __meta_kubernetes_pod_label_(.+)
  - source_labels: [__meta_kubernetes_pod_name]
    action: replace
    target_label: pod_name
    basic_auth:
      username: username
      password: password

  - job_name: 'wls-domain2'
    kubernetes_sd_configs:
  - role: pod
    relabel_configs:
  - source_labels: [__meta_kubernetes_namespace]
    action: replace
    target_label: namespace
  - source_labels: [__meta_kubernetes_pod_label_weblogic_domainUID,
__meta_kubernetes_pod_label_weblogic_clusterName]
    action: keep
    regex: bcws-domain
  - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_path]
    action: replace
    target_label: __metrics_path__
    regex: (.+)
  - source_labels: [__address__,
__meta_kubernetes_pod_annotation_prometheus_io_port]
    action: replace
    regex: ([^:]+)(?::\d+)?;(\d+)
    replacement: $1:$2
    target_label: __address__
  - action: labelmap
    regex: __meta_kubernetes_pod_label_(.+)
  - source_labels: [__meta_kubernetes_pod_name]
    action: replace
    target_label: pod_name
    basic_auth:
      username: username
      password: password
```

where *username* and *password* is your WebLogic Server user name and password.

3. Configure the alert rules in Prometheus.

   To do so, copy and paste the following into your file, replacing the variables with the appropriate values for your system. However, do not change the alert names **clusterScaleUp** and **clusterScaleDown**.

   The **clusterScaleUp** rule specifies to scale up the number of Billing Care and Billing Care REST API managed server pods when the number of servers goes below two for two minutes. The **clusterScaledown** rule specifies to scale down the number of Billing Care and Billing Care REST API managed server pods when the number of servers goes below two for two minutes. For examples of other expressions you can use, see "Sample Prometheus Alert Rules for Billing Care and Billing Care REST API".

```
serverFiles:
  alerts:
    groups:
      - name: node_rules
        rules:
          - alert: clusterScaleUp
            for: 2m
            expr: sum by(weblogic_domainUID, weblogic_clusterName)
(up{weblogic_domainUID="billingcare-domain"}) < 2
            labels:
              domain_uid: billingcare-domain
              severity: critical
            annotations:
              description: 'Server count is less than 2'
              summary: 'Some wls cluster is in warning state.'
          - alert: clusterScaleDown
            for: 2m
            expr: sum by(weblogic_domainUID, weblogic_clusterName)
(up{weblogic_domainUID="billingcare-domain"}) > 3
            labels:
              domain_uid: billingcare-domain
              severity: critical
            annotations:
              description: 'Server count is greater than
3'
              summary: 'Some wls cluster is in warning state.'

          - alert: clusterScaleUp
            for: 2m
            expr: sum by(weblogic_domainUID, weblogic_clusterName)
(up{weblogic_domainUID="bcws-domain"}) < 2
            labels:
              domain_uid: bcws-domain
              severity: critical
            annotations:
              description: 'Server count is less than
2'
              summary: 'Some wls cluster is in warning
state.'
          - alert: clusterScaleDown
            for: 2m
            expr: sum by(weblogic_domainUID, weblogic_clusterName)
(up{weblogic_domainUID="bcws-domain"}) > 3
            labels:
              domain_uid: bcws-domain
              severity: critical
            annotations:
              description: 'Server count is greater than
3'
              summary: 'Some wls cluster is in warning state.'
```

4. Configure Prometheus Alertmanager to send alerts to the webhook application.

   To do so, copy and paste the following into your file, replacing the variables with the appropriate values for your system. However, do not change the alert names **clusterScaleUp** and **clusterScaleDown**.

   For the **url** key, use the following syntax: **http://webhook.**_BRMNameSpace_**.svc.cluster.local:8080/action**, where _BRMNameSpace_ is the namespace for your BRM Kubernetes objects.

```
alertmanagerFiles:
  alertmanager.yml:
    global:
      resolve_timeout: 5m
    route:
      group_by: ['alertname']
      receiver: 'null'
      group_wait: 10s
      group_interval: 10s
      repeat_interval: 5m
      routes:
      - match:
        alertname: clusterScaleUp
        receiver: 'web.hook'
      - match:
        alertname: clusterScaleDown
        receiver: 'web.hook'
        receivers:
      - name: 'web.hook'
        webhook_configs:
      - send_resolved: false
        url: 'http://webhook.oc-cn-brm.svc.cluster.local:8080/action'
      - name: 'null'
```

5. Save and close your **override-values.yaml** file for Prometheus.
6. Run the **helm upgrade** command to update your Prometheus Helm chart.

Configuring Prometheus Operator for Billing Care

To configure Prometheus Operator for Billing Care cloud native:

1. Open your **override-values.yaml** file for Prometheus Operator.
2. Configure the alert rules for Prometheus Operator.

   To do so, copy and paste the following **additionalPrometheusRulesMap** section into your file, replacing the variables with the appropriate values for your system. However, do not change the alert names **clusterScaleUp** and **clusterScaleDown**.

   The **clusterScaleUp** rule specifies to scale up the number of managed server Billing Care or Billing Care REST API pods when the number of servers goes below two for two minutes. The **clusterScaledown** rule specifies to scale down the number of Billing Care or Billing Care REST API managed server pods when the number of servers goes below two for two minutes. For examples of other expressions you can use, see "Sample Prometheus Alert Rules for Billing Care and Billing Care REST API".

```
## Provide custom recording or alerting rules to be deployed into the
cluster.
##

additionalPrometheusRulesMap:
  - rule-name: Custom-rule
    groups:
  - name: custom-alert.rules
    rules:
    - alert: clusterScaleUp
      annotations:
        message: WLS cluster has less than 2 running server for more than
2 minutes.
      expr: sum by(weblogic_domainUID)
(up{serviceType="SERVER",weblogic_clusterName="cluster-1",weblogic_domainU
ID="billingcare-domain"}) < 2
      for: 2m
      labels:
        domain_uid: billingcare-domain
        severity: critical
    - alert: clusterScaleDown
      annotations:
        message: WLS cluster has more than 3 running servers for
more than 2 minutes.
      expr: sum by(weblogic_domainUID)
(up{serviceType="SERVER",weblogic_clusterName="cluster-1",weblogic_d
omainUID="billingcare-domain"}) > 3
      for: 2m
      labels:
        domain_uid: billingcare-domain
        severity: critical
    - alert: clusterScaleUp
      annotations:
        message: WLS cluster has less than 2 running server for
more than 2 minutes.
      expr: sum by(weblogic_domainUID)
(up{serviceType="SERVER",weblogic_clusterName="cluster-1",weblogic_d
omainUID="bcws-domain"}) < 2
      for: 2m
      labels:
        domain_uid: bcws-domain
        severity: critical
    - alert: clusterScaleDown
      annotations:
        message: WLS cluster has more than 3 running servers for
more than 2 minutes.
      expr: sum by(weblogic_domainUID)
(up{serviceType="SERVER",weblogic_clusterName="cluster-1",weblogic_d
omainUID="bcws-domain"}) > 3
      for: 2m
      labels:
        domain_uid: bcws-domain
        severity: critical
```

3. Configure Prometheus Operator to send alerts to the webhook application in WebLogic Monitoring Exporter.

   To do so, copy and paste the following **alertmanager** section into your file, replacing the variables with the appropriate values for your system. However, do not change the alert names **clusterScaleUp** and **clusterScaleDown**.

   For the **url** key, use the following syntax: **http://webhook.***BrmNameSpace***.svc.cluster.local:8080/action**, where *BrmNameSpace* is the namespace for your BRM Kubernetes objects.

```
alertmanager:
  config:
    global:
      resolve_timeout: 5m
    route:
      group_by: ['alertname']
      group_wait: 10s
      group_interval: 10s
      repeat_interval: 5m
      receiver: 'null'
      routes:
      - match:
          alertname: clusterScaleUp
        receiver: 'web.hook'
      - match:
          alertname: clusterScaleDown
        receiver: 'web.hook'
    receivers:
    - name: 'null'
    - name: 'web.hook'
      webhook_configs:
      - send_resolved: false
        url: 'http://webhook.oc-cn-brm.svc.cluster.local:8080/action'
```

4. Save and close your **override-values.yaml** file for Prometheus Operator.
5. Run the **helm upgrade** command to update your Prometheus Operator Helm chart.

Creating Grafana Dashboards for Billing Care and Billing Care REST API

You can create a dashboard in Grafana for displaying your Billing Care and Billing Care REST API metric data.

Alternatively, you can use the sample dashboards that are included in the **oc-cn-docker-files-12.0.0.*x*.0.tgz** package. To use the sample dashboards, import the following dashboard files into Grafana. See "Export and Import" in the *Grafana Dashboards* documentation for more information.

- Billing Care: **oc-cn-docker-files/samples/monitoring/ocbc-billingcare-dashboard.json**
- Billing Care REST API: **oc-cn-docker-files/samples/monitoring/ocbc-billingcare-rest-api-dashboard.json**

Note:

For the sample dashboards to work properly, the datasource name for the WebLogic Domain must be **Prometheus**.

Sample Prometheus Alert Rules for Billing Care and Billing Care REST API

You can use custom expressions for your Prometheus alert rules when setting up autoscaling in Billing Care and the Billing Care REST API.

Sample Scale Up Expressions

To raise an alert when the average CPU usage across managed servers for more than 2 minutes is greater than 70%:

- For a Billing Care REST API domain:

  ```
  avg(avg_over_time(wls_jvm_process_cpu_load{weblogic_clusterName=~".+",weblogic_doma
  inUID="bcws-domain",weblogic_serverName=~".+"}[2m]))*100> 70
  ```

- For a Billing Care domain:

```
avg(avg_over_time(wls_jvm_process_cpu_load{weblogic_clusterName=~".+",weblogi
c_domainUID="billingcare-domain",weblogic_serverName=~".+"}[2m]))*100 > 70
```

To raise an alert when the average memory usage over 2 minutes across managed servers is greater than 70%:

- For a Billing Care REST API domain:

  ```
  100 -
  avg(avg_over_time(wls_jvm_heap_free_percent{weblogic_domainUID="bcws
  -domain",weblogic_clusterName=~".+",weblogic_serverName=~".+"}
  [2m])) > 70
  ```

- For a Billing Care domain:

  ```
  100 -
  avg(avg_over_time(wls_jvm_heap_free_percent{weblogic_domainUID="bill
  ingcare-
  domain",weblogic_clusterName=~".+",weblogic_serverName=~".+"}[2m]))
  > 70
  ```

To raise an alert when the CPU usage is greater than 70% and the memory usage is greater than 70%:

- For a Billing Care REST API domain:

  ```
  avg(avg_over_time(wls_jvm_process_cpu_load{weblogic_clusterName=~".+
  ",weblogic_domainUID="bcws-domain",weblogic_serverName=~".+"}
  [2m]))* 100 > 70 and on() 100 -
  avg(avg_over_time(wls_jvm_heap_free_percent{weblogic_clusterName=~".
  +",weblogic_domainUID="bcws-domain",weblogic_serverName=~".+"}
  [2m]))> 70
  ```

- For a Billing Care domain:

  ```
  avg(avg_over_time(wls_jvm_process_cpu_load{weblogic_clusterName=~".+
  ",weblogic_domainUID="billingcare-domain",weblogic_serverName=~".+"}
  [2m]))* 100 > 70 and on() 100 -
  avg(avg_over_time(wls_jvm_heap_free_percent{weblogic_clusterName=~".
  +",weblogic_domainUID="billingcare-
  domain",weblogic_serverName=~".+"}[2m]))> 70
  ```

Sample Scale Down Expressions

To raise an alert when the CPU usage is less than 40%, memory usage is less than 40%, and the number of managed servers is equal to 5 for two minutes:

- For a Billing Care REST API domain:

  ```
  avg(avg_over_time(wls_jvm_process_cpu_load{weblogic_clusterName=~".+
  ",weblogic_domainUID="bcws-domain",weblogic_serverName=~".+"}
  [2m]))* 100 < 40 and on() 100 -
  avg(avg_over_time(wls_jvm_heap_free_percent{weblogic_clusterName=~".
  +",weblogic_domainUID="bcws-domain",weblogic_serverName=~".+"}
  [2m]))< 40 and on() sum by(weblogic_domainUID)
  ```

```
(up{weblogic_clusterName="cluster-1",weblogic_domainUID="bcws-domain"})
==5
```

- For a Billing Care domain:

```
avg(avg_over_time(wls_jvm_process_cpu_load{weblogic_clusterName=~".+",webl
ogic_domainUID="billingcare-domain",weblogic_serverName=~".+"}[2m]))* 100
< 40 and on() 100 -
avg(avg_over_time(wls_jvm_heap_free_percent{weblogic_clusterName=~".+",web
logic_domainUID="billingcare-domain",web
```

13

Monitoring BRM REST Services Manager Cloud Native

Learn how to use external applications, such as Prometheus, Grafana, and Helidon MP, to monitor BRM REST Services Manager in a cloud native environment.

Topics in this document:

- About Monitoring BRM REST Services Manager Cloud Native
- Setting Up Monitoring for BRM REST Services Manager
- Creating Grafana Dashboards for BRM REST Services Manager
- Modifying Prometheus and Grafana Alert Rules After Deployment
- About REST Endpoints for Monitoring BRM REST Services Manager

About Monitoring BRM REST Services Manager Cloud Native

You set up monitoring for BRM REST Services Manager by using the following applications:

- **Helidon MP**: Use this Eclipse Microprofile application to run health checks and collect metrics. Helidon MP is configured and ready to use in the BRM REST Services Manager deployment package.

  For information about using the health check and metrics endpoints, see "About REST Endpoints for Monitoring BRM REST Services Manager". For more information about Helidon MP, see "Helidon MP Introduction" in the Helidon MP documentation.

- **Prometheus**: Use this open-source toolkit to scrape metric data and then store it in a time-series database. Use Prometheus Operator for BRM REST Services Manager.

  See "prometheus-operator" on GitHub.

- **Grafana**: Use this open-source tool to view on a graphical dashboard all BRM REST Services Manager metric data stored in Prometheus.

  See "Grafana Support for Prometheus" in the Prometheus documentation for information about using Grafana and Prometheus together.

Setting Up Monitoring for BRM REST Services Manager

To set up monitoring for BRM REST Services Manager cloud native:

1. Install Prometheus Operator:

   a. Ensure that BRM cloud native prerequisite software, such as the Kubernetes cluster and Helm, are running, and that Git is installed on the node that runs the Helm chart.

   b. Create a namespace for monitoring. For example:

   ```
   kubectl create namespace monitoring
   ```

c. Set the HTTP_PROXY environment variable on all cluster nodes with the following command:

```
export HTTP_PROXY="proxy_host"
export HTTPS_PROXY=$HTTP_PROXY
```

where *proxy_host* is the hostname or IP address of your proxy server.

d. Download the Prometheus Operator helm charts with the following commands:

```
helm repo add stable https://charts.helm.sh/stable
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo update
helm fetch prometheus-community/kube-prometheus-stack
```

e. Unset the HTTP_PROXY environment variable with the following command:

```
unset HTTP_PROXY
unset HTTPS_PROXY
```

f. Create an **override-values.yaml** file for Prometheus Operator and configure optional values to:

- Add alert rules, such as the two rules in the sample below.
- Make Prometheus, Alert Manager, and Grafana accessible outside the cluster and host machine by changing the service type to **LoadBalancer**.
- Enable Grafana to send email alerts.

The following sample **override-values.yaml** shows alert rules and configuration options.

```
additionalPrometheusRulesMap:
  - rule-name: BRM-RSM-rule
    groups:
    - name: brm-rsm-alert-rules
      rules:
      - alert: CPU_UsageWarning
        annotations:
          message: CPU has reached 80% utilization
        expr: avg without(cpu)
(rate(node_cpu_seconds_total{job="node-exporter",
instance="instance", mode!="idle"}[5m])) > 0.8
        for: 5m
        labels:
          severity: critical
      - alert: Memory_UsageWarning
        annotations:
          message: Memory has reached 80% utilization
        expr: node_memory_MemTotal_bytes{job="node-exporter",
instance="instance"} - node_memory_MemFree_bytes{job="node-
exporter", instance="instance"} -
node_memory_Cached_bytes{job="node-
exporter",instance="instance"} -
```

```
node_memory_Buffers_bytes{job="node-exporter", instance="instance"} >
22322927872
        for: 5m
        labels:
          severity: critical
alertmanager:
  service:
    type: LoadBalancer
grafana:
  service:
    type: LoadBalancer
  grafana.ini:
    smtp:
      enabled: true
      host: email_host
      user: "email_address"
      password: "password"
      skip_verify: true
prometheus:
  service:
    type: LoadBalancer
```

For details about the default Prometheus Operator values to base your **override-values.yaml** on, see "prometheus-operator/values.yaml" on the GitHub website.

g. Save and close the **override-values.yaml** file.

h. Install Prometheus Operator with the following command:

```
helm install prometheus kube-prometheus-stack --values override-
values.yaml -n monitoringNamespace
```

where *monitoringNamespace* is the namespace you created for monitoring.

i. Verify the installation with the following command:

```
kubectl get all -n monitoringNamespace
```

Pods and services for the following components should be listed:

- Alert Manager
- Grafana
- Prometheus Operator
- Prometheus
- Node Exporter
- kube-state-metrics

For the list of compatible software versions, see "BRM Cloud Native Deployment Software Compatibility" in *BRM Compatibility Matrix*.

2. Configure BRM REST Services Manager ServiceMonitor, which specifies how to monitor groups of services. Prometheus Operator automatically generates the scrape configuration based on this definition.

a. Ensure that BRM REST Services Manager is running.

b. Create an **rsm-sm.yaml** file with the following content:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  annotations:
    meta.helm.sh/release-name: releaseName
    meta.helm.sh/release-namespace: rsm_namespace
  labels:
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: brm-rest-services-manager
    app.kubernetes.io/version: rsm_version
    chart: brmrestservicesmanager-12.0.0
    heritage: Helm
    release: prometheus
  name: brm-rest-services-manager-monitoring
  namespace: rsm_namespace
spec:
  endpoints:
  - path: /metrics
    port: api-http-prt
  namespaceSelector:
    matchNames:
    - rsm_namespace
  selector:
    matchLabels:
      app.kubernetes.io/name: brm-rest-services-manager
```

where:

- *releaseName* is the name given to the BRM REST Services Manager deployment during Helm installation
- *rsm_namespace* is the namespace where BRM REST Services Manager is deployed
- *rsm_version* is the version of BRM REST Services Manager, for example, **12.0.0.4.0**

c. Save and close the file.

d. Apply the changes with the following command:

```
kubectl apply -f rsm-sm.yaml -n rsm_namespace
```

e. Verify the configuration in the Prometheus user interface. From the **Status** menu, select **Targets**, and confirm that the **/metrics** endpoint appears.

3. Configure Grafana for displaying BRM REST Services Manager metric data. See "Creating Grafana Dashboards for BRM REST Services Manager".

4. Access the health and metrics REST endpoints. See "About REST Endpoints for Monitoring BRM REST Services Manager".

Creating Grafana Dashboards for BRM REST Services Manager

Create a dashboard in Grafana for displaying your BRM REST Services Manager metric data. You can alternatively use the sample dashboard JSON model that is included in the **oc-cn-docker-files-12.0.0.*x*.0.tgz** package.

To use the sample dashboard:

1. Open the **oc-cn-docker-files/samples/monitoring/ocrsm-rsm-dashboard.json** file in a text editor.
2. Search for **instance=\"** and replace the default host and port all occurrences with the host where your instance of Prometheus Operator is running and your prometheus-node-exporter port.

   For example, for the **node_memory_MemFree_bytes** expression, replace *Prometheus_Operator_host* and *Prometheus_Node_Exporter_Port*:

   ```
   {
      "exemplar": true,
      "expr": "node_memory_MemFree_bytes{job=\"node-exporter\",
   instance=\"Prometheus_Operator_host:Prometheus_Node_Exporter_Port\"}",
      "hide": false,
      "interval": "",
      "legendFormat": "Free",
      "refId": "D"
   }
   ```

3. Save and close the file.
4. In Grafana, import the edited **oc-cn-docker-files/samples/monitoring/ocrsm-rsm-dashboard.json** dashboard file. See "Export and Import" in the *Grafana Dashboards* documentation for more information.

Modifying Prometheus and Grafana Alert Rules After Deployment

After deploying Prometheus Operator, you can add alert rules in Prometheus, or make changes in the Grafana user interface.

You have the following options for editing or adding Prometheus alert rules:

- Edit the **override-values.yaml** file and upgrade the Helm release.
- If you added rules in **override-values.yaml** before installing Prometheus Operator, use the following command to edit the rules file:

  ```
  kubectl edit prometheusrule kube-prometheus-stack-0 -n monitoringNamespace
  ```

- If you didn't add any rules in **override-values.yaml**, use the following command to edit the rules file:

  ```
  kubectl edit prometheusrule prometheus-kube-prometheus-alertmanager -n
  monitoringNamespace
  ```

You can also configure alert rules and add or remove email recipients in the Grafana user interface. See "Legacy Grafana Alerts" in the Grafana documentation for more information.

About REST Endpoints for Monitoring BRM REST Services Manager

You can use REST endpoints to monitor metrics and run a health check on BRM REST Services Manager.

Use a browser to send HTTP/HTTPS requests to the endpoints listed in Table 13-1, where *hostname* and *port* are the URL and port for your BRM REST Services Manager server.

Table 13-1 BRM REST Services Manager Monitoring Endpoints

Type	Description	Endpoint
Health	Returns details for both **health/live** and **health/ready** endpoints	**`https://`***`hostname`*`:`*`port`***`/health`**
Liveness	Confirms that the application can run in the environment. Checks disk space, heap memory, and deadlocks.	**`https://`***`hostname`*`:`*`port`***`/health/live`**
Readiness	Confirms that the application is ready to perform work.	**`https://`***`hostname`*`:`*`port`***`/health/ready`**
Metrics	Returns standard Helidon MP monitoring metrics for BRM REST Services Manager.	**`https://`***`hostname`*`:`*`port`***`/metrics`**

Sample Response for the Health Endpoint

The following example shows a response for the health endpoint, which includes both liveness and readiness details:

```
{
    "outcome": "UP",
    "status": "UP",
    "checks": [
        {
            "name": "deadlock",
            "state": "UP",
            "status": "UP"
        },
        {
            "name": "diskSpace",
            "state": "UP",
            "status": "UP",
            "data": {
                "free": "144.85 GB",
                "freeBytes": 155532308480,
                "percentFree": "62.71%",
                "total": "231.00 GB",
                "totalBytes": 248031531008
            }
        },
        {
            "name": "heapMemory",
            "state": "UP",
            "status": "UP",
            "data": {
                "free": "225.08 MB",
                "freeBytes": 236014824,
                "max": "3.48 GB",
                "maxBytes": 3739746304,
                "percentFree": "97.37%",
                "total": "319.00 MB",
                "totalBytes": 334495744
            }
        }
    ]
}
```

Sample Response for the Metrics Endpoint

The response for the metrics endpoint contains the standard Helidon application and vendor metrics. The following example shows some of the metrics in the response:

```
# TYPE base_classloader_loadedClasses_count gauge
# HELP base_classloader_loadedClasses_count Displays the number of classes
that are currently loaded in the Java virtual machine.
base_classloader_loadedClasses_count 9095
# TYPE base_classloader_loadedClasses_total counter
# HELP base_classloader_loadedClasses_total Displays the total number of
classes that have been loaded since the Java virtual machine has started
execution.
base_classloader_loadedClasses_total 9097
...
# TYPE base_memory_usedHeap_bytes gauge
# HELP base_memory_usedHeap_bytes Displays the amount of used heap memory in
bytes.
base_memory_usedHeap_bytes 138109824
# TYPE base_thread_count gauge
# HELP base_thread_count Displays the current number of live threads
including both daemon and nondaemon threads
base_thread_count 20
...
# TYPE vendor_requests_count_total counter
# HELP vendor_requests_count_total Each request (regardless of HTTP method)
will increase this counter
vendor_requests_count_total 4
# TYPE vendor_requests_meter_total counter
# HELP vendor_requests_meter_total Each request will mark the meter to see
overall throughput
vendor_requests_meter_total 4
# TYPE vendor_requests_meter_rate_per_second gauge
vendor_requests_meter_rate_per_second 0.008296727017772145
```

For details about all of the metrics, and more information about Helidon monitoring, see:

- "Helidon MP Metrics Guide" in the Helidon MP documentation
- "MicroProfile Metrics specification" on the GitHub website

14

Tracing BRM REST Services Manager Cloud Native

Learn how to use Zipkin to trace the flow of API calls made to BRM REST Services Manager in your Oracle Communications Billing and Revenue Management (BRM) cloud native system.

Topics in this document:

About BRM REST Services Manager Tracing

You can trace the flow of REST API calls made to BRM REST Services Manager by using Zipkin, which is an open-source tracing system. For more information, see the Zipkin website: https://zipkin.io/.

To set up tracing in BRM REST Services Manager cloud native:

1. Install Zipkin. See the Zipkin Quickstart documentation: https://zipkin.io/pages/quickstart.html.
2. (Optional) Secure communication between BRM REST Services Manager and Zipkin. See "Securing Communication with Zipkin".
3. Enable Zipkin tracing in BRM REST Services Manager cloud native. See "Enabling Tracing in BRM REST Services Manager".

Afterward, you can start tracing the flow of API calls to BRM REST Services Manager by using the Zipkin UI or Zipkin API.

Securing Communication with Zipkin

To use secure communication with Zipkin:

1. Create a client TrustStore that BRM REST Services Manager can use to connect to Zipkin.
2. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the following keys:
   - **ocrsm.rsm.configEnv.trustStoreFileName**: The file name of the BRM REST Services Manager SSL certificate.
   - **ocrsm.rsm.secretVal.trustStorePassword**: The TrustStore password in Base64 format.

Enabling Tracing in BRM REST Services Manager

By default, tracing is disabled in BRM REST Services Manager cloud native. To enable tracing with Zipkin:

1. In your **override-values.yaml** file for **oc-cn-helm-chart**, set the following keys under **ocrsm.rsm.configEnv**:
   - **isTracingEnabled**: Set this to **true**.
   - **zipkinHostName**: Set to this to the hostname of the server on which Zipkin is running.
   - **zipkinPort**: Set this to the port number for Zipkin.
   - **zipkinProtocol**: Set this to **HTTP** or **HTTPS**.
2. Deploy or redeploy the BRM Helm release by running the **helm install** command:

   ```
   helm install BrmReleaseName oc-cn-helm-chart --values
   OverrideValuesFile -n BrmNameSpace
   ```

   where:
   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

Part III

Integrating with BRM Cloud Native

This part describes how to integrate and deploy Oracle Communications Billing and Revenue Management (BRM) cloud native with external systems. It contains the following chapters:

15

Integrating with Your BRM Cloud Native Deployment

Learn how to integrate the Oracle Communications Billing and Revenue Management (BRM) cloud native deployment with external systems, such as Oracle Business Intelligence (BI) Publisher.

Topics in this document:

- Integrating with Thick Clients
- Using a Custom TLS Certificate
- Integrating with JCA Resource Adapter
- Integrating with Kafka Servers
- Integrating with Oracle Analytics Publisher or BI Publisher

Integrating with Thick Clients

You can integrate BRM cloud native with thick clients, such as Customer Center and Pricing Center. To do so:

1. Set these entries in the **override-values.yaml** file for **oc-cn-helm-chart**:
   - **ocbrm.cm.dnsName**: Set this to the primary node name.
   - **ocbrm.isSSLEnabled**: Set this to **1**.
2. Copy the client wallet from the CM service to your thick client's wallet on Windows.

   Note:

   All thick clients installed in standard mode (that is, in non-WebStart mode) can be integrated with the BRM cloud native deployment. This is not relevant for self-care applications.

3. Run the **helm upgrade** command to update the BRM Helm release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
n BrmNameSpace
```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.

- *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

Using a Custom TLS Certificate

You can secure connections between your BRM cloud native deployment and external service providers, such as payment processors and tax calculators, by using Secure Sockets Layer (SSL) certificates. By default, the BRM cloud native deployment uses the TLS certificate provided with the BRM cloud native deployment package.

You can configure the BRM cloud native deployment to use your custom TLS certificate instead. You might do this, for example, to allow client applications outside of the cloud environment to access the BRM cloud native Connection Manager (CM). In this case, the CM is exposed as a Kubernetes NodePort service.

To use a custom TLS certificate, do this:

1. When you generate your custom TLS certificate, ensure that its Subject Alternative Name (SAN) includes these:

   ```
   dns:cm
   dns:HostName
   ```

   where *HostName* is the host name used to connect to the CM from outside the Kubernetes cluster.

   For example, if your CM is running on the **ocbrm.example.com** server and you use the Java **keytool** utility to generate the custom SAN certificate, you'd enter this command:

   ```
   keytool -genkey -keyalg RSA -alias brm -keystore brm_custom.jks -
   validity 365 -keysize 2048 -ext san=dns:cm,dns:ocbrm.example.com
   ```

2. Create an Oracle wallet named **brm_custom_wallet** in the staging area and then copy it to the top level of **oc-cn-helm-chart**:

   ```
   mkdir brm_custom_wallet
   orapki -nologo wallet create -wallet brm_custom_wallet -auto_login -
   pwd Password
   ```

3. Convert the Java KeyStore to the Oracle wallet:

   ```
   orapki wallet jks_to_pkcs12 -wallet brm_custom_wallet -pwd Password
   -keystore brm_custom.jks -jkspwd Password
   ```

4. Verify the contents of the wallet:

   ```
   orapki wallet display -wallet brm_custom_wallet
   ```

5. Move your custom TLS certificate to **oc-cn-helm-chart/brm_custom_wallet**.

   The wallet containing the custom certificate will be mounted at **/oms/wallet/custom**.

6. Update these keys in your **override-values.yaml** file for **oc-cn-helm-chart** and **oc-cn-op-job-helm-chart**:
   - **ocbrm.isSSLEnabled**: Set this to **1**.
   - **ocbrm.cmSSLTermination**: Set this to **true**.
   - **ocbrm.isSSLEnabled**: Set this to **true**.
   - **ocbrm.customSSLWallet**: Set this to **true**.
   - **ocbrm.wallet.client_location**: Set this to **/oms/wallet/custom**.
   - **ocbrm.wallet.server_location**: Set this to **/oms/wallet/custom**.
7. Install BRM cloud native services by entering this command from the **helmcharts** directory.

```
helm install BrmReleaseName oc-cn-helm-chart --namespace BrmNameSpace --
values OverrideValuesFile
```

   where:
   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the path to a YAML file that overrides the default configurations in the chart's **values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

Integrating with JCA Resource Adapter

You can deploy the BRM JCA Resource Adapter in WebLogic Server and use it to run opcodes in the BRM cloud native deployment. When connecting to the BRM cloud native deployment, verify that the following properties are correctly updated in the outbound connection properties.

For more information about JCA Resource Adapter, see *BRM JCA Resource Adapter*.

Table 15-1 lists the JCA Resource Adapter outbound connection properties for connecting to the BRM cloud native deployment.

Note:

To allow the JCA Resource Adapter to communicate with the BRM cloud native deployment, expose the CM service as NodePort. For information, see "Integrating with Thick Clients".

Table 15-1 JCA Resource Adapter Keys

Key	Description
Password	The BRM root password, which can be in plain text or encrypted in OZT.

Table 15-1 (Cont.) JCA Resource Adapter Keys

Key	Description
JavaPcmSSL	Set this to **true**.
SslWalletLocation	Specify the location of the Oracle wallet that contains the BRM client TLS certificate. This can be copied from a BRM installation.
SslCipherSuites	Set this to the desired TLS cipher. For a list of supported TLS ciphers, see "BRM-Supported Cipher Suites" in *BRM System Administrator's Guide*.

Integrating with Kafka Servers

You can integrate your BRM cloud native system with a Kafka server to keep data synchronized between BRM cloud native and your external applications that are connected to the Kafka server. To synchronize account, pricing, and other data, BRM cloud native takes data from internal notification events and constructs a business event that is published to a topic in your Kafka server. Your external applications can then retrieve and process the data from the Kafka topic. For more information, see "About Integrating BRM with an Apache Kafka Server" in *BRM Developer's Guide*.

You integrate BRM cloud native with a Kafka server and configure it to publish data to a Kafka server by using the CM, Kafka DM, and Enterprise Application Integration (EAI) framework.

To integrate BRM cloud native with a Kafka Server:

1. (Optional) Configure the KeyStores required for secure communication between the Kafka DM and Kafka Server.
   a. Create the client certificate, client KeyStore, and client TrustStore. See "Security" in the Apache Kafka documentation.
   b. Verify that the server KeyStore and TrustStore are set up properly by running the following command:

      ```
      openssl s_client -debug -connect DomainName:PortNumber -tls1_2
      ```

      If successful, the certificate is displayed. If the certificate isn't displayed or if there are any other error messages, the KeyStore isn't set up properly.
   c. Move the client's KeyStore files, such as **identity.p12** and **trust.p12**, under the **oc-cn-helm-chart/keystores** directory.
2. Open your **override-values.yaml** file for **oc-cn-helm-chart**.
3. Enable and configure the Kafka DM by editing the following keys:
   - **ocbrm.dm_kafka.is_enabled**: Set this to **true**.
   - **ocbrm.dm_kafka.deployment.kafka_bootstrap_server_list**: Set this to a comma-separated list of addresses for the Kafka brokers in this format: *hostname1*:*port1*, *hostname*2:*port*2. The default is **ece-kafka:9093**.
   - **ocbrm.dm_kafka.deployment.poolSize**: Set this to the number of threads that can run in the JS server to accept requests from the CM. Enter a number from 1 through 2000. The default is **64**.

- **ocbrm.dm_kafka.deployment.topicName**: Set this to the name of the default Kafka topic. The default name is BRM.
- **ocbrm.dm_kafka.deployment.topicFormat**: Set this to the format of the payload that is published to the default Kafka topic: **XML** or **JSON**.
- **ocbrm.dm_kafka.deployment.topicStyle**: Set this to the style of all field names in XML payloads:
  - **ShortName**: The XML field names are in all capitals, such as <POID>, <ACCOUNT_OBJ>, and <SUBSCRIBER_PREFERENCES_INFO>. This is the default.
  - **CamelCase**: The XML field names are in CamelCase, such as <Poid>, <AccountObj>, and <SubscriberPreferencesInfo>.
  - **NewShortName**: The XML field names are in CamelCase and are prefixed with fld, such as <fldPoid>, <fldAccountObj>, and <fldString>.
  - **OC3CNotification**: The input is transformed to match the field and formatting requirements of Oracle Communications Convergent Charging Controller. Use this style if Convergent Charging Controller is your external notification application.
- **ocbrm.dm_kafka.deployment.isSecurityEnabled**: Specifies whether SSL is enabled between the Kafka DM and Kafka Server.
- **ocbrm.dm_kafka.deployment.trustStorePassword**: Specifies the TrustStore password in Base64 format.
- **ocbrm.dm_kafka.deployment.keyStorePassword**: Specifies the KeyStore password in Base64 format.
- **ocbrm.dm_kafka.deployment.keyPassword**: Specifies the key password in Base64 format.
- **ocbrm.dm_kafka.deployment.password**: Specifies the password in Base64 format.

4. Configure the EAI Java Server (JS) to use the Kafka DM event notification file by setting the **ocbrm.eai_js.deployment.eaiConfigFile** key to **payloadconfig_kafka_sync.xml**.
5. Save and close the file.
6. To create additional Kafka topics or configure the Kafka DM to publish different business events to a Kafka topic, edit the **dm-kafka-config** ConfigMap.

   For more information about editing this ConfigMap, see "Mapping Business Events to Kafka Topics" in *BRM Developer's Guide*.
7. Run the **helm upgrade** command to update your BRM Helm release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
n BrmNameSpace
```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

Example: Integrating BRM Cloud Native with a Kafka Server

The following shows sample **override-values.yaml** entries for integrating a BRM cloud native system with a Kafka Server:

```
ocbrm:
    dm_kafka:
        isEnabled: true
        deployment:
            imageName: dm_kafka
            imageTag: $BRM_VERSION
            replicaCount: 1
            kafka_bootstrap_server_list: ece-kafka:9093
            poolSize: 64
            topicName: BRMTopic
            topicFormat: XML
            topicStyle: CamelCase
            isSecurityEnabled: true
            trustStorePassword: TrustStorePassword
            keyStorePassword: KeyStorePassword
            keyPassword: KeyPassword
            password: Password
    eai_js:
        deployment:
            imageName: eai_js
            imageTag: $BRM_VERSION
            eaiConfigFile: payloadconfig_kafka_sync.xml
```

Integrating with Oracle Analytics Publisher or BI Publisher

You can optionally integrate your BRM cloud native deployment with invoicing software such as Oracle Analytics Publisher or Oracle Business Intelligence (BI) Publisher. This integration enables you to generate more detailed and stylized customer invoices that can be viewed in your invoicing software or Billing Care.

To integrate your BRM cloud native deployment with Oracle Analytics Publisher or BI Publisher:

1. If you have not already done so, install Oracle Analytics Publisher or BI Publisher.

   For a list of compatible software versions, see "BRM Software Compatibility" in *BRM Compatibility Matrix*.

2. Install the BRM-BI Publisher invoicing integration package using the OUI installer on your BI Publisher server. This copies invoice layout templates, such as for corporate invoices and consumer invoices, to the BI Publisher server.

   The steps for installing the package on BRM cloud native is similar to installing the package on BRM on-premises. For more information, see "Installing the BRM-BI Publisher Invoicing Integration Package" in *BRM Designing and Generating Invoices*.

3. Configure how to connect your Billing Care and Billing Care REST API cloud native services with Oracle Analytics Publisher or BI Publisher.

   In your **override-values.yaml** file for **oc-cn-op-job-helm-chart**, set these keys:

- **ocbc.bc.configEnv.bipUrl**: The URL for **PublicReportService_v11** from your BI Publisher or Oracle Analytics Publisher instance, which is used by Billing Care to show invoices.
- **ocbc.bc.configEnv.bipUserId**: The name of the user with access to the BI Publisher or Oracle Analytics Publisher instance for viewing invoices from Billing Care.
- **ocbc.bc.secretVal.bipPassword**: The Base64-encoded password for the BI Publisher user.
- **ocbc.bcws.configEnv.bipUrl**: The URL for **PublicReportService_v11** from your BI Publisher or Oracle Analytics Publisher instance, which is used by the Billing Care REST API when accessing PDF invoices.
- **ocbc.bcws.configEnv.bipUserId**: The name of the user with access to the BI Publisher or Oracle Analytics Publisher instance for accessing invoices from the Billing Care REST API.
- **ocbc.bcws.secretVal.bipPassword**: The Base64-encoded password for the BI Publisher user.

4. Do one of the following:
   - Deploy your Billing Care and Billing Care REST API cloud native services. See "Deploying BRM Cloud Native Services" in *BRM Cloud Native Deployment Guide*.
   - Upgrade your Billing Care and Billing Care REST API cloud native services. See "Upgrading Your Billing Care and Billing Care REST API Cloud Native Services" in *BRM Cloud Native Deployment Guide*.
5. Configure how to connect BRM cloud native with Oracle Analytics Publisher or BI Publisher.

   In your **override-values.yaml** file for **oc-cn-helm-chart**, set these keys under **ocbrm.brm_apps.deployment.pin_inv_doc_gen**:
   - **bipServer**: The name of the server on which Oracle Analytics Publisher or BI Publisher is installed.
   - **bipPort**: The port number for Oracle Analytics Publisher or BI Publisher.
   - **bipUsername**: The name of the user with access to the Oracle Analytics Publisher or BI Publisher instance.
   - **bipPassword**: The Base64-encoded password for the BI Publisher user.
   - **schedulerDBServer**: The name of the server on which the Scheduler database is installed.
   - **schedulerDBPort**: The port number for communicating with the Scheduler database.
   - **schedulerDBService**: The service name for the Scheduler database.
   - **schedulerDBUsername**: The user name for the Scheduler database.
   - **schedulerDBServiceCredentials**: The security credentials for connecting to the Scheduler database.
   - **jdbcPoolSize**: The initial number of connections maintained in the pool.
   - **jdbcPoolMaxSize**: The maximum number of connections that can be created.
   - **securityCredentials**: The password for the Oracle wallet.
6. In the BRM Helm chart's **configmap_pin_conf_brm_apps_1.yaml** file, set the following entries:

- **pin_inv_export export_dir**: Set this to **./invoice_dir**.
- **pin_inv_export invoice_fmt**: Set this to **text/xml**.

For example:

```
- pin_inv_export  export_dir  ./invoice_dir
- pin_inv_export  invoice_fmt  text/xml
```

7. Run the **helm upgrade** command to update the Helm release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

   where *BrmReleaseName* is the release name assigned to your existing **oc-cn-helm-chart** installation.

8. In your **bus_params_Invoicing.xml** file, set the following entries:
   - **xsi:schemaLocation**: Set this to **http://www.portal.com/schemas/BusinessConfig/oms/xsd/business_configuration.xsd**.
   - **EnableInvoicingIntegration**: Set this to **enabled** to integrate BRM with your invoicing software.
   - **InvoiceStorageType**: Set this to **1** to store invoices in XML format.

   For example:

```
<BusinessConfiguration
    xmlns="http://www.portal.com/schemas/BusinessConfig"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.portal.com/schemas/
BusinessConfig
/oms/xsd/business_configuration.xsd">
...
<EnableInvoicingIntegration>enabled</EnableInvoicingIntegration>
<InvoiceStorageType>1</InvoiceStorageType>
```

9. In your **bus_params_billing.xml** file, set the following entries:
   - **RerateDuringBilling**: Specify whether delayed events that arrive after the end of the accounting cycle but during the delayed billing period can borrow against the rollover of the current cycle (**enabled**) or not (**disabled**).
   - **EnableCorrectiveInvoices**: Specify whether to enable corrective billing and corrective invoicing (**enabled**) or not (**disabled**).
   - **AllowCorrectivePaidBills**: Specify whether to allow a corrective bill to be generated for a bill that has been fully or partially paid (**enabled**) or not (**disabled**).
   - **RejectPaymentsForPreviousBill**: Specify whether to reject payments when the bill number associated with a payment does not match the last bill (**enabled**) or to accept them (**disabled**).
   - **CorrectiveBillThreshold**: Specify the minimum bill amount that triggers a corrective bill.
   - **GenerateCorrectiveBillNo**: Specify whether corrective invoices use corrective bill numbers (**enabled**) or the original bill numbers (**disabled**).

For example:

```
<RerateDuringBilling>enabled</RerateDuringBilling>
<EnableCorrectiveInvoices>enabled</EnableCorrectiveInvoices>
<AllowCorrectivePaidBills>enabled</AllowCorrectivePaidBills>
<RejectPaymentsForPreviousBill>enabled</RejectPaymentsForPreviousBill>
<CorrectiveBillThreshold>0</CorrectiveBillThreshold>
<GenerateCorrectiveBillNo>enabled</GenerateCorrectiveBillNo>
```

10. In your **events.file** file, specify which events to include in your invoices.

    For more information, see "Including Payment, A/R, and Tax Details in Invoices" in *BRM Designing and Generating Invoices*.

11. In your **pin_business_profile.xml** file, configure your business profiles and validation templates.

    For more information, see "Setting Up Business Profiles and Validation Templates" in *BRM Managing Customers*.

12. In your **pin_invoice_data_map** file, create or modify the data invoice templates.

    For more information, see "Using Data Map Templates" in *BRM Designing and Generating Invoices*.

13. Add the following lines to the **oc-cn-helm-chart/config_scripts/loadme.sh** script:

```
#!/bin/sh

cd /oms/sys/data/config; pin_bus_params -v /oms/load/
bus_params_Invoicing.xml
cd /oms/sys/data/config; pin_bus_params -v /oms/load/
bus_params_billing.xml
cd /oms/sys/data/config; pin_load_invoice_events -reload -brand "0.0.0.1/
account 1 0" -eventfile /oms/load/events.file
cd /oms/sys/data/config; load_pin_business_profile /oms/load/
pin_business_profile.xml
cd /oms/sys/data/config; load_pin_invoice_data_map -dv /oms/load/
pin_invoice_data_map
exit 0;
```

14. Move the following input files to the **oc-cn-helm-chart/config_scripts** directory:
    - **bus_params_invoicing.xml**
    - **bus_params_billing.xml**
    - **events.file**
    - **pin_business_profile.xml**
    - **pin_invoice_data_map**

15. Enable the configurator job.

    In your **override-values.yaml** file for **oc-cn-helm-chart**, set **ocbrm.config_jobs.run_apps** to **true**.

16. Run the **helm upgrade** command to update the Helm release:

    ```
    helm upgrade BrmReleaseName oc-cn-helm-chart --values
    OverrideValuesFile -n BrmNameSpace
    ```

    The configurator job runs the utilities specified in the **loadme.sh** script.

17. Restart the CM because it is required by **pin_bus_params**.

    a. Set these keys in the **override-values.yaml** file:

       - **ocbrm.config_jobs.restart_count**: Increment the existing value by 1
       - **ocbrm.config_jobs.run_apps**: Set this to **false**

    b. Update the Helm release again:

       ```
       helm upgrade BrmReleaseName oc-cn-helm-chart --values
       OverrideValuesFile -n BrmNameSpace
       ```

When configuring Oracle Analytics Publisher or BI Publisher, ensure that the **Create_Xmlp_Invoice_Job.sql** script is run in the schema in which the scheduler database is installed. This script creates the XMLP_INVOICE_JOB table, which should be present in the scheduler database.

Generating Invoices in Oracle Analytics Publisher or BI Publisher

After integration is complete, you can generate your customers' invoices in Oracle Analytics Publisher or BI Publisher by doing the following:

1. Creating **/invoice** objects for your customers by doing one of the following:
   - Running an invoicing job in Business Operations Center. See "Generating Invoices" in *Business Operations Center Help* for more information.
   - Running the **pin_inv_accts** utility through a brm-apps job. See "Running Applications and Utilities through brm-apps Jobs".
2. Generating your customer invoice documents using Oracle Analytics Publisher or BI Publisher templates by running the **pin_inv_doc_gen** utility through a brm-apps job. See "Running Applications and Utilities through brm-apps Jobs".

16 Deploying into Oracle Cloud Infrastructure

Learn how to deploy Oracle Communications Billing and Revenue Management (BRM) 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 computing capabilities (as physical hardware instances) and storage capacity in a flexible overlay virtual network that is securely accessible from your on-premise network. BRM cloud native deployment is tested in Oracle Cloud Infrastructure for the following services both on Virtual Machine and Bare Metal:

- BRM cloud native application and database running on IaaS
- BRM cloud native application managed by Oracle Kubernetes Engine and database on IaaS
- BRM cloud native application managed by Oracle Kubernetes Engine and database on DBaaS

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

> **Note:**
>
> These are the bare minimum tasks for deploying BRM 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 database system on a bare metal or virtual machine instance.

   Select a database version that is compatible with the BRM cloud native software requirements. See "BRM Software Compatibility" in *BRM Compatibility Matrix*.
3. 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 BRM cloud native software requirements.
4. 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.
5. Install and configure **kubectl** on your system to perform operations on your cluster in Oracle Cloud Infrastructure.

6. 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 --token-version 2.0.0
   ```

   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.

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

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

8. 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
   ```

9. Download and configure Helm in your local system.

10. Place the BRM cloud native Helm chart on your system where you have downloaded and configured **kubectl** and Helm. Then, follow the instructions in "Configuring and Deploying BRM Cloud Native" in *BRM Cloud Native Deployment Guide*.

Part IV

Administering PDC Cloud Native Services

This part describes how to administer Oracle Communications Pricing Design Center (PDC) cloud native services. It contains the following chapters:

17

Running PDC Applications

Learn how to run Oracle Communications Pricing Design Center (PDC) applications, such as **ImportExportPricing** and **SyncPDC**, in an Oracle Communications Billing and Revenue Management (BRM) cloud native environment.

Topics in this document:

About Running the PDC Utilities

You can create your pricing and setup components by using these PDC utilities:

- **ImportExportPricing**: Use this utility to import, export, display, delete, or publish the pricing and setup components that are defined in PDC.

  See "Importing and Exporting Pricing and Setup Components" in *PDC Creating Product Offerings* for more information.
- **SyncPDC**: Use this utility to synchronize setup components that are defined in BRM with PDC.

  See "Synchronizing Pricing Setup Components" in *PDC Creating Product Offerings* for more information.

In a BRM cloud native environment, you run these utilities by setting keys in your **override-values.yaml** file for **oc-cn-helm-chart** and then running the **helm upgrade** command.

Importing Pricing and Setup Components with ImportExportPricing

After you deploy PDC, you can create your pricing and setup components by defining them in one or more XML files and then importing them into the PDC database with the **ImportExportPricing** utility.

Note:

To import large XML files, increase the WebLogic transaction timeout settings. For more information, see "Customizing WebLogic for PDC" in *BRM Cloud Native Deployment Guide*.

Importing from a Single XML File

You can import data from a single XML file that contains your pricing and setup components.

To import from a single XML file:

1. Delete the **pdc-import-export-job** Kubernetes job:

   ```
   kubectl delete job pdc-import-export-job
   ```

2. Copy your import XML file to one of these:
   - The HostPath that you specified in **ocpdc.volMnt.pdcIEHostPath**
   - **pdc-ie-pvc**
3. Open your **override-values.yaml** file for **oc-cn-helm-chart**.
4. Under the **ocpdc.configEnv.importExport** section, set these keys:
   - **IE_Operation**: Set this to **import**.
   - **IE_Component**: Set this to one of the following component and object types to import into the PDC database:
     - **config**: Imports pricing setup components, such as tax codes, business profiles, and general ledger IDs.
     - **pricing**: Imports pricing components, such as events, charges, and chargeshares.
     - **metadata**: Imports event, service, account, and profile attribute specifications.
     - **profile**: Imports pricing profile data.
     - **customfields**: Imports custom fields.
     - **all**: Imports all objects and components.
   - **IE_File_OR_Dir_Name**: Set this to the name of your import XML file.
   - **extraCmdLineArgs**: Set this to any extra command-line arguments for **ImportExportPricing**, apart from operation, component, and file name. The value must be surrounded by quotes. For example: "**-n** *ObjectName*".

   For more information about the utility's commands, see "ImportExportPricing" in *PDC Creating Product Offerings*.
5. Save and close the file.
6. Run the **helm upgrade** command to update the release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -n
   BrmNameSpace
   ```

   where:
   - *BrmReleaseName* is the release name assigned to your existing **oc-cn-helm-chart** installation.
   - *OverrideValuesFile* is the file name and path of your **override-values.yaml** file.

- *BrmNameSpace* is the namespace for your existing BRM deployment.

  PDC cloud native runs the **ImportExportPricing** utility at the command line, and the specified pricing and setup components are imported into the PDC database.

Example: Importing Pricing Setup Components from a Single File

This shows sample YAML settings for importing pricing components, such as charge offers, into the PDC database:

```
ocpdc:
   configEnv:
      importExport:
         IE_Operation: import
         IE_Component: pricing
         IE_File_OR_Dir_Name: PDC_ChargeOffers.xml
         extraCmdLineArgs: "-ow -ignoreID"
```

In this case, PDC cloud native runs the following command:

```
./ImportExportPricing -import -pricing PDC_ChargeOffers.xml -ow -ignoreID
```

Importing Multiple XML Files from a Directory

The **ImportExportPricing** utility can import pricing components, setup components, or metadata object from a directory containing multiple import XML files.

Note:

The XML files in the directory must contain only one type of configuration object: only metadata objects, only setup components, or only pricing components.

To import data from multiple XML files in a directory:

1. Delete the **pdc-import-export-job** Kubernetes job:

   ```
   kubectl delete job pdc-import-export-job
   ```

2. Create your import XML files. Ensure the files contain only one type of configuration object: only pricing components, only setup components, or only metadata objects.
3. (Optional) Create an **import_order.cfg** file listing the order in which to import the XML files. For example, you could specify to import **chargeRatePlans.xml** before **chargeOffers.xml**.

   Note:

   - Ensure **import_order.cfg** does not contain empty lines.
   - Without the file, **ImportExportPricing** imports your XML files in a random order.

4. Copy your import XML files and **import_order.cfg** file to one of these:

- The HostPath that you specified in **ocpdc.volMnt.pdcIEHostPath**
- **pdc-ie-pvc**

  The input directory can include one or more subdirectories, but the **import_order.cfg** file must be at the top level of your input directory.

5. Set the ownership and permissions of the input directory, its subdirectories, your import XML files, and **import_order.cfg** file to **chown 1000:0** and **chmod 755**.
6. Open your **override-values.yaml** file for **oc-cn-helm-chart**.
7. Under the **ocpdc.configEnv.importExport** section, set these keys:
   - **IE_Operation**: Set this to **import**.
   - **IE_Component**: Set this to one of the following component and object types to import into the PDC database:
     – **config**: Imports pricing setup components, such as tax codes, business profiles, and general ledger IDs.
     – **pricing**: Imports pricing components, such as events, charges, and chargeshares.
     – **metadata**: Imports event, service, account, and profile attribute specifications.
   - **IE_File_OR_Dir_Name**: Set this to the path in which your import XML files reside.
   - **extraCmdLineArgs**: Set this to any extra command-line arguments for **ImportExportPricing**, apart from operation, component, and file name. The value must be surrounded by quotes. For example: "**-n** *ObjectName*".

   For more information about the utility's commands, see "ImportExportPricing" in *PDC Creating Product Offerings*.
8. Save and close the file.
9. Run the **helm upgrade** command to update the release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -n
   BrmNameSpace
   ```

   where:
   - *BrmReleaseName* is the release name assigned to your existing **oc-cn-helm-chart** installation.
   - *OverrideValuesFile* is the file name and path of your **override-values.yaml** file.
   - *BrmNameSpace* is the name space for your existing BRM deployment.

   PDC cloud native runs the **ImportExportPricing** utility at the command line, and the specified pricing and setup components are imported into the PDC database.

Example: Importing Pricing Setup Components from a Directory

This shows sample YAML settings for importing setup components, such as tax codes, business profiles, and general ledger IDs, into the PDC database:

```
ocpdc:
   configEnv:
      importExport:
```

```
IE_Operation: import
IE_Component: config
IE_File_OR_Dir_Name: MyDirectory
extraCmdLineArgs: "-ow -ignoreID"
```

In this case, PDC cloud native runs the following command:

```
./ImportExportPricing -import -config MyDirectory -ow -ignoreID
```

Exporting Pricing and Setup Components with ImportExportPricing

You can export pricing and setup components from the PDC database into one or more XML files by using the **ImportExportPricing** utility.

Note:

To export large XML files, increase the WebLogic transaction timeout settings. For more information, see "Customizing WebLogic for PDC" in *BRM Cloud Native Deployment Guide*.

To export pricing and setup components from the PDC database:

1. Delete the **pdc-import-export-job** Kubernetes job:

   ```
   kubectl delete job pdc-import-export-job
   ```

2. Open your **override-values.yaml** file for **oc-cn-helm-chart**.
3. Under the **ocpdc.configEnv.importExport** section, set these keys:
   - **IE_Operation**: Set this to **export**.
   - **IE_Component**: Set this to one of the following component and object types to export from the PDC database into an XML file:
     – **config**: Exports pricing setup components, such as tax codes, business profiles, and general ledger IDs.
     – **pricing**: Exports pricing components, such as events, charges, and chargeshares.
     – **metadata**: Exports event, service, account, and profile attribute specifications.
     – **profile**: Exports pricing profile data.
     – **customfields**: Exports custom fields.
     – **brmObject**: Exports all BRM-mastered setup components from PDC.
     – **all**: Exports all objects and components.
   - **extraCmdLineArgs**: Set this to any extra command-line arguments for **ImportExportPricing**, apart from operation, component, and file name. The value must be surrounded by quotes. For example: "**-n** *ObjectName*".

For more information about the utility's commands, see "ImportExportPricing" in *PDC Creating Product Offerings*.

4. Save and close the file.
5. Run the **helm upgrade** command to update your Helm release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

   where:

   - *BrmReleaseName* is the release name assigned to your existing **oc-cn-helm-chart** installation.
   - *OverrideValuesFile* is the file name and path of your **override-values.yaml** file.
   - *BrmNameSpace* is the name space for your existing BRM deployment.

   PDC cloud native runs the **ImportExportPricing** utility, which generates one or more of the following output files to the HostPath specified in the **ocpdc.volMnt.pdcIEHostPath** key:

   - **export_pricing.xml** for the file containing pricing components. If this file already exists in PDC, the utility generates the file name as **export_pricing_*timestamp*.xml**, where *timestamp* is the server's local time in the format *yyyy-mm-dd_hh-mm-ss*.
   - **export_config.xml** for the file containing setup components. If this file already exists in PDC, the utility generates the file name as **export_config_*timestamp*.xml**.
   - **export_profile.xml** for the file containing pricing profile data. If this file already exists in PDC, the utility generates the file name as **export_profile_*timestamp*.xml**.

Example: Exporting Pricing Components

This shows sample YAML settings for exporting pricing components, such as charge offers and discount offers, from the PDC database:

```
ocpdc:
   configEnv:
      importExport:
         IE_Operation: export
         IE_Component: pricing
         extraCmdLineArgs: "-v"
```

In this case, PDC cloud native runs the following command and then exports the pricing data from the PDC database to a file named **export_pricing.xml**.

```
./ImportExportPricing -export -pricing -v
```

Using SyncPDC to Synchronize Setup Components

After you define the following setup components in BRM, you can synchronize the components with PDC on a regular basis by using the **SyncPDC** process:

- Service definitions
- Event definitions
- Account definitions
- General ledger (G/L) IDs
- Provisioning tags
- Tax codes
- Tax suppliers
- Business profiles

The **SyncPDC** process determines which BRM components to synchronize with PDC by using the **ECEEventEnrichmentSpec.xml** file. The default file specifies to synchronize all BRM setup components with PDC, but you can edit it at any time to meet your business needs. The **ECEEventEnrichmentSpec.xml** file is located in the HostPath specified in the **ocpdc.volMnt.pdcBrmHostPath** key.

You specify the schedule and frequency at which to run the **SyncPDC** process when you deploy PDC by using these **override-values.yaml** keys for **oc-cn-helm-chart**:

- **ocpdc.configEnv.syncPDC.SyncPDCStartAt**: Specifies the schedule for running the **SyncPDC** process, such as the time the job was scheduled or at 14:00.
- **ocpdc.configEnv.syncPDC.SyncPDCInterval**: Specifies the frequency at which to run the **SyncPDC** process, such as daily or every 2 hours.

For more information, see "Adding PDC Keys for oc-cn-helm-chart" in *BRM Cloud Native Deployment Guide*.

After PDC is deployed, you can start or stop the synchronization process by creating or deleting the SyncPDC pod. When the pod is created, it automatically begins the BRM-to-PDC synchronization process and runs as a server process in the background, continuously checking for data to synchronize from BRM or the rating system with PDC.

To start or stop the synchronization process:

1. Open your **override-values.yaml** file for **oc-cn-helm-chart**.
2. Under the **ocpdc.configEnv.syncPDC** section, set the **runSyncPDC** key to one of the following:
   - **true** to create the SyncPDC pod and start the synchronization process.
   - **false** to delete the SyncPDC pod and stop the synchronization process.
3. Save and close the file.
4. Run the **helm upgrade** command to update the release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile -
   n BrmNameSpace
   ```

   where:

   - *BrmReleaseName* is the release name assigned to your existing **oc-cn-helm-chart** installation.
   - *OverrideValuesFile* is the file name and path of your **override-values.yaml** file.
   - *BrmNameSpace* is the name space for your existing BRM deployment.

18

Monitoring PDC in a Cloud Native Environment

Learn how to monitor Pricing Design Center (PDC) in your Oracle Communications Billing and Revenue Management (BRM) cloud native environment by using external applications.

Topics in this document:

About Monitoring PDC Cloud Native

You use the following external applications to monitor operations in PDC cloud native:

- **WebLogic Monitoring Exporter**: Use this Oracle web application to scrape runtime information from PDC and then export the metric data in Prometheus format. It exposes different WebLogic Mbeans metrics, such as memory usage and sessions count, that are required for monitoring and maintaining the PDC application deployed on the server.
- **Prometheus**: Use this open-source toolkit to aggregate and store the PDC metric data scraped by the WebLogic Monitoring Exporter.

  You can install a standalone version of Prometheus or Prometheus Operator. If you install Prometheus Operator, PDC adds a ServiceMonitor that declaratively specifies how to monitor groups of services. It automatically generates the Prometheus scrape configuration based on the definition.
- **Grafana**: Use this open-source tool to view on a graphical dashboard all PDC metric data stored in Prometheus.

  To configure Grafana for displaying PDC metric data, see "Getting Started with Grafana" in the Grafana documentation.

Setting Up Monitoring in PDC Cloud Native

Setting up monitoring in PDC cloud native involves these high-level tasks:

1. Deploying Prometheus in one of the following ways:
   - Deploy a standalone version of Prometheus. See "Installation" in the Prometheus documentation.
   - Deploy Prometheus Operator. See "prometheus-operator" on the GitHub website.

   For the list of compatible software versions, see "BRM Cloud Native Deployment Software Compatibility" in *BRM Compatibility Matrix*.
2. Configuring Prometheus to scrape data and send alerts. For more information, see "Configuration" in the Prometheus documentation.
3. Installing Grafana. See "Install Grafana" in the Grafana documentation for information.

For the list of compatible software versions, see "BRM Cloud Native Deployment Software Compatibility" in *BRM Compatibility Matrix*.

4. Enabling monitoring in your PDC cloud native deployment:

   a. In the **override-values.yaml** file for **oc-cn-helm-chart**, set the **ocpdc.configEnv.monitoring.isEnabled** key to **true**.

   b. If you are using Prometheus Operator, also set these keys:

      - **ocpdc.configEnv.monitoring.prometheus.operator.isEnabled**: Set this to **true**.
      - **ocpdc.configEnv.monitoring.prometheus.operator.namespace**: Set this to the namespace of the Prometheus Operator.

   c. Run the **helm upgrade** command to update the Helm release:

      ```
      helm upgrade BrmReleaseName oc-cn-helm-chart --values
      OverrideValuesFile -n BrmNameSpace
      ```

      where:

      - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
      - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
      - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

      WebLogic Monitoring Exporter is installed in your cloud native environment.

5. Edit the **wls-exporter-config.yaml** file to include the PDC metrics that you want to monitor. For the list of metrics that can be used with PDC, see "WebLogic-Based Application Metrics".

6. To create custom metrics for monitoring PDC, do the following:

   a. Create a Python file that defines the custom metrics that you want scraped from PDC.

      For more information, see "Writing Client Libraries" in the *Prometheus Instrumenting* documentation.

   b. Set the **wls-exporter-config.yaml** file's permission to:

      ```
      chown 1000:1000
      chmod 777
      ```

   c. Move your Python file to the HostPath specified in the **ocpdc.volMnt.pdcHostPath** key.

7. Run the **helm upgrade** command to update your BRM Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values
   OverrideValuesFile -n BrmNameSpace
   ```

19 Monitoring PDC REST Services Manager

Learn how to monitor Oracle Communication Pricing Design Center (PDC) REST Services Manager in a cloud native environment using logging, tracing, metrics, and system health data.

Topics in this document:

About PDC REST Services Manager Logs

You can review the PDC REST Services Manager logs to troubleshoot errors and monitor system activity.

PDC REST Services Manager uses the Apache Log4j Java logging utility to log information and errors about the following:

- Start up and shut down activity
- Interaction with other applications at integration points while processing publication events. This includes interactions with PDC, Oracle Identity Cloud Service, and your master product catalog.
- Authorization requests
- Authentication requests
- Zipkin tracing (see "About PDC REST Services Manager Tracing")

You access the logs in the Cloud Native BRM environment using the **kubectl** command in the BRM namespace. See "Accessing the PDC REST Services Manager Logs".

The logs support the standard Java logging levels. By default, the log levels are set to **INFO**. You can change the levels after installation. For example, setting the log levels to **ALL** allows you to log detailed authentication or authorization errors for Helidon security providers. See "Changing the Log Levels".

By default, PDC REST Services Manager routes Java logging to the Log4j log manager. After setting up PDC REST Services Manager, you can change the log manager. See "Changing the Default Log Manager Using Helm".

For general information about Java logging, see *Java Platform, Standard Edition Core Libraries*. For information about Log4j, see:

https://logging.apache.org/log4j/2.x/manual/index.html

Oracle recommends using automated log file rotation for PDC REST Services Manager logs. For information about configuring log file rotation, see My Oracle Support article 2087525.1 at:

https://support.oracle.com/knowledge/Oracle%20Linux%20and%20Virtualization/2087525_1.html

Accessing the PDC REST Services Manager Logs

You access the PDC REST Services Manager logs to monitor and troubleshoot your system.

To access the logs:

1. To get the names of the PDC REST Services Manager pods, enter this command:

   ```
   kubectl -n BRMNameSpace get pods | grep pdcrsm
   ```

   The following is an example of the command's output, with the pod names in bold:

   ```
   pdcrsm-7f48565595-bndp8                    1/1     Running
   0          6h35m
   pdcrsm-7f48565595-hqfwb                    1/1     Running
   0          6h35m
   ```

2. To access the logs, enter this command:

   ```
   kubectl -n BRMNameSpace logs PDCRSMPodName
   ```

   where *PDCRSMPodname* is the name of the PDC REST Services Manager pod you want the log for.

   The following is an example of the logs for updating the **500FreeMinutes** product offering:

   ```
   pdcrsm-6f88869785-vtbw2 pdcrsm 2020-11-13T15:58:06.702Z | INFO |
   9fcdb109-8682-4368-b4d5-b5b720a1af77 | 548aee87-5ef0-4c1a-b8c8-
   d2b8a8c6fb40 | 500FreeMinutes | 4ca071fde65d2a61 | pool-3-
   thread-1      | ctPublishEventServiceImpl | Processing Publish
   Event 548aee87-5ef0-4c1a-b8c8-d2b8a8c6fb40->500FreeMinutes
   pdcrsm-6f88869785-vtbw2 pdcrsm 2020-11-13T15:58:07.303Z | INFO |
   9fcdb109-8682-4368-b4d5-b5b720a1af77 | 548aee87-5ef0-4c1a-b8c8-
   d2b8a8c6fb40 | 500FreeMinutes | 4ca071fde65d2a61 | pool-3-
   thread-1      | ductOfferingServiceLaunch | Retrieving
   ProductOffering for ID 500FreeMinutes
   pdcrsm-6f88869785-vtbw2 pdcrsm 2020-11-13T15:58:09.088Z | INFO |
   9fcdb109-8682-4368-b4d5-b5b720a1af77 | 548aee87-5ef0-4c1a-b8c8-
   d2b8a8c6fb40 | 500FreeMinutes | 4ca071fde65d2a61 | pool-3-
   thread-1      | .c.b.i.d.PdcRmiConnection | Attempting to connect
   to PDC using t3s://pdc-service:8002 ...
   pdcrsm-6f88869785-vtbw2 pdcrsm Handshake failed: TLSv1.3, error =
   No appropriate protocol (protocol is disabled or cipher suites are
   inappropriate)
   pdcrsm-6f88869785-vtbw2 pdcrsm Handshake succeeded: TLSv1.2
   pdcrsm-6f88869785-vtbw2 pdcrsm 2020-11-13T15:58:12.437Z | INFO |
   9fcdb109-8682-4368-b4d5-b5b720a1af77 | 548aee87-5ef0-4c1a-b8c8-
   d2b8a8c6fb40 | 500FreeMinutes | 4ca071fde65d2a61 | pool-3-
   thread-1      | c.b.i.d.PdcDatasourceImpl | Checking if PDC object
   with the name "500FreeMinutes" exists
   ```

```
pdcrsm-6f88869785-vtbw2 pdcrsm 2020-11-13T15:58:12.479Z | INFO |
9fcdb109-8682-4368-b4d5-b5b720a1af77 | 548aee87-5ef0-4c1a-b8c8-
d2b8a8c6fb40 | 500FreeMinutes | 4ca071fde65d2a61 | pool-3-thread-1      |
o.c.b.i.s.PdcServiceImpl  | Updating the PDC object "500FreeMinutes"
pdcrsm-6f88869785-vtbw2 pdcrsm 2020-11-13T15:58:16.134Z | INFO |
9fcdb109-8682-4368-b4d5-b5b720a1af77 | 548aee87-5ef0-4c1a-b8c8-
d2b8a8c6fb40 | 500FreeMinutes | 4ca071fde65d2a61 | pool-3-thread-1      |
o.c.b.i.s.PdcServiceImpl  | PDC object successfully updated for
"500FreeMinutes"
```

Note:

This task shows how to access a single log at a time. To tail logs from multiple pods, Oracle recommends using the Kubernetes Stern tool. See the Stern repository for more information:

https://github.com/stern/stern

Changing the Log Levels

You can change the root log level and the level for PDC REST Services Manager application-specific log entries either by changing Helm values or by editing the PDC REST Services Manager Kubernetes deployment resource.

For a more permanent solution, use Helm, which requires upgrading the Helm deployment. See "Changing the Log Levels Using Helm".

For quicker troubleshooting, use Kubernetes. See "Changing the Log Levels Using Kubernetes".

Changing the Log Levels Using Helm

Change the log levels using Helm for longer term logging.

To change the log levels using Helm:

1. In the **override-values.yaml** file, under the entry for **ocpdcrsm**, edit the values for **rootLoglevel** and **appLogLevel** as needed.

   The following is an example of the **ocpdcrsm** entry, with the default values of **INFO** in bold:

   ```
   ocpdcrsm:
       isEnabled: true
       labels:
           name: "pdcrsm"
           version: "12.0.0.3.4"
       deployment:
           deadlineSeconds: 60
           revisionHistLimit: 10
           imageName:
             pdcrsm: "oracle/pdcrsm"
   ```

```
        # For non-empty tag, ":" MUST be prepended
        imageTag: ":12.0.0.3.4-31848489"
        imagePullPolicy: IfNotPresent
        rootLoglevel: INFO
        appLogLevel: INFO
```

2. Update your Helm release. See "Updating a Helm Release".

Changing the Log Levels Using Kubernetes

Change the log levels using Kubernetes for short term troubleshooting logging.

To change the log levels using Kubernetes:

1. Enter this command:

   ```
   kubectl -n BRMNameSpace set env deployment/pdcrsm
   ROOT_LOG_LEVEL=level PDC_RSM_LOG_LEVEL=level
   ```

   where *level* is the log level you want to set.

   The following is an example of the **ocpdcrsm** entry, with the default values of **INFO** in bold:

   ```
   ocpdcrsm:
       isEnabled: true
       labels:
           name: "pdcrsm"
           version: "12.0.0.3.4"
       deployment:
           deadlineSeconds: 60
           revisionHistLimit: 10
           imageName:
             pdcrsm: "oracle/pdcrsm"
           # For non-empty tag, ":" MUST be prepended
           imageTag: ":12.0.0.3.4-31848489"
           imagePullPolicy: IfNotPresent
           rootLoglevel: INFO
           appLogLevel: INFO
   ```

2. Update your Helm release. See "Updating a Helm Release".

> **Note:**
>
> Next time a Helm update is performed, changes made using Kubernetes will be overwritten. If you want to make the change permanent, update the Helm **override-values.yaml** file as described in "Changing the Log Levels Using Helm".

Changing the Default Log Manager Using Helm

By default, PDC REST Services Manager uses the Log4J Log Manager. You can change this after configuring PDC REST Services Manager.

To change the log manager using Helm:

1. In the **override-values.yaml** file, under the entry for **ocpdcrsm**, edit the value for **-Djava.util.logging.manager=** in **JAVA_OPTS**.
   By default, this is set to **org.apache.logging.log4j.jul.LogManager** when you install PDC REST Services Manager. To use your system default, leave **-Djava.util.logging.manager=** empty, as in the following example.

   ```
   ocpdcrsm:
       isEnabled: true
       labels:
           name: "pdcrsm"
           version: "12.0.0.6"
       deployment:
           deadlineSeconds: 60
           revisionHistLimit: 10
           imageName:
             pdcrsm: "oracle/pdcrsm"
           # For non-empty tag, ":" MUST be prepended
           imageTag: ":12.0.0.6.0-220412.0148.B30-M3"
           imagePullPolicy: IfNotPresent
           rootLoglevel: ALL
           appLogLevel: ALL
           JAVA_OPTS: -Djava.util.logging.manager=
   ```

2. Update your Helm release. See "Updating a Helm Release".

About PDC REST Services Manager Tracing

You can trace the flow of REST API calls made to PDC REST Services Manager by using Zipkin, which is an open-source tracing system. For more information, see the Zipkin website: https://zipkin.io/.

To set up tracing in PDC REST Services Manager cloud native:

1. Install Zipkin. See the Zipkin Quickstart documentation: https://zipkin.io/pages/quickstart.html.
2. Enable Zipkin tracing in PDC REST Services Manager cloud native. See "Enabling Tracing in PDC REST Services Manager".
3. Optionally, add trace tags to help troubleshoot and trace messages and objects through the system. See "Using Trace Tags to Troubleshoot Issues".

Afterward, you can start tracing the flow of REST API calls made to PDC REST Services Manager by using the Zipkin UI or Zipkin API.

Enabling Tracing in PDC REST Services Manager

By default, tracing is disabled in PDC REST Services Manager cloud native, but you can enable it at any time.

To enable tracing with Zipkin:

1. In the **override-values.yaml** file for **oc-cn-helm-chart**, set **ocpdcrsm.configEnv.isTracingEnabled** to **true**.

2. Run the **helm upgrade** command to update your Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values
   OverrideValuesFile -n BrmNameSpace
   ```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the file name and path to your **override-values.yaml** file.
   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

Using Trace Tags to Troubleshoot Issues

Instead of reading through logs to identify and troubleshoot issues, you can use trace tags in PDC REST Services Manager to correlate logs and traces.

PDC REST Services Manager tags events with the following trace tags:

- **publishId**: A general tag for the event. In the example below, this is the first **id**.
- **eventId**: A tag for the event that is specific to PDC REST Services Manager. In the example below, this is the **eventId**.
- **projectId**: A tag for the project in the enterprise product catalog. In the example below, this is the ID under **project**.
- **productOfferId**: A tag for a product offering. In the example below, this is the ID under each entry in the **projectItems** array.
- **productSpecificationId**: A tag for product specifications. This does not appear in the example below, but would appear in log messages. You use the productOfferId tag to filter logs and locate related productSpecificationId tags as needed.

The following shows an example event for publishing updates to two product offerings from an enterprise product catalog to PDC. To illustrate an error scenario, a URL in the payload for the **testInit4Offer** product offering has become corrupt. The IDs corresponding to trace tags are shown in bold.

```
{
  "id": "d64066bd-2954-4f43-b8f2-69603c88c683",
  "eventId": "ea09ae5a-8098-4fb2-b634-ee8048b9cc1d",
  "eventTime": "2030-11-18T09:31:50.001Z",
  "eventType": "projectPublishEvent",
  "correlationId": "UC4Fcfc6a70f-60f5-456c-93d5-d8e038215201",
  "domain": "productCatalogManagement",
  "timeOcurred": "2030-11-18T09:31:50.001Z",
  "event": {
    "project": {
      "id": "demopackage11",
      "lifecycleStatus": "IN_DESIGN",
      "name": "Project01",
      "acknowledgementUrl": "http://host:port/mobile/custom/
PublishingAPI",
```

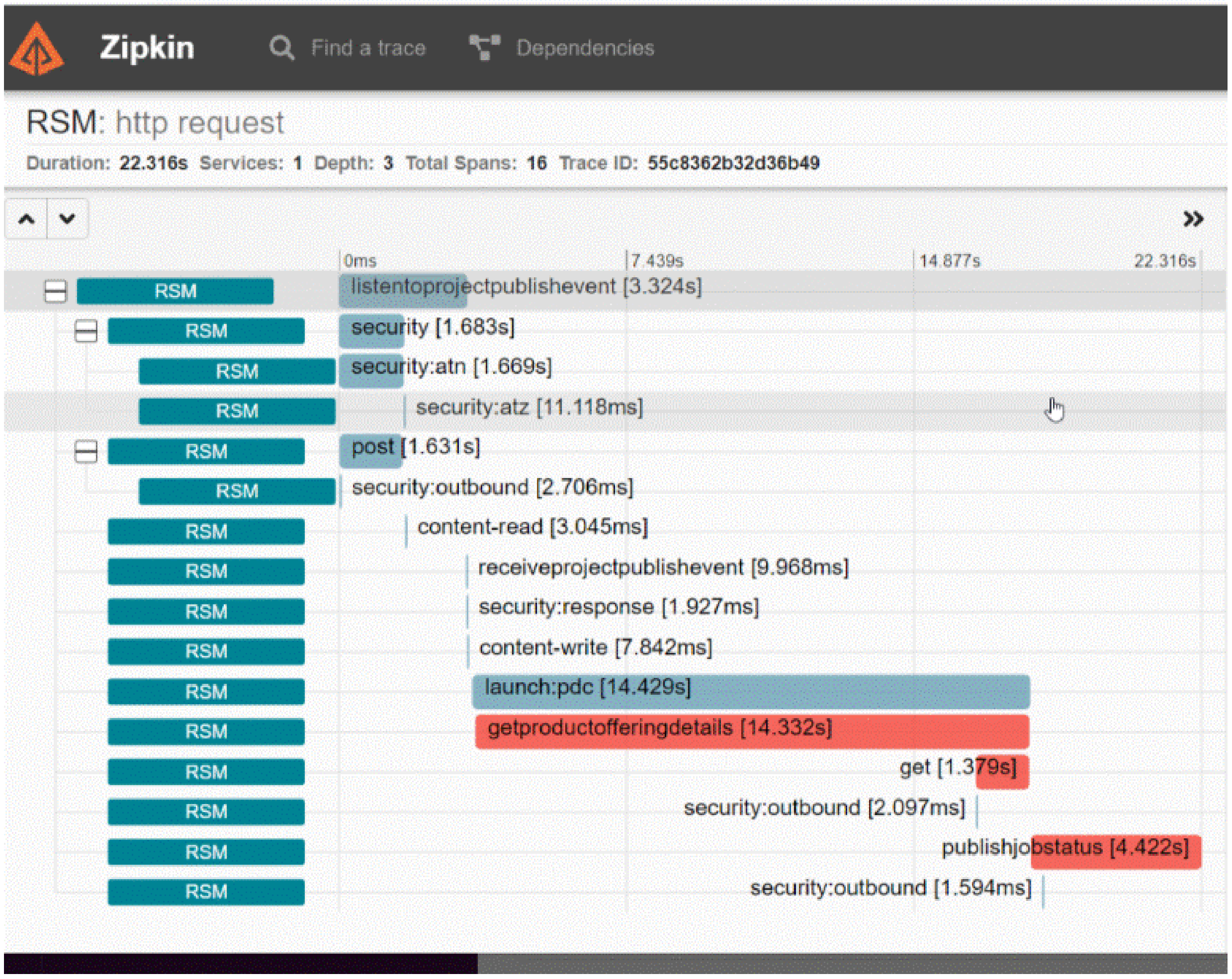
```
      "projectItems": [
        {
          "id": "55c8362b32d36b49",
          "href": "http://host:port/mobile/custom/catalogManagement/
productOffering/testSuccess",
          "name": "testSuccess",
          "version": "1.0",
          "@referredType": "ProductOfferingOracle"
        },
        {
          "id": "55c8362b32d36b55",
          "href": "http://host:port/mobile/custom/CORRUPTDATA/
productOffering/testInit4Offer",
          "name": "100Minutes",
          "version": "1.0",
          "@referredType": "ProductOfferingOracle"
        }
      ]
    }
  }
}
```

Trace Tags in Tracer Tools

After submitting the event, you can follow its progress and look for the trace tags in a tracer tool like Zipkin.

Figure 19-1 shows excerpts from a tracer. You can immediately see that the error occurred in the GET request of the getProductOfferingDetails operation. You can expand the trace spans to get the IDs for the event and the object in question, then search in the logs for those tags, as well as the span and trace IDs, to troubleshoot the issue.

Figure 19-1 Sample Tracer Excerpts



RSM: http request

Duration: 22.316s Services: 1 Depth: 3 Total Spans: 16 Trace ID: 55c8362b32d36b49

```
0ms          7.439s          14.877s          22.316s
RSM  listentoprojectpublishevent [3.324s]
RSM  security [1.683s]
RSM  security:atn [1.669s]
RSM  security:atz [11.118ms]
RSM  post [1.631s]
RSM  security:outbound [2.706ms]
RSM  content-read [3.045ms]
RSM  receiveprojectpublishevent [9.968ms]
RSM  security:response [1.927ms]
RSM  content-write [7.842ms]
RSM  launch:pdc [14.429s]
RSM  getproductofferingdetails [14.332s]
RSM  get [1.379s]
RSM  security:outbound [2.097ms]
RSM  publishjobstatus [4.422s]
RSM  security:outbound [1.594ms]
```

The following is the same data for listenToProjectPublishEvent and getProductOfferingDetails in JSON format, with the relevant IDs in the tags arrays in bold:

```
{
  "traceID": "55c8362b32d36b49",
  "spanID": "bad2ef5f3ff26084",
  "flags": 1,
  "operationName": "listenToProjectPublishEvent",
  "references": [
    {
      "refType": "CHILD_OF",
      "traceID": "f2f902949ee8e661",
      "spanID": "8ce5e8f8cda38d3b"
    }
  ],
  "startTime": 1605709909244000,
  "duration": 18160,
  "tags": [
    {
      "key": "eventId",
      "type": "string",
      "value": "ea09ae5a-8098-4fb2-b634-ee8048b9cc1d"
    },
    {
      "key": "http.status_code",
      "type": "int64",
      "value": 201
    },
    {
      "key": "component",
      "type": "string",
      "value": "jaxrs"
    },
    {
      "key": "span.kind",
      "type": "string",
      "value": "server"
    },
    {
      "key": "http.url",
      "type": "string",
      "value": "http://host:port/productCatalogManagement/v1/
projectPublishEvent"
    },
    {
      "key": "http.method",
      "type": "string",
      "value": "POST"
    },
    {
      "key": "projectId",
      "type": "string",
      "value": "demopackage11"
    },
    {
      "key": "publishId",
      "type": "string",
      "value": "d64066bd-2954-4f43-b8f2-69603c88c683"
    },
    {
      "key": "internal.span.format",
      "type": "string",
      "value": "Zipkin"
    }
  ],
  "logs": [],
  "processID": "p1",
  "warnings": null
},
...
{
  "traceID": "f2f902949ee8e661",
  "spans": [
    {
      "traceID": "f2f902949ee8e661",
      "spanID": "03031b1c18e679f2",
      "flags": 1,
      "operationName": "getProductOfferingDetails",
      "references": [
        {
          "refType": "CHILD_OF",
          "traceID": "f2f902949ee8e661",
          "spanID": "528a32ac350706e2"
        }
      ],
      "startTime": 1605709909256000,
      "duration": 688729,
      "tags": [
        {
          "key": "productOfferId",
          "type": "string",
          "value": "testInit4Offer"
        },
        {
          "key": "internal.span.format",
          "type": "string",
          "value": "Zipkin"
        }
      ],
      "logs": [],
      "processID": "p1",
      "warnings": null
    },
    {
      "traceID": "f2f902949ee8e661",
      "spanID": "303707dcd9c9d1ef",
      "flags": 1,
      "operationName": "getProductOfferingDetails",
      "references": [
        {
          "refType": "CHILD_OF",
          "traceID": "f2f902949ee8e661",
          "spanID": "d1d2c068248a5542"
        }
      ],
      "startTime": 1605709909277000,
      "duration": 529234,
      "tags": [
        {
          "key": "error",
          "type": "bool",
          "value": true
        },
        {
          "key": "productOfferId",
          "type": "string",
          "value": "testInit4Offer"
        },
        {
          "key": "internal.span.format",
          "type": "string",
          "value": "Zipkin"
        }
      ],
      "logs": [
        {
          "timestamp": 1605709909807000,
          "fields": [
            {
              "key": "event",
              "type": "string",
              "value": "error"
            },
            {
              "key": "error.object",
              "type": "string",
              "value":
"oracle.communications.brm.integration.exceptions.EccServiceException"
            }
          ]
        }
      ],
      "processID": "p1",
      "warnings": null
    }
  ]
}
```

Trace Tags in Logs

After finding the trace tags in the tracer tool, you can search the logs for them. You can do simple searches in the raw log data, or you can search and filter by the tags using a logging tool, such as Grafana Loki.

The trace tags appear in the following format in PDC REST Service Manager logs:

```
yyyy-MM-dd'T'HH:mm:ss.SSSXXX, UTC | level | eventId | projectId |
productOfferId | traceId | thread | logging service | message
```

The following shows the success message in the logs for updating the **testInit4Offer** product, with the relevant trace tags from the event in bold:

```
2030-10-11T11:34:36,231+05:30 | INFO | ea09ae5a-8098-4fb2-b634-ee8048b9cc1d
| demopackage11 | testInit4Offer | 55c8362b32d36b49 | pool-4-thread-1 |
ctPublishEventServiceImpl | Processing Publish Event ea09ae5a-8098-4fb2-b634-
ee8048b9cc1d->testInit4Offer
```

For the **testInit4Offer** product, the following error log appears:

```
2020-11-18T14:31:49.814Z | ERROR | ea09ae5a-8098-4fb2-b634-ee8048b9cc1d |
demopackage11 | testInit4Offer | f2f902949ee8e661 | pool-3-thread-4
```

```
| .s.LaunchPdcItemPublisher | Error calling API service 'Product
Offering Service' for 'testInit4Offer'. Status Code: 404 Error: '
```

Based on this message and what you saw in the tracer, you would know that PDC REST Services Manager wasn't able to make the call to the enterprise product catalog to request information about the **testInit4Offer** product offering. Expanding and inspecting the GET span in the tracer would reveal the corrupt URL. You could then review the message that came from your enterprise product catalog to confirm, and make appropriate changes to resolve the issue.

About PDC REST Services Manager Metrics

You can monitor the PDC REST Services Manager metrics by using the Metrics REST endpoint. The metrics count successful and failed messages passing through the PDC REST Services Manager integration points.

Use a monitoring tool that scrapes metrics data, such as Prometheus, to monitor the metrics available from the PDC REST Services Manager Metrics endpoint. You can get the metrics in plain text format, which is compatible with Prometheus, or JSON format. See "Checking Access to PDC REST Services Manager Metrics" for information about accessing the metrics endpoint and requesting different formats. For more information about Prometheus, see: https://prometheus.io/.

Table 19-1 shows the available PDC REST Services Manager metrics.

Table 19-1 PDC REST Services Manager Metrics

Integration Point	Metric	Description
PDC interface	pdc-create-object-success-total	The number of Create events that returned a success from PDC.
PDC interface	pdc-create-object-error-total	The number of Create events that returned an error from PDC.
PDC interface	pdc-update-object-success-total	The number of update events that returned a success from PDC.
PDC interface	pdc-update-object-error-total	The number of update events that returned an error from PDC.
Product Offer Price Project life cycle event listener	notification-listener-change-success-total	The number of well-formed publish events received by PDC REST Services Manager.
Product Offer Price Project life cycle event listener	notification-listener-change-error-total	The number of publish events accepted by PDC REST Services Manager that could not be processed due to invalid or incomplete event payloads.
Product Offering interface	product-offering-get-success-total	The number of Product Offering GET API requests that returned a success from the master product catalog.
Product Offering interface	product-offering-get-error-total	The number of Product Offering GET API requests that returned an error from the master product catalog.

Table 19-1 (Cont.) PDC REST Services Manager Metrics

Integration Point	Metric	Description
Product Specification interface	product-specification-get-success-total	The number of Product Specification GET API requests that returned a success from the master product catalog.
Product Specification interface	product-specification-get-error-total	The number of Product Specification GET API requests that returned an error from the master product catalog.
Publish Notification interface	publish-job-status-success-total	The number of Publish Notification Acknowledgments that returned a success from the master product catalog.
Publish Notification interface	publish-job-status-fail-total	The number of Publish Notification POST Acknowledgments that returned an error from the master product catalog.
Publish Product Offering service	publish-product-offering-success-total	The number of successful Product Offering Publish actions.
Publish Product Offering service	publish-product-offering-fail-total	The number of Failed Product Offering Publish actions.

You can also use Helidon framework metrics. See the Helidon documentation for more information: https://helidon.io/docs/v1/#/metrics/01_metrics.

Checking Access to PDC REST Services Manager Metrics

You can access the PDC REST Services Manager metrics from any tool that can access REST API endpoints using an OAuth token generated by Oracle Identity Cloud Service for PDC REST Services Manager. You can check whether you have access by using cURL commands.

To check whether you have access to the PDC REST Services Manager metrics:

1. In the command line on the system where cURL and your scraping tool are installed, export your OAuth access token with the following command:

   ```
   export TOKEN=OAuth_metrics_token
   ```

   where *OAuth_metrics_token* is the client secret you stored for the Metrics scope in "Configuring OAuth Authentication in PDC REST Services Manager" in *BRM Cloud Native Deployment Guide*.

2. Enter one of the following commands:
   - To get the metrics in plain text format:

     ```
     curl --insecure -H "Authorization: Bearer $TOKEN" https://
     hostname:port/metrics
     ```

     where:
     – *hostname* is the URL for the PDC REST Services Manager server.
     – *port* is the TLS port for the PDC REST Services Manager server.

- To get the metrics in JSON format:

  ```
  curl --insecure -H "Authorization: Bearer $TOKEN" -H "Accept: 
  application/json" https://hostname:port/metrics
  ```

About Monitoring PDC REST Services Manager System Health

You can assess the health of the PDC REST Services Manager system by monitoring the pod status and using the Health REST endpoint.

See:

- Verifying the PDC REST Services Manager Pod Status
- Using the PDC REST Services Manager Health Endpoint

Verifying the PDC REST Services Manager Pod Status

To verify the pod status, run this command:

```
kubectl -n BRMNameSpace get pods --selector=app.kubernetes.io/
name=pdcrsm
```

The following is an example of the command output:

```
NAME                             READY     STATUS      RESTARTS
AGE
pdcrsm-b9d7bb7d6-j2xsl7          1/1       Running     0
105m
pdcrsm-b9d7bb7d6-lfxcl           1/1       Running     0
105m
```

Note:

Kubernetes provides automatic health monitoring and will attempt to restart applications when they fail.

Using the PDC REST Services Manager Health Endpoint

You can monitor overall system health by submitting a GET request to the following endpoint:

```
https://hostname:port/health
```

where:

- *hostname* is the URL for the PDC REST Services Manager server
- *port* is the TLS port for the PDC REST Services Manager server

The response contains information about:

- Deadlocked threads
- Disk space used
- Memory heap used

The following is an example of the response:

```
{
    "outcome": "UP",
    "status": "UP",
    "checks": [
        {
            "name": "deadlock",
            "state": "UP",
            "status": "UP"
        },
        {
            "name": "diskSpace",
            "state": "UP",
            "status": "UP",
            "data": {
                "free": "101.80 GB",
                "freeBytes": 109306679296,
                "percentFree": "69.01%",
                "total": "147.52 GB",
                "totalBytes": 158399414272
            }
        },
        {
            "name": "heapMemory",
            "state": "UP",
            "status": "UP",
            "data": {
                "free": "399.05 MB",
                "freeBytes": 418431544,
                "max": "6.89 GB",
                "maxBytes": 7393378304,
                "percentFree": "99.41%",
                "total": "440.88 MB",
                "totalBytes": 462290944
            }
        }
    ]
}
```

20
Rotating PDC Log Files

Learn how to rotate log files for your Oracle Communications Pricing Design Center application to prevent them from growing too large.

Topics in this document:

About Rotating PDC Log Files

During log file rotation, PDC cloud native writes to a log file until it reaches a maximum size. It then closes the log file and starts writing to a new log file. Rotation prevents your log files from growing too large, making them slow to open and search.

You can set these log file rotation properties for PDC applications:

- **Log level**: Sets the logging level, which can be SEVERE, WARNING, INFO, CONFIG, FINE, FINER, or FINEST.
- **Log limit**: Sets the maximum file size, in bytes, of the log files. After the log file meets the maximum, PDC closes the log file and creates a new log file.
- **Log file count**: Specifies the maximum number of log files to retain for the application.
- **Persist log setting**: Specifies whether to persist log files in the database after they are closed. Possible values are:
  - **enabled** or **all**: Persists all log files.
  - **disabled**: Does not persist log files.
  - **failed**: Persists failed log files only.

Note:

Only Real-Time Rating Engine (RRE) and Batch Rating Engine (BRE) transaction log files and **ImportExportPricing** log files can be persisted.

Table 20-1 lists the default log file rotation settings for the PDC applications.

Table 20-1 PDC Application Log Files

PDC Application Name or Log File	Default Log Level	Default Log Limit	Default Log File Count	Default Persist Log Setting
Pricing Server Log	WARNING	500000	50	N/A
Pricing Server Trace Log	WARNING	500000	50	N/A
ImportExportPricing utility	WARNING	1048576 (1 MB)	100	failed
SyncPDC utility	WARNING	20000	10	N/A

Table 20-1 (Cont.) PDC Application Log Files

PDC Application Name or Log File	Default Log Level	Default Log Limit	Default Log File Count	Default Persist Log Setting
RRE/BRE Transformation Master Log	WARNING	50000	50	N/A
RRE/BRE Transaction Logs	WARNING	N/A	N/A	failed

The following sections show how to configure log file rotation for PDC cloud native applications.

Configuring Pricing Server Log File Rotation

This shows sample **override-values.yaml** keys for **oc-cn-op-job-helm-chart**. It configures log file rotation for the Pricing Server logs and tracer logs:

```
ocpdc:
   configEnv:
      pdcAppLogLevel: WARNING
      pdcAppLogFileSize: 500000
      pdcAppLogFileCount: 50
```

Configuring ImportExportPricing Log File Rotation

This shows sample **override-values.yaml** keys for **oc-cn-helm-chart**. It configures log file rotation for the **ImportExportPricing** utility:

```
ocpdc:
   configEnv:
      importExport:
         logLevel: SEVERE
         logSize: 50000
         logCount: 100
         persistIELogs: true
```

Configuring SyncPDC Log File Rotation

This shows sample **override-values.yaml** keys for **oc-cn-helm-chart**. It configures log file rotation for the **SyncPDC** utility:

```
ocpdc:
   configEnv:
      syncPDC:
         logLevel: INFO
         logFileSize: 50000
         logFileCount: 100
```

Configuring RRE/BRE Log File Rotation

This shows sample **override-values.yaml** keys for **oc-cn-helm-chart**. It configures log file rotation for the RRE/BRE transformation master log and transaction logs:

```
ocpdc:
   configEnv:
      transformation:
         logLevel: INFO
         logFileSize: 50000
```

```
logFileCount: 100
persistTransactionLogs: failed
```

Part V
Administering ECE Cloud Native Services

This part describes how to perform administration tasks on Oracle Communications Elastic Charging Engine (ECE) cloud native services. It contains the following chapters:

21 Administering ECE Cloud Native Services

Learn how to perform common system administration tasks in Oracle Communications Billing and Revenue Management (BRM) cloud native on your Elastic Charging Engine (ECE) cloud native services.

Topics in this document:

Running SDK Jobs

You can run sample scripts for ECE cloud native services by running an SDK job.

To run SDK jobs:

1. In the **override-values.yaml** file for the ECE Helm chart, set the **job.sdk.runjob** key to **true**.
2. The SDK directory containing the SDK sample scripts, configuration files, source code, and so on is exposed in the PVC defined under the **pvc.sdk** section of the **values.yaml** file.
3. Run the **helm install** command to deploy the ECE Helm chart:

   ```
   helm install EceReleaseName oc-cn-ece-helm-chart --namespace BrmNameSpace
   --values OverrideValuesFile
   ```

   The command creates a default SDK job that prints the following since you have not run the SDK job with any valid parameters:

   ```
   "Run the SDK job with script name and parameters. Usage - cd usage; sh
   <scriptname> build; sh <scriptname> run <parameters>"
   ```

   The SDK job then goes into a **Completed** state.
4. Check the logs printed by the job by running this command:

   ```
   kubectl logs sdkJobName
   ```

5. After deployment completes and all of the pods are in a healthy state, you can run any sample SDK script by doing one of these:

   - Running the **helm upgrade** command in the following format:

     ```
     'helm upgrade eceDeploymentName helmChartFolder --set
     job.sdk.name=SDKJobName --set job.sdk.command="cd <folder-name>;
     sh <script-name> build; sh <scriptname> run <parameters>"'
     ```

     where:

     – *eceDeploymentName* is the deployment name given during Helm installation. The deployment name can be retrieved by running the **helm ls** command.

     – *helmChartFolder* is the location where the ECE Helm chart is located.

     – *SDKJobName* is the user-defined name for this instance of the SDK job.

     – **job.sdk.command** is set to the command to run as part of the job. The SDK job runs from the **ocecesdk/bin** directory, so you only need to provide the script file location from the reference point of the **ocecesdk/bin** directory.

       For example:

       ```
       helm upgrade ece . --set job.sdk.name=samplegprssessionjob --
       set job.sdk.command="cd usage; sh sample_gprs_session.sh
       build; sh sample_gprs_session.sh run 773-20190923 INITIATE
       60 1024 1024 TelcoGprs EventDelayedSessionTelcoGprs 1.0
       2020-02-10T00:01:00 1024 1024 sessionId CUMULATIVE 1"
       ```

       This command will not affect any other running pod in the name space, except it creates the job specified in **job.sdk.name**. The job runs the command specified in **job.sdk.command**.

   - Setting the SDK job and SDK command in your **override-values.yaml** file:

     ```
     sdk:
        name: "SDKJobName"
        command: "cd <folder-name>; sh <script-name> build; sh
     <scriptname> run <parameters>"
        runjob: "true"
     ```

     Then, running the **helm upgrade** command:

     ```
     helm upgrade eceDeploymentName helmChartFolder
     ```

6. After the job completes, it goes into a **Completed** state. You can check the logs by running this command:

   ```
   kubectl logs sdkJobName
   ```

   sdkJobName will be available from the **kubectl get po** command. The job name will be in the format: *JobName-IDfromKubernetes*.

7. To view the logs created by the SDK script, check the **sdk** logs folder in the PVC.

Error Handling for SDK Jobs

Any error that occurs while running an SDK job will result in the job going into an Error state. For example, an SDK job will go into an Error state when the job is running when ECE has not reached UsageProcessing state or when the SDK command includes invalid parameter values.

You can check the reason why an error occurred by doing the following:

1. Running this command, which prints the output of the script:

   ```
   kubectl logs sdkJobName
   ```

2. Checking the log file created under the SDK PVC location.

After correcting the error, run the **helm upgrade** command with a new job name. See "Running SDK Jobs".

If you don't provide SDK commands while running the **helm upgrade** command, it prints the following:

```
Run the SDK job with script name and parameters.
Usage - cd usage; sh <scriptname> build; sh <scriptname> run <parameters>
```

If you don't provide a job name, it uses the default job name of **sdk**. However, since Kubernetes doesn't allow a completed job to be rerun, you must delete any previous job named **sdk** before running the **helm upgrade** command again.

Using a Custom TLS Certificate for Secure Connections

To configure ECE to use a custom TLS certificate for communicating with external service providers, set these keys in the **override-values.yaml** file for **oc-cn-ece-helm-chart**:

- **charging.customSSLWallet**: Set this to **true**.
- **charging.secretCustomWallet.name**: Set this to the Secret name.
- **charging.emGatewayConfigurations.emGatewayConfigurationList.emGateway1Config.wallet**: Set this to **/home/charging/wallet/custom/cwallet.sso**.
- **charging.emGatewayConfigurations.emGatewayConfigurationList.emGateway2Config.wallet**: Set this to the custom wallet path.
- **charging.brmWalletServerLocation**: Set this to the custom wallet path.
- **charging.brmWalletClientLocation**: Set this to the custom wallet path.
- **charging.brmWalletLocation**: Set this to the custom wallet path.
- **charging.radiusGatewayConfigurations.wallet**: Set this to the custom wallet path.
- **charging.connectionConfigurations.BRMConnectionConfiguration.brmwallet**: Set this to the custom wallet path.

Note:

If the custom wallet is deployed after ECE is installed, perform a Helm upgrade. You can update the wallet location configured for ECE pods such as radiusgateway, emgateway, and brmgateway by using JMX.

Configuring Subscriber-Based Tracing for ECE Services

You can selectively trace your subscribers' sessions based on one or more subscriber IDs. You can also specify to trace and log selective functions, such as alterations (discounts), charges, and distributions (charge sharing), for each subscriber.

ECE generates log files for the listed subscribers for each session. If a subscriber has multiple sessions, separate log files are generated for each session. The trace file names are unique and are in the format *nodeName***.***subscriberID***.***sessionID***.log**. For example, **ecs1.SUBSCRIBER1.SESSION1.log**.

Note:

ECE does not archive or remove the log files that are generated. Remove or archive the log files periodically to avoid running out of disk space.

To configure subscriber-based tracing for your ECE services:

1. To enable subscriber-based tracing, do the following:
   a. Open your **override-values.yaml** file for **oc-cn-ece-helm-chart**.
   b. Set the following keys under the **subscriberTrace** section:
      - **logMaxSubscribers**: Specify the maximum number of subscribers for whom you want to enable tracing. The default value is **100**.
      - **logMaxSubscriberSessions**: Specify the maximum number of sessions for which the logs need to be generated per subscriber. The default value is **24**.
      - **logExpiryWaitTime**: Specify how long to wait, in seconds, before the logging session expires. The default value is **1**.
      - **logCleanupInterval**: Specify the interval time, in seconds, for log cleanup. The default value is **2**.
      - **logLevel**: Specify the log level you want to use for generating logs, such as DEBUG or ERROR. The default value is **DEBUG**.
      - **subscriberList**: Specify a list or range of subscriber IDs to trace. For example, you could enter **subscriberId1-subscriberId10** to specify the range of subscribers from 1 through 10.
   c. Save and close your **override-values.yaml** file.
2. To enable subscriber-based tracing for the alterations, charges, and distributions functions, do the following:

a. Open your **charging-settings.yaml** ConfigMap.
b. Go to the **subscriber-trace.xml** section of the file.
c. Update the **<componentLoggerList>** element to include the list of functions to trace and log.

   For example, to enable subscriber-based tracing and logging for the alteration function, you would add the following lines:

```
<componentLoggerList config-
class="java.util.ArrayList">
   <componentLogger
      loggerName="ALL"
      loggerLevel="ERROR"
      config-
class="oracle.communication.brm.charging.subscribertrace.configuration
.internal.ComponentLoggerImpl"/>
   <componentLogger
      loggerName="oracle.communication.brm.charging.rating.alteration"
      loggerLevel="DEBUG"
      config-
class="oracle.communication.brm.charging.subscribertrace.configuration
.internal.ComponentLoggerImpl"/>
</componentLoggerList>
```

d. Save and close your **override-values.yaml** file.

3. Run the **helm upgrade** command to update your ECE Helm chart:

```
helm upgrade EceReleaseName oc-cn-ece-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

   where:

   - *EceReleaseName* is the release name for **oc-cn-ece-helm-chart** and is used to track this installation instance.
   - *OverrideValuesFile* is the name and location of your **override-values.yaml** file for **oc-cn-ece-helm-chart**.
   - *BrmNameSpace* is the name space in which the BRM Kubernetes objects reside.

4. In your **override-values.yaml** file for **oc-cn-ece-helm-chart**, set the **charging.jmxport** key to **31022**.

5. Label the ecs1-0 pod so that JMX can connect to it:

```
kubectl -n namespace label pod ecs1-0 ece-jmx=ece-jmx-external
```

6. Update the **/etc/hosts** file on the remote machine with the worker node of ecs1-0:

```
IP_OF_WORKER_NODE ecs1-0.ece-server.namespace.svc.cluster.local
```

7. Connect to JConsole by entering this command:

```
jconsole ecs1-0.ece-server.namespace.svc.cluster.local:31022
```

JConsole starts.

8. Do the following in JConsole:
   a. In the editor's MBean hierarchy, expand the **ECE Logging** node.
   b. Expand **Configuration**.
   c. Expand **Operations**.
   d. Select **updateSubscriberTraceConfiguration**.
   e. Click the **updateSubscriberTraceConfiguration** button.
   f. In the editor's MBean hierarchy, expand the **ECE Subscriber Tracing** node.
   g. Expand **SubscriberTraceManager**.
   h. Expand **Attributes**.
9. Verify that the values that you specified in step 3 appears.

> **Note:**
>
> The attributes displayed here are *read-only*. You can update these attributes by editing the *ECE_home***/config/subscriber-trace.xml** file.

To disable subscriber-based tracing, remove the list of subscribers from the **subscriberTrace.subscriberList** key in your **override-values.yaml** file and then run the **helm upgrade** command.

Creating a JMX Connection to ECE Using JConsole

To create a JMX connection to ECE cloud native using JConsole:

1. In your **override-values.yaml** file, set the **charging.jmxport** key to the JMX port.

> **Note:**
>
> The global **charging.jmxport** key sets the default JMX port for all ECE pods. However, you can override the JMX port for an individual pod by specifying a different port in the pod's **jmxport** key.
>
> If an individual pod's JMX port is exposed for JMX connection, create custom services similar to ece-jmx-service-external for each ECE deployment type and set the **jmxservice.port** key to the same value as the pod's **jmxport** key.

2. Label the pod as the ece-jmx-service-external service endpoint by running this command:

```
kubectl label po ecs1-0 ece-jmx=ece-jmx-external
```

3. Retrieve the worker node's IP address by running this command:

```
kubectl get pod ecs1-0 -o wide
```

4. Update the **/etc/hosts** file in the remote machine with the worker node's IP by running this command:

```
ipAddress ecs1-0.ece-server.namespace.svc.cluster.local
```

> **Note:**
>
> You don't need to update the **/etc/hosts** file if JConsole is connecting to JMX from within a cluster or machines where the pod's FQDN is resolved by DNS.

5. Connect to JConsole by running this command:

```
jconsole ecs1-0.ece-server.namespace.svc.cluster.local:jmxport
```

Enabling SSL Communication When Separate Clusters for BRM and ECE

If BRM and ECE are located in different Kubernetes clusters or cloud native environments, enable SSL communication between BRM and the External Manager (EM) Gateway.

To enable SSL communication:

1. In the CM configuration file (*BRM_home*/**sys/cm/pin.conf**), set the **em_pointer** parameter to the host name and port of either the emgateway service or the load balancer:

```
- cm em_pointer ece ip hostname port
```

   where *hostname* is the worker node IP or LoadBalancer IP, and *port* is the emgateway service node port or LoadBalancer exposed port.

2. In your **override-values.yaml** file for **oc-cn-ece-helm-chart**, set the **emgateway.serviceFqdn** key to the dedicated worker node IP or load balancer IP.

   The emgateway pod can be scheduled on specific worker nodes using nodeSelector.

3. If this is the first time you are deploying ECE, run the **helm install** command:

```
helm install EceReleaseName oc-cn-ece-helm-chart --namespace BrmNameSpace
--values OverrideValuesFile
```

4. If you have already deployed ECE, do the following:

   a. Delete the **.brm_wallet_date** hidden files from the *ece-wallet-pvcLocation*/**brmwallet** directory, where *ece-wallet-pvcLocation* is the directory for the wallet PVC.

   b. Move the *ece-wallet-pvcLocation*/**brmwallet/server** directory to **server_bkp**.

   c. Perform a rolling restart of the ecs1 pod by incrementing the **restartCount** key in your **override-values.yaml** file and then running a **helm upgrade** command. See "Rolling Restart of ECE Pods" for more information.

d. Delete the emgateway pods. This enables the pods to read the updated BRM Server wallet entries.

e. Run the **helm upgrade** command to update the ECE Helm chart:

```
helm upgrade EceReleaseName oc-cn-ece-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

Using Third-Party Libraries and Custom Mediation Specifications

To use third-party libraries and custom mediation specifications with ECE cloud native:

1. Place all third-party libraries in the **3rdparty_jars** directory inside external-pvc.
2. Place your custom mediation specifications in the **ece_custom_data** directory inside external-pvc.
3. Run the **helm install** command:

   ```
   helm install EceReleaseName oc-cn-ece-helm-chart --namespace
   BrmNameSpace --values OverrideValuesFile
   ```

   where:

   - *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.
   - *EceReleaseName* is the release name for **oc-cn-ece-helm-chart** and is used to track this installation instance. It must be different from the one used for the BRM Helm chart.
   - *OverrideValuesFile* is the path to the YAML file that overrides the default configurations in the chart's **values.yaml** file.

If you need to load custom mediation specifications into ECE cloud native after the ECE cluster is set up, do the following:

1. Do one of the following:
   - For Patch Set 8 and later releases, stop the configloader pod.
   - For Patch Set 7 and earlier releases, delete the **configloader** job.

   Your mediation specifications will be loaded into ECE cache from the **configloader** job or configloader pod.
2. Place your custom mediation specifications in the **ece_custom_data** directory inside external-pvc.
3. Connect to JConsole. See "Creating a JMX Connection to ECE Using JConsole".
4. In JConsole, click the **MBeans** tab.
5. Expand the **ECE Configuration** node.
6. Expand **migration.loader**.
7. Expand **Attributes**.

8. Set the **configObjectsDataDirectory** attribute to**/home/charging/opt/ECE/oceceserver/sample_data/config_data/specifications/**.

   This will load all mediation specifications that are placed inside the **specifications** directory, including those in the **ece_custom_data** directory.

   > **Note:**
   >
   > To load only specific mediation specifications, set the **configObjectsDataDirectory** attribute to the absolute path where the specifications are located (that is, the external-pvc pod's mounted path). For example, set the attribute to **/home/charging/ext/ece_custom_data** or **/home/charging/opt/ECE/oceceserver/sample_data/config_data/specifications/ece_custom_data**.

9. Exit JConsole.
10. In your **override-values.yaml** file for **oc-cn-ece-helm-chart**, set the **migration.loader.configObjectsDataDirectory** key to the same value as specified in step 8.
11. Run the **helm upgrade** command to update the ECE Helm release:

```
helm upgrade EceReleaseName oc-cn-ece-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

Setting Up ECE Cloud Native in Firewall-Enabled Environments

To set up your ECE cloud native services in a firewall-enabled environment, do the following:

1. Ensure that the **conntrack** library is installed on your system. The library must be installed so Coherence can form clusters correctly. Most Kubernetes distributions install it for you.

   You can check whether the library is installed by running this command:

```
rpm -qa | grep conntrack
```

   If it is installed, you should see output similar to the following:

```
libnetfilter_conntrack-1.0.6-1.el7_3.x86_64
conntrack-tools-1.4.4-4.el7.x86_64
```

2. Kubernetes distributions can create iptables rules that block some types of traffic that Coherence requires to form clusters. If you are not able to form clusters, do the following:

   a. Check whether iptables rules are blocking traffic by running the following command:

```
sudo iptables -t nat -v -L POST_public_allow -n
```

If you have entries in the chain, you will see output similar to the following. Sample chain entries are shown in bold.

```
Chain POST_public_allow (1 references)
pkts bytes target prot opt in out source destination
53 4730 MASQUERADE all -- * !lo 0.0.0.0/0 0.0.0.0/0
0 0 MASQUERADE all -- * !lo 0.0.0.0/0 0.0.0.0/0
```

b. Remove any chain entries. To do so, run this command for each chain entry:

```
iptables -t nat -v -D POST_public_allow 1
```

c. Ensure that the chain entries have been removed by running this command:

```
sudo iptables -t nat -v -L POST_public_allow -n
```

If all chain entries have been removed, you will see something similar to the following:

```
Chain POST_public_allow (1 references)
pkts bytes target prot opt in out source destination
```

3. Open ports on the firewall for the following:
   - The ECE coherence cluster. That is, if the **coherencePort** key in your **override-values.yaml** file for **oc-cn-ece-helm-chart** is configured as **15000/tcp** or **15000/udp**, open them on the firewall service.
   - Open port 19612/tcp on the firewall for the pod init check done by the metric service.
   - Open a port on the firewall configured as jmxPort for JMX connection with ecs1 pod and node-ports for other ece services in **values.yaml**.
   - Ensure that ports specific to the network plugin, such as flannel and coredns, are open on the firewall.
   - Ensure that ports required by the volume provisioner are open on the firewall.
4. Add your network interface and worker node subnets to your firewall by doing the following:

   a. Look up the network interface that the Kubernetes cluster uses for communication:

```
sudo ip a
```

   The network interface is returned.

   b. Add the network interface to the firewall's trusted zone.

   For example, to change the subnet and interface specific to your cluster:

```
sudo firewall-cmd --zone=trusted --add-interface=cni0 —permanent”
```

c. (Optional) Add worker node subnets to the firewall's trusted zone. For example:

```
sudo firewall-cmd --permanent --zone=trusted --add-source=ipAddress/16
sudo firewall-cmd --permanent --zone=trusted --add-source=ipAddress/16
```

d. Restart the firewall services.

Enabling Federation in ECE

Enabling federation in ECE allows you to easily manage and monitor your ecs pods across multiple clusters in the federation. You enable federation by adding each Kubernetes cluster as a member of the Coherence federation, specifying which cluster is the primary cluster and which ones are secondary clusters, specifying how to connect to the ECE service, and adding the ecs pod to JMX.

To enable federation in ECE:

1. Set up the primary cluster by updating these keys in your **override-values.yaml** file for **oc-cn-ece-helm-chart**:

   Note:

   Set the **jvmCoherenceOpts** keys in each **charging.***coherenceMemberName* section with Coherence Federation parameters for the primary and secondary cluster.

   - **charging.clusterName**: Set this to the name of your primary cluster.
   - **charging.isFederation**: Set this to **true**. This specifies that the cluster is a participant in a federation.
   - **charging.primaryCluster**: Set this to **true**.
   - **charging.secondaryCluster**: Set this to **false**.
   - **charging.cluster.primary.eceServiceName**: Set this to the ECE service name that creates the Kubernetes cluster with all ECE components in the primary cluster.
   - **charging.cluster.primary.eceServicefqdnOrExternalIP**: Set this to the fully qualified domain name (FQDN) of the ECE service running in the primary cluster. For example: **ece-server.***NameSpace***.svc.cluster.local**.
   - **charging.cluster.secondary.eceServiceName**: Set this to the ECE service name that creates the Kubernetes cluster with all ECE components in the secondary cluster.
   - **charging.cluster.secondary.eceServicefqdnOrExternalIP**: Set this to the FQDN of the ECE service. For example: **ece-server.***NameSpace***.svc.cluster.local**.

2. Install **oc-cn-ece-helm-chart** by running this command from the **helmcharts** directory:

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

   This brings up the necessary pods in the primary cluster.

3. Set up the secondary cluster by updating these keys in your **override-values.yaml** file for **oc-cn-ece-helm-chart**:

   Note:

   Set the **jvmCoherenceOpts** keys in each **charging.***coherenceMemberName* section with Coherence Federation parameters for the primary and secondary cluster.

   - **charging.clusterName**: Set this to the name of your secondary cluster.
   - **charging.isFederation**: Set this to **true**.
   - **charging.secondaryCluster**: Set this to **true**.
   - **charging.primaryCluster**: Set this to **false**.
   - **charging.cluster.primary.eceServiceName**: Set this to the ECE service name that creates the Kubernetes cluster with all ECE components in the primary cluster.
   - **charging.cluster.primary.eceServicefqdnOrExternalIP**: Set this to the fully qualified domain name (FQDN) of the ECE service running in the primary cluster. For example: **ece-server.***NameSpace***.svc.cluster.local**.
   - **charging.cluster.secondary.eceServiceName**: Set this to the ECE service name that creates the Kubernetes cluster with all ECE components in the secondary cluster.
   - **charging.cluster.secondary.eceServicefqdnOrExternalIP**: Set this to the FQDN of the ECE service in the secondary cluster. For example: **ece-server-2.***NameSpace***.svc.cluster.local**.

4. Install **oc-cn-ece-helm-chart** by running this command from the **helmcharts** directory:

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

   This brings up the necessary pods in the secondary cluster.

5. Invoke federation from the primary production site to your secondary production sites by connecting from JConsole of the ecs1 pod.

   a. Update the label for the ecs1-0 pod:

      ```
      kubectl label -n NameSpace po ecs1-0 ece-jmx=ece-jmx-external
      ```

   b. Update the **/etc/hosts** file on the remote machine with the worker node of ecs1-0:

      ```
      IP_OF_WORKER_NODE ecs1-0.ece-server.namespace.svc.cluster.local
      ```

   c. Connect to JConsole:

      ```
      jconsole ecs1-0.ece-server.namespace.svc.cluster.local:31022
      ```

JConsole starts.

d. Invoke **start()** and **replicateAll()** with the secondary production site name from the coordinator node of each federated cache in JMX. To do so:

   i. Expand the **Coherence** node, expand **Federation**, expand **BRMFederatedCache**, expand **Coordinator**, and then expand **Coordinator**. Click on **start(***BRM2***)** and **replicateAll(***BRM2***)**, where *BRM2* is the secondary production site name.

   ii. Expand the **Coherence** node, expand **Federation**, expand **OfferProfileFederatedCache**, expand **Coordinator**, and then expand **Coordinator**. Click on **start(***BRM2***)** and **replicateAll(***BRM2***)**.

   iii. Expand the **Coherence** node, expand **Federation**, expand **ReplicatedFederatedCache**, expand **Coordinator**, and then expand **Coordinator**. Click on **start(***BRM2***)** and **replicateAll(***BRM2***)**.

   iv. Expand the **Coherence** node, expand **Federation**, expand **XRefFederatedCache**, expand **Coordinator**, and then expand **Coordinator**. Click on **start(***BRM2***)** and **replicateAll(***BRM2***)**.

e. From the secondary production site, verify that data is being federated from the primary production site to the secondary production sites, and that all pods are running.

Enabling Parallel Pod Management in ECE

You can configure the Kubernetes StatefulSet controller to start all ecs pods at the same time by enabling parallel pod management. To do so:

1. Open your **override-values.yaml** file for **oc-cn-ece-helm-chart**.
2. Set the **parallelPodManagement** key to one of the following:
   - **true**: The ecs pods will start in parallel. You must scale down the replicas manually. See "Scaling Down the ecs Pod Replicas".
   - **false**: The ecs pods will wait for a pod to be in the **Running and Ready** state or completely stopped prior to starting or stopping another pod. This is the default.
3. Deploy the ECE Helm chart (**oc-cn-ece-helm-cart**):

```
helm install EceReleaseName oc-cn-ece-helm-chart --namespace BrmNameSpace
--values OverrideValuesFile
```

Scaling Down the ecs Pod Replicas

To scale down ecs pod replicas when **parallelPodManagement** is enabled:

1. Ensure that the ecs pod is in the **Usage Processing** state.
2. Check the ecs pod's current replica count by running one of these commands:

   -
     ```
     kubectl get po -n BrmNameSpace | grep -i ecs
     ```

   -
     ```
     kubectl get sts ecs -n BrmNameSpace
     ```

   where *BrmNameSpace* is the name space in which the BRM Kubernetes objects reside.

3. Reduce the ecs pod's replica count by one by running this command:

   ```
   kubectl scale sts ecs --replicas=newReplicaCount -n BrmNameSpace
   ```

   where *newReplicaCount* is the current replica count reduced by one.

   For example, if the current replica count is 6, you would run this command to scale down ecs to 5 replicas:

   ```
   kubectl scale sts ecs --replicas=5 -n BrmNameSpace
   ```

4. Wait for the replica to stop.
5. Continue reducing the ecs pod replica count until you reach the desired amount.

   The desired minimum ecs replica count is 3.

Customizing SDK Source Code

If you want to customize the ECE SDK source code for any of the sample scripts or Java code, the SDK directory with all of these files is exposed under the SDK PVC. You can change any file in the PVC, and the same will be reflected inside the pod.

When you run the SDK job with the **build** and **run** options, the customized code is built and run from the job.

22

Managing Persisted Data in the Oracle Database

Learn about data persistence and the tasks for managing Elastic Charging Engine (ECE) data stored in an Oracle Communications Billing and Revenue Management (BRM) cloud native database.

Topics in this document:

Enabling Persistence in ECE

You can set up ECE to persist its cache data in the Oracle database, creating a permanent backup of the cache in case a node fails, a partition is lost, or so on. ECE automatically recovers the cache data from the persistence database when it is needed.

When persistence is enabled, the ECE core components, such as Customer Updater, Pricing Updater, and configLoader, persist the following at start up:

- The data published from BRM and PDC into ECE cache
- The mediation specification data loaded into ECE cache
- The data that is synchronized or received from BRM
- Other data such as balance, top-up history, recurring bundle history, rated events, and Portal object IDs (POIDs)

During installation, upgrade, auto-recovery, and pod restart, ECE uses the Kubernetes REST API to:

- Automatically update the **charging-settings-***namespace* ConfigMap to enable reloading of cache data from the persistence database
- Retrieve the metadata from ECE statefulsets and pods
- Automatically apply management labels to ecs pods

To configure ECE cloud native for persistence:

1. Configure ECE to reload cache data, retrieve metadata, and apply management labels during installation, upgrade, auto-recovery, and pod restart. To do so, configure the **ece-***namespace* service account to authenticate the API server.

   For information about the rules defined in the role-based access control (RBAC) ece-*namespace*, see the **ece-clusterrole-sa.yaml** file in the ECE Helm chart.

2. Enable and configure persistence in ECE cloud native. To do so, set these keys in the **override-values.yaml** file for **oc-cn-ece-helm-chart**:

Note:

Ensure the persistence tablespace names are all uppercase.

```
secretEnv:
   PERSISTENCEDATABASEPASSWORD:
     - schema: 1
       PASSWORD: password
   PERSISTENCEDBAPASSWORD:
     - schema: 1
       PASSWORD: password     # SYSDBA user
   PERSISTENCEDATABASEKEYPASS:
     - schema: 1
       PASSWORD: password
charging:
   persistenceEnabled: "true"
   cachePersistenceConfigurations:
      cachePersistenceConfigurationList:
         - clusterName: "BRM"
           persistenceStoreType: "OracleDB"
           persistenceConnectionName: "oraclePersistence1"
           reloadThreadPoolSize: "10"
           configLoadFromPersistence: "true"
           pricingLoadFromPersistence: "true"
           customerLoadFromPersistence: "true"
           partitionLossRecoverFromPersistence: "true"
           writeBehindThreadPoolSize: "1"
   connectionConfigurations:
      OraclePersistenceConnectionConfigurations:
         - clusterName: "BRM"
           schemaNumber: "1"
           name: "oraclePersistence1"
           dbSysDBAUser: "sys"
           dbSysDBARole: "sysdba"
           userName: "ece"
           hostName: ""
           port: "1521"
           sid: ""
           service: ""
           tablespace: "ECETABLE"
           temptablespace: "ECETEMP"
           cdrstoretablespace: "ECECDRTABLESPACE"
           cdrstoreindexspace: "ECECDRINDEXSPACE"
           jdbcUrl: ""
           retryCount: "3"
           retryInterval: "1"
           maxStmtCacheSize: "100"
           connectionWaitTimeout: "300"
           timeoutConnectionCheckInterval: "300"
           inactiveConnectionTimeout: "300"
           databaseConnectionTimeout: "600"
           persistenceInitialPoolSize: "4"
           persistenceMinPoolSize: "4"
           persistenceMaxPoolSize: "12"
           reloadInitialPoolSize: "0"
           reloadMinPoolSize: "0"
           reloadMaxPoolSize: "20"
           dbSSLEnabled: "true"
```

```
                dbSSLType: "twoway"
                sslServerCertDN: "DC=local,DC=oracle,CN=pindb"
                trustStoreLocation: "/home/charging/ext/ece_ssl_db_wallet/schema1/
cwallet.sso"
                trustStoreType: "SSO"
                walletLocation: "/home/charging/wallet/ecewallet/"
                cdrStorePartitionCount: "32"
                queryTimeout: "5"
```

When you deploy **oc-cn-ece-helm-chart** with this configuration, the Helm chart creates a schema user if one doesn't already exist, creates ECE tables, creates indexes, and runs stored procedures.

To see the ECE deployment logs, run this command:

```
kubectl logs -f EcePersistenceJobPod -n BrmNameSpace
```

where *EcePersistenceJobPod* is the name of the pod where **ece-persistence-job** is deployed, and *BrmNameSpace* is the name space in which to create BRM Kubernetes objects for the BRM Helm chart.

Re-Creating the ECE Schema After Deployment

If you want to re-create the ECE schema, any table, or any index after the ECE Helm chart is already deployed, do the following:

1. Delete the ECE Helm chart.
2. Delete the pre-existing **ece-persistence-job** from your system by running this command:

   ```
   kubectl delete job ece-persistence-job -n BrmNameSpace
   ```

3. Install the Helm chart again by running the following command:

   ```
   helm install ece -n BrmNameSpace oc-cn-ece-helm-chart [--no-hooks]
   ```

   Note:

   Include the **--no-hooks** argument only if everything needed for persistence is already in the persistence database.

Loading Only Partial Data into ECE Cache

You can optionally configure ECE to load only partial data from the persistence database into the ECE cache. In this case, the initial load of data into ECE cache includes data only up to a specified minimum amount (**back-low-limit**). If the data required for processing a usage request is not available in the ECE cache, ECE loads that data into the ECE cache from the persistence database and evicts some other data from the ECE cache. This ensures that the maximum limit (**back-high-limit**) is not exceeded. Later, when you restart the ECE system, ECE loads the most recently used data into the ECE cache.

For more information, see "Enabling Partial Loading of Data" in *BRM System Administrator's Guide*.

To load only partial data into ECE cache, set these parameters for the **charging-cache-config-persistence.xml** file in the **oc-cn-ece-helm-chart/templates/charging-settings.yaml** ConfigMap:

- **back-high-limit**: The maximum amount of data that can be loaded into the ECE cache.
- **back-low-limit**: The minimum amount of data that can be loaded or reloaded into the ECE cache from the persistence database.

Incremental Customer Loading in ECE Cache

By default, the customerupdater pod loads all customer data from the BRM database into ECE cache at start up, but you can configure the pod to load customer data incrementally.

To incrementally load customer data into ECE cache:

1. Configure the customerupdater pod to load only an initial set of customers into ECE cache and bring ECE to the UsageProcessing state by setting these keys in the **override-values.yaml** file for **oc-cn-ece-helm-chart**:
   - **job.customerloader.runjob**: Set this to **false**.
   - **charging.incrementalCustomerLoad**: Set this to **true**.
   - **migration.loader.initialCustomerLoadFilterQuery**: Set this to a query such as **"and ROWNUM <= 1"** to load one customer.
2. Install the ECE Helm chart.
3. Load the remaining customers incrementally into ECE cache by setting these keys in the **override-values.yaml** file for **oc-cn-ece-helm-chart**:
   - **job.customerloader.runjob**: Set this to **true**.
   - **job.customerloader.command**: Set this to **-incremental** *customer_updater_schema_name*, where *customer_updater_schema_name* is the schema name specified for the customerupdater pod.
   - **charging.incrementalCustomerLoad**: Set this to **true**.
   - **migration.loader.incrementalCustomerLoadFilterQuery**: Set this to a query such as **"and POID_ID0 NOT IN (select POID_ID0 from ACCOUNT_T where POID_ID0 <> 1 and ROWNUM <= 1)"** to load remaining customers.
4. Perform a Helm upgrade by running this command:

   ```
   helm upgrade EceReleaseName oc-cn-ece-helm-chart --values OverrideValuesFile
   -n BrmNameSpace
   ```

   where *EceReleaseName* is the release name for **oc-cn-ece-helm-chart** and is used to track this installation instance, and *OverrideValuesFile* is the path to the YAML file that overrides the default configurations in the chart's **values.yaml** file.

23

Configuring Disaster Recovery in ECE Cloud Native

Learn how to set up your Oracle Communications Elastic Charging Engine (ECE) cloud native services for disaster recovery.

Topics in this document:

Setting Up Active-Active Disaster Recovery for ECE

Disaster recovery provides continuity in service for your customers, and guards against data loss if a system fails. In ECE cloud native, disaster recovery is implemented by configuring two or more active production sites that are at different geographical locations. If one production site fails, another active production site takes over the traffic from the failed site.

During operation, ECE requests are routed across the production sites based on your load balancing configuration. All updates that occur in an ECE cluster at one production site are replicated to other production sites through Coherence cache federation.

For more information about the active-active disaster recovery configuration, see "About the Active-Active System" in *BRM System Administrator's Guide*.

To configure ECE cloud native for active-active disaster recovery:

1. In each Kubernetes cluster, expose ports on the external IP using the Kubernetes LoadBalancer service.

   The ECE Helm chart includes a sample YAML file for the LoadBalancer service (**oc-cn-ece-helm-chart/templates/ece-service-external.yaml**) that you can configure for your environment.

2. On your primary production site, update the **override-values.yaml** file with the external IP of the LoadBalancer service, the federation-related parameters, the JMX port for the monitoring agent, the active-active disaster recovery parameters, and so on.

   The following shows example **override-values.yaml** file settings for a primary production site:

```
monitoringAgent:
   monitoringAgentList:
      - name: "monitoringagent1"
        replicas: 1
        jmxport: "31020"
        jmxEnabled: "true"
        jvmJMXOpts: "-Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false -
Dcom.sun.management.jmxremote.local.only=false  -
Dcom.sun.management.jmxremote.password.file=../config/
jmxremote.password -Dsecure.access.name=admin -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.port=31020 -
Dcom.sun.management.jmxremote.rmi.port=31020"
        jvmOpts: "-Djava.net.preferIPv4Addresses=true"
        jvmGCOpts: ""
        restartCount: "0"
        nodeSelector: "node1"
      - name: "monitoringagent2"
        replicas: 1
        jmxport: "31021"
        jmxEnabled: "true"
        jvmJMXOpts: "-Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false -
Dcom.sun.management.jmxremote.local.only=false  -
Dcom.sun.management.jmxremote.password.file=../config/
jmxremote.password -Dsecure.access.name=admin -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.port=31021 -
Dcom.sun.management.jmxremote.rmi.port=31021"
        jvmOpts: "-Djava.net.preferIPv4Addresses=true"
        jvmGCOpts: ""
        restartCount: "0"
        nodeSelector: "node2"
charging:
   jmxport: "31022"
   coherencePort: "31015"
...
...
   clusterName: "BRM"
   isFederation: "true"
   primaryCluster: "true"
   secondaryCluster: "false"
   clusterTopology: "active-active"
   cluster:
      primary:
         clusterName: "BRM"
         eceServiceName: ece-server
         eceServicefqdnOrExternalIP: "0.1.2.3"
      secondary:
         - clusterName: "BRM2"
           eceServiceName: ece-server
           eceServicefqdnOrExternalIp: "0.1.2.3"
   federatedCacheScheme:
      federationPort:
         brmfederated: 31016
         xreffederated: 31017
         replicatedfederated: 31018
         offerProfileFederated: 31019
```

3. On your secondary production site, update the **override-values.yaml** file with the external IP of the LoadBalancer service, the federation-related parameters, the JMX port for the monitoring agent, the active-active disaster recovery parameters, and so on.

   The following shows example settings in an **override-values.yaml** for a secondary production site:

```
monitoringAgent:
   monitoringAgentList:
      - name: "monitoringagent1"
        replicas: 1
        jmxport: "31020"
        jmxEnabled: "true"
        jvmJMXOpts: "-Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false -
Dcom.sun.management.jmxremote.local.only=false  -
Dcom.sun.management.jmxremote.password.file=../config/jmxremote.password -
Dsecure.access.name=admin -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.port=31020 -
Dcom.sun.management.jmxremote.rmi.port=31020"
        jvmOpts: "-Djava.net.preferIPv4Addresses=true"
        jvmGCOpts: ""
        restartCount: "0"
        nodeSelector: "node1"
      - name: "monitoringagent2"
        replicas: 1
        jmxport: "31021"
        jmxEnabled: "true"
        jvmJMXOpts: "-Dcom.sun.management.jmxremote -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.ssl=false -
Dcom.sun.management.jmxremote.local.only=false  -
Dcom.sun.management.jmxremote.password.file=../config/jmxremote.password -
Dsecure.access.name=admin -
Dcom.sun.management.jmxremote.authenticate=false -
Dcom.sun.management.jmxremote.port=31021 -
Dcom.sun.management.jmxremote.rmi.port=31021"
        jvmOpts: "-Djava.net.preferIPv4Addresses=true"
        jvmGCOpts: ""
        restartCount: "0"
        nodeSelector: "node2"
charging:
   jmxport: "31022"
   coherencePort: "31015"
...
...
  clusterName: "BRM2"
   isFederation: "true"
   primaryCluster: "false"
   secondaryCluster: "true"
```

```
clusterTopology: "active-active"
cluster:
   primary:
      clusterName: "BRM"
      eceServiceName: ece-server
      eceServicefqdnOrExternalIP: "0.1.2.3"
   secondary:
      - clusterName: "BRM2"
        eceServiceName: ece-server
        eceServicefqdnOrExternalIp: "0.1.2.3"
federatedCacheScheme:
   federationPort:
      brmfederated: 31016
      xreffederated: 31017
      replicatedfederated: 31018
      offerProfileFederated: 31019
```

4. On your primary and secondary production sites, add the **customerGroupConfigurations** and **siteConfigurations** sections to the **override-values.yaml** file.

   The following shows example settings to add to the **override-values.yaml** file in your primary and secondary production sites:

```
customerGroupConfigurations:
   - name: "customergroup1"
     clusterPreference:
       - priority: "1"
         routingGatewayList: "0.1.2.3:31500"
         name: "BRM"
       - priority: "2"
         routingGatewayList: "0.1.2.3:31500"
         name: "BRM2"
   - name: "customergroup2"
     clusterPreference:
       - priority: "2"
         routingGatewayList: "0.1.2.3:31500"
         name: "BRM"
       - priority: "1"
         routingGatewayList: "0.1.2.3:31500"
         name: "BRM2"
siteConfigurations:
   - name: "BRM"
     affinitySiteNames: "BRM2"
     monitorAgentJmxConfigurations:
       - name: "monitoringagent1"
         host: "node1"
         jmxPort: "31020"
         disableMonitor: "true"
       - name: "monitoringagent2"
         host: "node2"
         jmxPort: "31021"
         disableMonitor: "true"
   - name: "BRM2"
     affinitySiteNames: "BRM"
     monitorAgentJmxConfigurations:
       - name: "monitoringagent1"
         host: "node1"
         jmxPort: "31020"
         disableMonitor: "true"
       - name: "monitoringagent2"
         host: "node2"
         jmxPort: "31021"
         disableMonitor: "true"
```

5. In your **override-values.yaml** file, configure **kafkaConfigurationList** with both primary and secondary site Kafka details.

   The following shows example settings to add to the **override-values.yaml** file in your primary and secondary production sites:

```
kafkaConfigurationList:
   -  name: "BRM"
      hostname:"hostname:port"
      topicName: "ECENotifications"
      suspenseTopicName: "ECESuspenseQueue"
      partitions: "200"
      kafkaProducerReconnectionInterval: "120000"
      kafkaProducerReconnectionMax: "36000000"
      kafkaDGWReconnectionInterval: "120000"
      kafkaDGWReconnectionMax: "36000000"
      kafkaBRMReconnectionInterval: "120000"
      kafkaBRMReconnectionMax: "36000000"
      kafkaHTTPReconnectionInterval: "120000"
      kafkaHTTPReconnectionMax: "36000000"
   -  name: "BRM2"
      hostname:"hostname:port"
      topicName: "ECENotifications"
      suspenseTopicName: "ECESuspenseQueue"
      partitions: "200"
      kafkaProducerReconnectionInterval: "120000"
      kafkaProducerReconnectionMax: "36000000"
      kafkaDGWReconnectionInterval: "120000"
      kafkaDGWReconnectionMax: "36000000"
      kafkaBRMReconnectionInterval: "120000"
      kafkaBRMReconnectionMax: "36000000"
      kafkaHTTPReconnectionInterval: "120000"
      kafkaHTTPReconnectionMax: "36000000"
```

6. If data persistence is enabled, configure a primary and secondary Rated Event Formatter instance on your primary and secondary production sites for each site in the **ratedEventFormatter** section of the **override-values.yaml** file.

   The following shows example settings to add to the **override-values.yaml** file in your primary and secondary production sites:

```
ratedEventFormatter:
   ratedEventFormatterList:
        ratedEventFormatterConfiguration:
           name: "ref_site1_primary"
           partition: "1"
           connectionName: "oracle1"
           siteName: "site1"
           threadPoolSize: "2"
           retainDuration: "0"
           ripeDuration: "30"
           checkPointInterval: "20"
           pluginPath: "ece-ratedeventformatter.jar"
           pluginType:
"oracle.communication.brm.charging.ratedevent.formatterplugin.intern
al.SampleFormatterPlugInImpl"
           pluginName: "brmCdrPluginDC1Primary"
           noSQLBatchSize: "25"
        ratedEventFormatterConfiguration:
           name: "ref_site1_secondary"
           partition: "1"
           connectionName: "oracle2"
           siteName: "site1"
           primaryInstanceName: "ref_site1_primary"
           threadPoolSize: "2"
           retainDuration: "0"
           ripeDuration: "30"
           checkPointInterval: "20"
           pluginPath: "ece-ratedeventformatter.jar"
           pluginType:
"oracle.communication.brm.charging.ratedevent.formatterplugin.intern
al.SampleFormatterPlugInImpl"
           pluginName: "brmCdrPluginDC1Primary"
           noSQLBatchSize: "25"
        ratedEventFormatterConfiguration:
           name: "ref_site2_primary"
           partition: "1"
           connectionName: "oracle2"
           siteName: "site2"
           threadPoolSize: "2"
           retainDuration: "0"
           ripeDuration: "30"
           checkPointInterval: "20"
           pluginPath: "ece-ratedeventformatter.jar"
           pluginType:
"oracle.communication.brm.charging.ratedevent.formatterplugin.intern
al.SampleFormatterPlugInImpl"
           pluginName: "brmCdrPluginDC1Primary"
           noSQLBatchSize: "25"
        ratedEventFormatterConfiguration:
           name: "ref_site2_secondary"
           partition: "1"
           connectionName: "oracle1"
           siteName: "site2"
           primaryInstanceName: "ref_site2_primary"
           threadPoolSize: "2"
           retainDuration: "0"
           ripeDuration: "30"
           checkPointInterval: "20"
           pluginPath: "ece-ratedeventformatter.jar"
           pluginType:
```

```
"oracle.communication.brm.charging.ratedevent.formatterplugin.internal.Sam
pleFormatterPlugInImpl"
           pluginName: "brmCdrPluginDC1Primary"
           noSQLBatchSize: "25"
```

The **siteName** property determines the site that the instance processes rated events for. This lets you configure secondary instances as backups for remote sites. The sample specifies that the **ref_site1_secondary** instance running is running at site 2, but processes rated events federated from site 1 in case of an outage.

For more information about Rated Event Formatter in active-active systems, see "About Rated Event Formatter in a Persistence-Enabled Active-Active System" in *BRM System Administrator's Guide*.

7. Depending on whether persistence is enabled in ECE, do one of the following:

   - If persistence is enabled, add the **cachePersistenceConfigurations** and **connectionConfigurations.OraclePersistenceConnectionConfigurations** sections to your **override-values.yaml** file on both primary and secondary production sites.

     The following shows example settings to add to the **override-values.yaml** file on your primary and secondary sites:

```
 cachePersistenceConfigurations:
      cachePersistenceConfigurationList:
        -  clusterName: "BRM"
           persistenceStoreType: "OracleDB"
           persistenceConnectionName: "oraclePersistence1"
...
...
        -  clusterName: "BRM2"
           persistenceStoreType: "OracleDB"
           persistenceConnectionName: "oraclePersistence2"
...
...
   connectionConfigurations:
         OraclePersistenceConnectionConfigurations:
            - clusterName: "BRM"
              name: "oraclePersistence1"
...
...
            - clusterName: "BRM2"
              name: "oraclePersistence2"
...
...
```

   - If persistence is disabled, add the **ratedEventPublishers** and **NoSQLConnectionConfigurations** sections to your **override-values.yaml** file on both primary and secondary production sites.

     The following shows example settings to add to the **override-values.yaml** file on your primary and secondary sites:

```
   ratedEventPublishers:
     -  clusterName: "BRM"
        noSQLConnectionName: "noSQLConnection1"
```

```
                threadPoolSize: "4"
             -  clusterName: "BRM2"
                noSQLConnectionName: "noSQLConnection2"
                threadPoolSize: "4"
           connectionConfigurations:
                 NoSQLConnectionConfigurations:
                    - clusterName: "BRM"
                      name: "noSQLConnection1"
...
...
                    - clusterName: "BRM2"
                      name: "noSQLConnection2"
...
...
```

8. Deploy the ECE Helm chart (**oc-cn-ece-helm-cart**) on the primary cluster and bring the primary cluster to the **Usage Processing** state.
9. Invoke federation from the primary production site to your secondary production sites by connecting from JConsole of the ecs1 pod.
   a. Update the label for the ecs1-0 pod:

      ```
      kubectl label -n NameSpace po ecs1-0 ece-jmx=ece-jmx-external
      ```

   b. Update the **/etc/hosts** file on the remote machine with the worker node of ecs1-0:

      ```
      IP_OF_WORKER_NODE ecs1-0.ece-server.namespace.svc.cluster.local
      ```

   c. Connect to JConsole:

      ```
      jconsole ecs1-0.ece-server.namespace.svc.cluster.local:31022
      ```

      JConsole starts.
   d. Invoke **start()** and **replicateAll()** with the secondary production site name from the coordinator node of each federated cache in JMX. To do so:
      i. Expand the **Coherence** node, expand **Federation**, expand **BRMFederatedCache**, expand **Coordinator**, and then expand **Coordinator**. Click on **start(***BRM2***)** and **replicateAll(***BRM2***)**, where *BRM2* is the secondary production site name.
      ii. Expand the **Coherence** node, expand **Federation**, expand **OfferProfileFederatedCache**, expand **Coordinator**, and then expand **Coordinator**. Click on **start(***BRM2***)** and **replicateAll(***BRM2***)**.
      iii. Expand the **Coherence** node, expand **Federation**, expand **ReplicatedFederatedCache**, expand **Coordinator**, and then expand **Coordinator**. Click on **start(***BRM2***)** and **replicateAll(***BRM2***)**.
      iv. Expand the **Coherence** node, expand **Federation**, expand **XRefFederatedCache**, expand **Coordinator**, and then expand **Coordinator**. Click on **start(***BRM2***)** and **replicateAll(***BRM2***)**.

e. From the secondary production site, verify that data is being federated from the primary production site to the secondary production sites, and that all pods are running.

After federation completes, your primary and secondary production sites move to the **Usage Processing** state, and the monitoring agent pods are spawned.

10. When all pods are up and ready on each site, scale down and then scale up the monitoring agent pods in each production site. This synchronizes the monitoring agent pods with the other pods in the cluster.

Note:

Repeat these steps if you need to scale up or down any pod after the monitoring agent is initialized.

a. Scale down **monitoringagent1** to **0**:

```
kubectl -n NameSpace scale deploy monitoringagent1 --replicas=0
```

b. Wait for **monitoringagent1** to stop and then scale it back up to **1**.

```
kubectl -n NameSpace scale deploy monitoringagent1 --replicas=1
```

c. Scale down **monitoringagent2** to **0**:

```
kubectl -n NameSpace scale deploy monitoringagent2 --replicas=0
```

d. Wait for **monitoringagent2** to stop and then scale it back up to **1**.

```
kubectl -n NameSpace scale deploy monitoringagent2 --replicas=1
```

11. Verify that the monitoring agent logs are collecting metrics.

Processing Usage Requests in Site Receiving Request

By default, the ECE active-active disaster recovery mode processes usage requests according to the preferred site assignments in the customerGroup list. For example, if subscriber A's preferred primary site is site 1, ECE processes subscriber A's usage requests in site 1. If subscriber A's usage request is received by production site 2, it is sent to production site 1 for processing.

You can configure the ECE active-active mode to process usage requests in the site that receives the request, regardless of the subscriber's preferred site. For example, if a subscriber's usage request is received by production site 1, it is processed in production site 1. Similarly, if the usage request is received by production site 2, it is processed in production site 2.

> **Note:**
>
> This configuration does not apply to usage charging requests for sharing group members. Usage requests for sharing group members are processed in the same site as the sharing group parent.

To configure the ECE active-active mode to process usage requests in the site that receives the request irrespective of the subscriber's preferred site:

1. In your **override-values.yaml** file for **oc-cn-ece-helm-chart**, set the **charging.brsConfigurations.brsConfigurationList.brsConfig.skipActiveActivePreferredSiteRouting** key to **true**.
2. Run the helm upgrade command to update your ECE Helm release:

   ```
   helm upgrade EceReleaseName oc-cn-ece-helm-chart --values OverrideValuesFile
   -n BrmNameSpace
   ```

   where:

   - *EceReleaseName* is the release name for **oc-cn-ece-helm-chart** and is used to track the installation instance.
   - *OverrideValuesFile* is the path to the YAML file that overrides the default configurations in the **oc-cn-ece-helm-chart/values.yaml** file.
   - *BrmNameSpace* is the namespace in which to create BRM Kubernetes objects for the BRM Helm chart.

Stopping ECE from Routing to a Failed Site

When an active production site fails, you must notify the monitoring agent about the failed site. This stops ECE from rerouting requests to the failed production site.

To notify the monitoring agent about a failed production site:

1. Connect to the monitoring agent through JConsole:
   a. Update **/etc/hosts** with the worker IP of the monitoringagent1 pod.

      ```
      worker_IP ece-monitoringagent-service-1
      ```

   b. Connect through JConsole by running this command:

      ```
      jconsole ece-monitoringagent-service-1:31020
      ```

      JConsole starts.
2. Expand the **ECE Monitoring** node.
3. Expand **Agent**.
4. Expand **Operations**.
5. Set the **failoverSite()** operation to the name of the failed production site.

You can also use the **activateSecondaryInstanceFor** operation to fail over to a backup Rated Event Formatter as described in "Activating a Secondary Rated Event

Formatter Instance." See "Resolving Rated Event Formatter Instance Outages" in *BRM System Administrator's Guide* for conceptual information about how to resolve Rated Event Formatter outages.

Adding Fixed Site Back to ECE System

Notify the monitoring agent after a failed production site starts functioning again. This allows ECE to route requests to the site again.

To add a fixed site back to the ECE disaster recovery system:

1. Connect to the monitoring agent through JConsole:
    a. Update **/etc/hosts** with the worker IP of the monitoringagent1 pod.

    ```
    worker_IP ece-monitoringagent-service-1
    ```

    b. Connect through JConsole by running this command:

    ```
    jconsole ece-monitoringagent-service-1:31020
    ```

    JConsole starts.
2. Expand the **ECE Monitoring** node.
3. Expand **Agent**.
4. Expand **Operations**.
5. Set the **recoverSite()** operation to the name of the original production site.

Activating a Secondary Rated Event Formatter Instance

If a primary Rated Event Formatter instance is down, you can activate a secondary instance to take over rated event processing.

To activate a secondary Rated Event Formatter instance:

1. Connect to the ratedeventformatter pod through JConsole by doing the following:
    a. Update the label for the **ratedeventformatter** pod:

    ```
    kubectl label -n NameSpace po ratedeventformatter1-0 ece-jmx=ece-jmx-
    external
    ```

    > **Note:**
    >
    > ece-jmx-service-external has only one endpoint as IP of the **ratedeventformatter** pod.

   b. Update the **/etc/hosts** file on the remote machine with the worker node of the ratedeventformatter pod.

      ```
      IP_OF_WORKER_NODE ratedeventformatter1-0.ece-
      server.namespace.svc.cluster.local
      ```

   c. Connect through JConsole by running this command:

      ```
      jconsole redeventformatter1-0.ece-
      server.namespace.svc.cluster.local:31022
      ```

      JConsole starts.

2. Expand the **ECE Monitoring** node.
3. Expand **RatedEventFormatterMatrices**.
4. Expand **Operations**.
5. Run the **activateSecondaryInstance** operation.

   The secondary Rated Event Formatter instance begins processing rated events.

Getting Rated Event Formatter Checkpoint Information

You can retrieve information about the last Rated Event Formatter checkpoint that was committed to the database.

To retrieve information about the last Rated Event Formatter checkpoint:

1. Connect to the **ecs1** pod through JConsole. See "Creating a JMX Connection to ECE Using JConsole" for more information.
2. Expand the **ECE Configuration** node.
3. Expand the database connection you want checkpoint information from.
4. Expand **Operations**.
5. Run the **queryRatedEventCheckPoint** operation.

   Checkpoint information appears for all Rated Event Formatter instances using the database connection. Information includes site, schema, and plugin names as well as the time of the most recent checkpoint.

24

Managing ECE Pods

Learn how to manage the Elastic Charging Engine (ECE) pods in your Oracle Communications Billing and Revenue Management (BRM) cloud native environment.

Topics in this document:

- Setting up Autoscaling of ECE Pods
- Rolling Restart of ECE Pods

Scaling Kubernetes Pods

Kubernetes pods that are created as part of the deployment can be scaled up or down. By default, three ECE server replicas are created during the installation process.

To scale a Kubernetes pod, run this command:

> **Note:**
>
> Kubernetes pods can be scaled only if the partitions are balanced.

```
kubectl scale statefulsets componentName --replicas=newReplicaCount
```

If scaling doesn't occur, check the **partitionUnbalanced** count under **Coherence.service.partitionUnbalanced** for all cache services.

Setting up Autoscaling of ECE Pods

You can use the Kubernetes Horizontal Pod Autoscaler to automatically scale up or scale down the number of ECE pod replicas based on a pod's CPU or memory utilization. In BRM cloud native deployments, the Horizontal Pod Autoscaler monitors and scales these ECE pods:

- ecs
- ecs1
- httpgateway

Changing the number of replicas in an ECE autoscalable ReplicaSet results in a re-balancing of the in-memory cache distribution across the replicas. This re-balancing activity consumes incremental CPU and memory resources and can take multiple seconds to complete. Therefore, an ECE autoscaling design should attempt to strike a balance between optimizing infrastructure resource usage and minimizing changes to the number of replicas in a ReplicaSet due to autoscaling.

> **Note:**
>
> Enabling autoscaling of ECE pods in a production environment should be preceded by comprehensive validation of all scenarios expected to trigger autoscaling (scale up and scale down). It is recommended that this validation be performed in a demonstration or test environment using infrastructure equivalent to the target production infrastructure. In addition, monitoring the frequency of autoscaling is recommended to detect flapping conditions so that adjustments can be incorporated to avoid flapping.

To set up and enable autoscaling for ECE pods:

1. Ensure that your ECE cluster is set up and the system is in the **UsageProcessing** state.

   > **Note:**
   >
   > Do not enable Horizontal Pod Autoscaler for your ECE cluster until ECE reaches the **UsageProcessing** state. Enabling it during customer or balance data loading could lead to customer load failure due to re-balancing of the in-memory cache.

2. Open your **override-values.yaml** file for **oc-cn-ece-helm-chart**.
3. Enable the Horizontal Pod Autoscaler in ECE by setting the **charging.hpaEnabled** key to **true**:

   ```
   charging
      hpaEnabled: "true"
   ```

4. Specify the memory and CPU usage for each supported ECE pod. To do so, set the required keys under the **ecs**, **ecs1**, and **httpgateway***n* sections:
   - **maxReplicas**: Set this to the maximum number of pod replicas to deploy when scale up is triggered.

     If a pod's average utilization goes above **averageCpuUtilization** or **averageMemoryUtilization**, the Horizontal Pod Autoscaler increases the number of pod replicas up to this maximum count.
   - **averageCpuUtilization**: Set this as a target or threshold for average CPU usage across all of the pod's replicas with the same entry point. For example, if a cluster has four ecs pod replicas and one ecs1 pod replica, the average will be the sum of CPU usage divided by five. The default is 70% for ecs.

     The autoscaler increases or decreases the number of ecs or httpgateway pod replicas to maintain the average CPU utilization you specified across all pods.

> **Note:**
>
> Only the ecs pod and httpgateway pod (with NRF disabled) will be scaled up and down.

- **averageMemoryUtilization**: Set this as a target or threshold for average resource consumption across all of the pod's replicas, such as 1 Gi. For example, if a cluster has four ecs pod replicas and one ecs1 pod replica, the average will be the sum of memory utilization divided by five.

  The autoscaler increases or decreases the number of ecs or httpgateway pod replicas to maintain the average memory utilization you specified across all pods.

  > **Note:**
  >
  > Only the ecs pod and httpgateway pod (with NRF disabled) will be scaled up and down.

- **cpuLimit**: Set this to the maximum amount of CPU that a pod can utilize.
- **cpuRequest**: Set this to the minimum CPU amount, in milli-cores, that must be available in a Kubernetes node to deploy a pod. For example, enter **1000m** for 1 CPU core.

  If the minimum CPU amount is not available, the pod's status is set to **Pending**.
- **memoryLimit**: Set this to the maximum amount of memory that a pod can utilize. The default is 3 Gi for the ecs pod.
- **memoryRequest**: Set this to the minimum amount of memory required for a Kubernetes node to deploy a pod. The default is 2 Gi for the ecs pod.

  If the minimum amount is not available, the pod's status is set to **Pending**.
- **scaleDownStabilizationWindowSeconds**: Specifies the duration, in seconds, of the stabilization window when scaling down pods. Oracle recommends using a value of 120 seconds or more.
- **disableHpaScaleDown**: Set this to **true** to prevent the Horizontal Pod Autoscaler from scaling down the pod.

This shows sample entries for the httpgateway pod:

```
httpgateway:
   httpgatewayList:
      - coherenceMemberName: "httpgateway1"
        maxreplicas: 3
        averageCpuUtilization: 70
        averageMemoryUtilization: ""
        cpuLimit: 2000m
        cpuRequest: 1000m
        memoryLimit: 3Gi
        memoryRequest: 1Gi
        scaleDownStabilizationWindowSeconds: 120
        disableHpaScaleDown: "false"
```

```
    - coherenceMemberName: "httpgateway2"
      maxreplicas: 3
      averageCpuUtilization: 70
      averageMemoryUtilization: ""
      cpuLimit: 2000m
      cpuRequest: 1000m
      memoryLimit: 3Gi
      memoryRequest: 1Gi
      scaleDownStabilizationWindowSeconds: 120
      disableHpaScaleDown: "false"
```

5. To lower the heap memory used by the ECE pods, set the appropriate JVM garbage collection (GC) parameters in the **jvmGCOpts** key.

   Memory-based scale down occurs only if the amount of pod memory decreases. You can decrease pod memory by using JVM garbage collection (GC). For more information about JVM GC, see the "Java Garbage Collection Basics" tutorial.

6. Under the **ecs**, **ecs1**, and **httpgateway***n* sections, set the **replicas** key based on your configured Horizontal Pod Autoscaler values. For example, the number of replicas should meet the average resource consumption requirements you set in **averageMemoryUtilization**.

   This prevents the autoscaler from scaling down the ECE pods during the Helm upgrade, which could result in cache data loss.

7. Save and close your **override-values.yaml** file.

8. Enable Horizontal Pod Autoscaler in ECE by running the **helm upgrade** command for **oc-cn-ece-helm-chart**:

   ```
   helm upgrade EceReleaseName oc-cn-ece-helm-chart --namespace
   BrmNameSpace --values OverrideValuesFile
   ```

   where:

   - *EceReleaseName* is the release name for **oc-cn-ece-helm-chart** and is used to track this installation instance.
   - *BrmNameSpace* is the name space in which the BRM Kubernetes objects reside.
   - *OverrideValuesFile* is the path to the YAML file that overrides the default configurations in the **values.yaml** file.

Rolling Restart of ECE Pods

You can force a rolling restart of any ECE pod. If you restart a pod with multiple replicas, the pod replicas are restarted in reverse order. For example, if the ecs pod contains three replicas, the replicas are restarted in this order: 3, 2, 1.

To force a rolling restart of one of more ECE pods:

1. In your **override-values.yaml** file for **oc-cn-ece-helm-chart**, increment the appropriate pod's **restartCount** key by 1. For example, if the key was set to 3, you would increment it to 4.

   Table 24-1 lists the keys to use for restarting each ECE pod.

Table 24-1 Keys for Restarting ECE Pods

ECE Pod	Key
ecs	charging.ecs.restartCount
pricingupdater	charging.pricingupdater.restartCount
customerupdater	customerUpdater.customerUpdaterList.[*N*].restartCount[1]
emgateway	emgateway.emgatewayList.[*N*].restartCount[1]
diametergateway	diametergateway.diametergatewayList.[*N*].restartCount[1]
httpgateway	httpgateway.httpgatewayList.[*N*].restartCount[1]
brmgateway	brmgateway.brmgatewayList.[*N*].restartCount[1]
radiusgateway	radiusgateway.radiusgatewayList.[*N*].restartCount[1]
ratedeventformatter	ratedEventFormatter.ratedEventFormatterList.[*N*].restartCount[1]
monitoringagent	monitoringAgent.monitoringAgentList.[*N*].restartCount[1]

Notes:

(**1**) *N* represents the item block list, which is indicated by a dash (**–**) in the **override-values.yaml** file.

2. Perform a **helm upgrade** to update the Helm release:

```
helm upgrade EceReleaseName oc-cn-ece-helm-chart --values
OverrideValuesFile -n BrmNameSpace
```

25

Monitoring ECE in a Cloud Native Environment

Learn how to monitor system processes, such as memory and thread usage, of your Oracle Communications Elastic Charging Engine (ECE) components in a cloud native environment.

Topics in this document:

About Monitoring ECE in a Cloud Native Environment

You can set up monitoring of your ECE components in a cloud native environment. When configured to do so, ECE exposes JVM, Coherence, and application metric data through a single endpoint in an OpenMetrics/Prometheus exposition format. You can then use an external centralized metrics service, such as Prometheus, to scrape the ECE cloud native metrics and store them for analysis and monitoring.

ECE cloud native exposes metric data for the following components by default:

- ECE Server
- BRM Gateway
- Customer Updater
- Diameter Gateway
- EM Gateway
- HTTP Gateway
- CDR Formatter
- Pricing Updater
- Radius Gateway
- Rated Event Formatter

Setting up monitoring of these ECE cloud native components involves the following high-level tasks:

1. Ensuring that the ECE metric endpoints are enabled. See "Enabling ECE Metric Endpoints".

   ECE metric data will be exposed through the following endpoint: **http://localhost:19612/metrics**.

2. Setting up a centralized metrics service, such as Prometheus, to scrape metrics from the endpoint.

3. Setting up a visualization tool, such as Grafana, to display your ECE metric data in a graphical format.

Enabling ECE Metric Endpoints

The default ECE cloud native configuration exposes JVM, Coherence, and application metric data for all ECE components to a single REST endpoint. If you create any additional instances of ECE components, you must configure it to expose metric data.

To ensure that the ECE metric endpoints are enabled:

1. Open your **override-values.yaml** file for **oc-cn-ece-helm-chart**.
2. Verify that the **charging.metrics.port** key is set to the port number at which you want to expose the ECE metrics. The default is **19612**.
3. Verify that each ECE component instance has metrics enabled.

   Each application role under the **charging** key can be configured to enable or disable metrics. In the **jvmOpts** key, setting the **ece.metrics.http.service.enabled** option either enables (**true**) or disables (**false**) the metrics service for that role.

   For example, these **override-values.yaml** entries would enable the metrics service for **ecs1**.

```
charging:
   labels: "ece"
   jmxport: "9999"
   …
   metrics:
      port: "19612"
   ecs1:
      jmxport: ""
      replicas: 1
      …
      jvmOpts: "-Dece.metrics.http.service.enabled=true"
      restartCount: "0"
```

4. Save and close your **override-values.yaml** file.
5. Run the **helm upgrade** command to update your ECE Helm release:

```
helm upgrade EceReleaseName oc-cn-ece-helm-chart --namespace
EceNameSpace --values OverrideValuesFile
```

   where:

   - *EceReleaseName* is the release name for **oc-cn-ece-helm-chart**.
   - *EceNameSpace* is the name space in which to create ECE Kubernetes objects for the ECE Helm chart.
   - *OverrideValuesFile* is the name and location of your **override-values.yaml** file for **oc-cn-ece-helm-chart**.

ECE Cloud Native Metrics

ECE cloud native collects metrics in the following groups to produce data for monitoring your ECE components:

- JVM Metrics
- BRS Metrics
- Session Metrics
- Rated Events Metrics
- CDR Formatter Metrics
- Coherence Metrics

JVM Metrics

The JVM Metrics group contains standard metrics about the central processing unit (CPU) and memory utilization of JVMs, which are members of the ECE grid. Table 25-1 lists the metrics in this group.

Table 25-1 JVM Metrics

Metric Name	Type	Description
jvm_memory_bytes_init	Gauge	Contains the initial size, in bytes, for the Java heap and non-heap memory.
jvm_memory_bytes_committed	Gauge	Contains the committed size, in bytes, for the Java heap and non-heap memory.
jvm_memory_bytes_used	Gauge	Contains the amount of Java heap and non-heap memory, in bytes, that are in use.
jvm_memory_bytes_max	Gauge	Contains the maximum size, in bytes, for the Java heap and non-heap memory.
jvm_memory_pool_bytes_init	Gauge	Contains the initial size, in bytes, of the following JVM memory pools: **G1 Survivor Space**, **G1 Old Gen**, and **G1 Survivor Space**.
jvm_memory_pool_bytes_committed	Gauge	Contains the committed size, in bytes, of the following JVM memory pools: **G1 Survivor Space**, **G1 Old Gen**, and **G1 Survivor Space**.
jvm_memory_pool_bytes_used	Gauge	Contains the amount of Java memory space, in bytes, is in use by the following JVM memory pools: **G1 Survivor Space**, **G1 Old Gen**, and **G1 Survivor Space**.
jvm_buffer_count_buffers	Gauge	Contains the estimated number of mapped and direct buffers in the JVM memory pool.
jvm_buffer_total_capacity_bytes	Gauge	Contains the estimated total capacity, in bytes, of the mapped and direct buffers in the JVM memory pool.

Table 25-1 (Cont.) JVM Metrics

Metric Name	Type	Description
process_cpu_usage	Gauge	Contains the CPU usage information (in percentage) for each ECE component on the server. This data is collected from the corresponding MBean attributes by JVMs.
process_files_open_files	Gauge	Contains the total number of file-descriptors currently available for an ECE component and the descriptors that are in use for that ECE component.
coherence_os_system_cpu_load	Gauge	Contains the CPU load information (in percentage) for each system in the cluster. These statistics are based on the average data collected from all the ECE grid members running on a server.
system_load_average_1m	Gauge	Contains the system load average (the number of items waiting in the CPU run-queue) information for each machine in the cluster. These statistics are based on the average data collected from all the ECE grid members running on a server.
coherence_os_free_swap_space_size	Gauge	Contains system swap usage information (by default in megabytes) for each system in the cluster. These statistics are based on the average data collected from all the ECE grid members running on a server.

BRS Metrics

The BRS Metrics group contains the metrics for tracking throughput and latency of the charging clients that use batch request service (BRS). Table 25-2 lists the metrics in this group.

Table 25-2 ECE BRS Metrics

Metric Name	Metric Type	Description
ece_brs_task_processed	Counter	Tracks the total number of requests that have been accepted, processed, timed out, or rejected by the ECE component. You can use this to track the approximate processing rate over time, aggregate over all client applications, and so on.
ece_brs_task.pending_count	Gauge	Contains the number of requests that are pending by the ECE component.

Table 25-2 (Cont.) ECE BRS Metrics

Metric Name	Metric Type	Description
ece.brs.current.latency.by.type	Gauge	Tracks the latency of a charging client for each charging operation type in the current query interval. This metric provides the latency information for the following operation types: **Initiate**, **Update**, **Terminate**, **Cancel**, **Price_Enquiry**, **Balance_Query**, **Debit_Amount**, **Debit_Unit**, **Refund_Amount**, and **Refund_Unit**.
ece.brs.current.latency	Gauge	Tracks the current operation latency for a charging client in the current scrape interval. This metric contains the BRS statistics tracked using the **charging.brsConfigurations** MBean attributes. This configuration tracks maximum and average latency for an operation type since the last query. The maximum window size for the collecting this data is 30 seconds, so the query has to be run within every 30 seconds. This metric provides the latency information for the following operation types: **Initiate**, **Update**, **Terminate**, **Cancel**, **Price_Enquiry**, **Balance_Query**, **Debit_Amount**, **Debit_Unit**, **Refund_Amount**, **Refund_Unit**, and **Spending_Limit_Report**.

Session Metrics

The Session Metrics group contains metrics on ECE server sessions. Table 25-3 lists the metrics in this group.

Table 25-3 Session Metrics

Metric Name	Type	Description
ece_session_metrics	Counter	Contains the total number of sessions opened or closed by rating group, node, or cluster.

Rated Events Metrics

The Rated Events Metrics group contains metrics on rated events processed by ECE server sessions. Table 25-4 lists the metrics in this group.

Table 25-4 Rated Events Metrics

Metric Name	Type	Description
ece_rated_events_formatted	Counter	Contains the number of successful or failed formatted rated events per RatedEventFormatter worker thread upon each formatting job operation from NoSQL or the Oracle database.
ece_rated_events_cached	Counter	Contains the total number of rated events cached by each ECE node.
ece_rated_events_inserted	Counter	Contains the total number of rated events that were successfully inserted into the cache.
ece_rated_events_insert_failed	Counter	Contains the total number of rated events that failed to be inserted into the cache.
ece_rated_events_purged	Counter	Contains the total number of rated events that were purged.
ece_requests_by_result_code	Counter	Tracks the total requests processed by using the result code.

CDR Formatter Metrics

The CDR Formatter Metrics group contains the metrics for tracking Charging Function (CHF) records. Table 25-5 lists the metrics in this group.

Table 25-5 CDR Formatter Metrics

Metric Name	Metric Type	Description
ece_chf_records_processed	Counter	Tracks the total number of CHF records that have been processed by the CDR formatter.
ece_chf_records_purged	Counter	Tracks the total number of CHF records that have been purged by the CDR formatter.
ece_chf_records_loaded	Counter	Tracks the total number of CHF records that have been loaded by the CDR formatter.

Coherence Metrics

All Coherence metrics that are available through the Coherence metrics endpoint are also accessible through the ECE metrics endpoint. For more information about the Coherence metrics, see "Oracle Coherence MBeans Reference" in *Oracle Fusion Middleware Managing Oracle Coherence*.

For information about querying for Coherence metrics, see "Querying for Coherence Metrics" in *Oracle Fusion Middleware Managing Oracle Coherence*.

A

WebLogic-Based Application Metrics

This appendix lists the WebLogic-based application metrics that are supported by the Oracle Communications Billing and Revenue Management (BRM) cloud native deployment.

WebLogic Monitoring Exporter collects metrics in the following groups to produce data for monitoring Pricing Design Center (PDC), Business Operations Center, Billing Care, and Billing Care REST API in a cloud native environment:

- WLS Server Metrics Group
- Application Runtime Metric Group
- Servlets Metric Group
- JVM Runtime Metric Group
- Execute Queue Runtimes Metric Group
- Work Manager Runtimes Metric Group
- Thread Pool Runtime Metric Group
- JDBC Service Runtime Metric Group
- JTA Runtime Metric Group
- WLS Scrape MBean Metric Group
- Persistent Store Runtime MBean Metric Group

WLS Server Metrics Group

Use the WLS server metrics group to retrieve runtime information about a server instance and to transition a server from one state to another. Table A-1 lists the metrics in this group.

Table A-1 WLS Server Metrics

Metric Name	Label	Metric Type	Description
wls_server_activation_time	location	long	Returns the time when the server was started.
wls_server_admin_server_listen_port	location	int	Returns the port on which this server is listening for requests.
wls_server_open_sockets_current_count	location	int	Returns the current number of sockets registered for socket muxing on this server.
wls_server_state_val	location	int	Returns the current state of the server as an integer: • **0**: Shutdown • **1**: Starting • **2**: Running

Application Runtime Metric Group

Use the application runtime metric group to collect runtime information about a deployed enterprise application. Table A-2 describes the metrics in the group.

Table A-2 Application Runtime Metrics

Metric Name	Label	Metric Type	Description
wls_webapp_config_deployment_state	location app name	int	Returns the current state of the deployment as an integer.
wls_webapp_config_open_sessions_current_count	location app name	int	Returns the current number of open sessions in this module.
wls_webapp_config_open_sessions_high_count	location app name	int	Returns the highest number of open sessions on this server at any one time.
wls_webapp_config_sessions_opened_total_count	location app name	int	Returns the total number of sessions that were opened.

Servlets Metric Group

Each WAR file can contain multiple servlets, and each WAR file can be integrated into an enterprise archive (EAR). Use the servlets metric group to obtain runtime information about a web application and each servlet. Table A-3 describes the metrics in this group.

Table A-3 Servlets Metrics

Metric Name	Label	Metric Type	Description
wls_servlet_execution_time_average	location app name servlet Name	long	Displays the average amount of time, in milliseconds, it took to run all invocations of the servlet since it was most recently deployed.
wls_servlet_execution_time_high	location app name servlet Name	long	Displays the average amount of time, in milliseconds, that the single longest invocation of the servlet has run since it was most recently deployed.

Table A-3 (Cont.) Servlets Metrics

Metric Name	Label	Metric Type	Description
wls_servlet_execution_time_low	location app name servlet Name	long	Displays the average amount of time, in milliseconds, that the single shortest invocation of the servlet has run since it was most recently deployed.
wls_servlet_execution_time_total	location app name servlet Name	long	Displays the average amount of time, in milliseconds, that all invocations of the servlet have run since it was most recently deployed.
wls_servlet_invocation_total_count	location app name servlet Name	int	Displays the total number of times the servlet has been invoked since WebLogic Server started.
wls_servlet_pool_max_capacity	location app name servlet Name	int	Displays the maximum capacity of this servlet for single thread model servlets.
wls_servlet_reload_total_count	location app name servlet Name	int	Displays the total number of times WebLogic Server has reloaded the servlet since it was last deployed. WebLogic Server typically reloads a servlet if it has been modified.

JVM Runtime Metric Group

Use the JVM runtime metric group to retrieve information about the Java Virtual Machine (JVM) that the current server instance is running. Table A-4 describes the metrics in this group.

Table A-4 JVM Runtime Metrics

Metric Name	Labels	Metric Type	Description
wls_jvm_heap_free_current	name	long	Returns the current amount of memory, in bytes, that is available in the JVM heap.
wls_jvm_heap_free_percent	name	int	Returns the percentage of the JVM heap that is free.
wls_jvm_heap_size_current	name	long	Returns the current size, in bytes, of the JVM heap.
wls_jvm_heap_size_max	name	long	Returns the maximum size, in bytes, of the JVM heap.

Table A-4 (Cont.) JVM Runtime Metrics

Metric Name	Labels	Metric Type	Description
wls_jvm_process_cpu_load	name	time	Returns the amount of CPU time that the Java virtual machine is running in nanoseconds.
wls_jvm_uptime	name	long	Returns the number of milliseconds that the virtual machine has been running.

Execute Queue Runtimes Metric Group

Use the execute queue runtime metric group to return information about the queue. Table A-5 describes the metrics in this group.

Table A-5 Execute Queue Runtimes Metrics

Metric Name	Labels	Metric Type	Description
wls_socketmuxer_pending_request_current_count	name	int	Returns the number of waiting requests in the queue.

Work Manager Runtimes Metric Group

Use the work manager runtimes metric group to retrieve information about requests from the work manager. Table A-6 describes the metrics in this group.

Table A-6 Work Manager Runtimes Metrics

Metric Name	Label	Metric Type	Description
wls_workmanager_completed_requests	name	int	Returns the number of requests that have been processed.
wls_workmanager_pending_requests	name	int	Returns the number of waiting requests in the queue.
wls_workmanager_stuck_thread_count	name	int	Returns the number of stuck threads in the thread pool.

Thread Pool Runtime Metric Group

Use the thread pool runtime metric group to monitor the self-tuning queue. Table A-7 describes the metrics in this group.

Table A-7 Thread Pool Runtime Metrics

Metric Name	Label	Metric Type	Description
wls_threadpool_execute_thread_total_count	name	int	Returns the total number of threads in the pool.
wls_threadpool_hogging_thread_count	name	int	Returns the threads that are currently being held by a request. These threads will either be declared as stuck after the configured timeout period or be returned to the pool.
wls_threadpool_queue_length	name	int	Returns the number of pending requests in the priority queue.
wls_threadpool_stuck_thread_count	name	int	Returns the number of stuck threads in the thread pool.

JDBC Service Runtime Metric Group

Use the JDBC service runtime metric group to retrieve runtime information about a server instance and to transition a server from one state to another. Table A-8 describes the metrics in this group.

Table A-8 JDBC Service Runtime Metrics

Metric Name	Label	Metric Type	Description
wls_datasource_active_connections_average_count	name	int	Returns the average number of active connections in this data source instance.
wls_datasource_active_connections_current_count	name	int	Returns the number of connections currently in use by applications.
wls_datasource_active_connections_high_count	name	int	Returns the highest number of active database connections in this data source instance since the data source was instantiated.
wls_datasource_commit_outcome_retry_total_count	name	int	Returns the cumulative total number of commit outcome query retries conducted before resolving the outcome or exceeding the retry seconds in this data source since the data source was deployed.
wls_datasource_connection_delay_time	name	int	Returns the average amount of time, in milliseconds, that it takes to create a physical connection to the database.
wls_datasource_connections_total_count	name	int	Returns the cumulative total number of database connections created in this data source since the data source was deployed.
wls_datasource_curr_capacity_high_count	name	int	Returns the highest number of database connections available or in use (current capacity) in this data source instance since the data source was deployed.
wls_datasource_curr_capacity	name	int	Returns the current count of JDBC connections in the data source's connection pool.

Table A-8 (Cont.) JDBC Service Runtime Metrics

Metric Name	Label	Metric Type	Description
wls_datasource_deployment_state	name	int	Returns the module's current deployment state.
wls_datasource_failed_repurpose_count	name	int	Returns the number of repurpose errors that have occurred since the data source was deployed.
wls_datasource_failed_reserve_request_count	name	int	Returns the cumulative running count of connection requests from this data source that could not be fulfilled.
wls_datasource_failures_to_reconnect_count	name	int	Returns the number of times that the data source attempted to refresh a database connection and failed.
wls_datasource_highest_num_available	name	int	Returns the highest number of database connections that were idle and available to be used by an application at any time in this data source instance since the data source was deployed.
wls_datasource_highest_num_unavailable	name	int	Returns the highest number of database connections that were in use by applications or being tested by the system in this data source instance since the data source was deployed.
wls_datasource_leaked_connection_count	name	int	Returns the number of leaked connections.
wls_datasource_num_available	name	int	Returns the number of database connections that are currently idle and available to be used by applications in this data source instance.
wls_datasource_num_unavailable	name	int	Returns the number of connections currently in use by applications or being tested in this data source instance.
wls_datasource_prep_stmt_cache_access_count	name	long	Returns the cumulative, running count of the number of times that the statement cache was accessed.
wls_datasource_prep_stmt_cache_add_count	name	long	Returns the cumulative, running count of the number of statements added to the statement cache.
wls_datasource_prep_stmt_cache_current_size	name	int	Returns the number of prepared and callable statements currently cached in the statement cache.
wls_datasource_prep_stmt_cache_delete_count	name	long	Returns the cumulative, running count of statements discarded from the cache.
wls_datasource_prep_stmt_cache_hit_count	name	long	Returns the cumulative, running count of the number of times that statements from the cache were used.
wls_datasource_prep_stmt_cache_miss_count	name	long	Returns the number of times that a statement request could not be satisfied with a statement from the cache.
wls_datasource_reserve_request_count	name	long	Returns the cumulative, running count of connection requests from this data source.

Table A-8 (Cont.) JDBC Service Runtime Metrics

Metric Name	Label	Metric Type	Description
wls_datasource_waiting_for_connection_current_count	name	int	Returns the number of connection requests waiting for a database connection.
wls_datasource_waiting_for_connection_failure_total	name	long	Returns the cumulative, running count of connection requests from this data source that had to wait before getting a connection and eventually failed to get a connection.
wls_datasource_waiting_for_connection_high_count	name	int	Returns the highest number of application requests concurrently waiting for a connection from this data source instance.
wls_datasource_waiting_for_connection_success_total	name	long	Returns the cumulative, running count of connection requests from this data source that had to wait before getting a successful connection.
wls_datasource_waiting_for_connection_total	name	long	Returns the cumulative, running count of connection requests from this data source that had to wait before getting a connection. This includes requests that eventually got a connection and those that did not get a connection.

JTA Runtime Metric Group

Use the JTA runtime metric group to access transaction runtime characteristics within a WebLogic Server. Table A-9 describes the metrics in this group.

Table A-9 JTA Runtime Metrics

Metric Name	Label	Metric Type	Description
wls_jta_active_transactions_total_count	name	long	Returns the number of active transactions on the server.
wls_jta_seconds_active_total_count	name	int	Returns the total number of seconds that transactions were active for all committed transactions.
wls_jta_transaction_abandoned_total_count	name	long	Returns the total number of transactions that were abandoned since the server was started.
wls_jta_transaction_committed_total_count	name	long	Returns the total number of transactions committed since the server was started.
wls_jta_transaction_heuristics_total_count	name	long	Returns the number of transactions that completed with a heuristic status since the server was started.
wls_jta_transaction_llrcommitted_total_count	name	long	Returns the total number of LLR transactions that were committed since the server was started.
wls_jta_transaction_no_resources_committed_total_count	name	long	Returns the total number of transactions with no enlisted resources that were committed since the server was started.

Table A-9 (Cont.) JTA Runtime Metrics

Metric Name	Label	Metric Type	Description
wls_jta_transaction_one_resource_one_phase_committed_total_count	name	long	Returns the total number of transactions with more than one enlisted resource that were one-phase committed due to read-only optimization since the server was started.
wls_jta_transaction_total_count	name	long	Returns the total number of transactions processed. This total includes all committed, rolled back, and heuristic transaction completions since the server was started.

WLS Scrape MBean Metric Group

Use the WLS scrape metric group to monitor the performance of the WebLogic Server. Table A-10 describes the metrics in this group.

Table A-10 WLS Scrape MBean Metrics

Metric Name	Label	Metric Type	Description
wls_scrape_mbeans_count_total	instance	long	Returns the number of metrics scraped.
wls_scrape_duration_seconds	instance	long	Returns the time required to do the scrape.
wls_scrape_cpu_seconds	instance	long	Returns the amount of time the CPU used during the scrape.

Persistent Store Runtime MBean Metric Group

Use the persistent store runtime MBean metric group to monitor a persistent store. Table A-11 describes the metrics in this group.

Table A-11 Persistent Store Runtime MBean Metrics

Metric Name	Label	Metric Type	Description
wls_persistentstore_allocated_io_buffer_bytes	name	long	Returns the amount of off-heap (native) memory, in bytes, reserved for file store use. When applicable, this is a multiple of the file store configurable attribute IOBufferSize. This applies to synchronous write policies Direct-Write and Cache-Flush policies.

Table A-11 (Cont.) Persistent Store Runtime MBean Metrics

Metric Name	Label	Metric Type	Description
wls_persistentstore_allocated_window_buffer_bytes	name	long	Returns the amount of off-heap (native) memory, in bytes, reserved for file store window buffer use. Applies to synchronous write policies Direct-Write-With-Cache and Disabled, but only when the native wlfileio library is loaded.
wls_persistentstore_create_count	name	long	Returns the number of create requests issued by this store.
wls_persistentstore_delete_count	name	long	Returns the number of delete requests issued by this store.
wls_persistentstore_object_count	name	int	Returns the number of objects contained in the connection.
wls_persistentstore_physical_write_count	name	long	Returns the number of times the store flushed its data to durable storage.
wls_persistentstore_read_count	name	long	Returns the number of read requests issued by this store, including requests that occur during store initialization.
wls_persistentstore_update_count	name	long	Returns the number of update requests issued by this store.

B Supported Scripts and Utilities

This appendix lists the scripts and utilities that are supported out-of-the-box by the Oracle Communications Billing and Revenue Management (BRM) cloud native deployment.

You configure and run these scripts and utilities by editing your **override-values.yaml** file and then updating the Helm release:

- **ImportExportPricing**
- **pin_multidb**
- **pin_virtual_time**
- **syncPDC**

C

Supported Utilities and Applications for brm-apps Jobs

This appendix lists the utilities and applications that are supported by the brm-apps job in your Oracle Communications Billing and Revenue Management (BRM) cloud native deployment.

The brm-apps job facilitates the running of utilities and applications on demand without entering a pod.

Table C-1 lists the applications that can be run by the brm-apps job.

Table C-1 Supported Applications

Directory	Application
apps/pin_inv_doc_gen	pin_inv_doc_gen
apps/pin_amt	pin_amt pin_amt_install.pl
apps/telco	RunSimulator
apps/pin_aq	pin_portal_sync_oracle.pl
apps/pin_billd	pin_bill_day pin_bill_accts pin_deposit pin_mass_refund pin_refund pin_deferred_act pin_ledger_report pin_recycle
apps/pin_collections	pin_collect pin_collections_process pin_collections_send_dunning
apps/load_channel_config	pin_channel_export
apps/pin_trial_bill	pin_trial_bill_accts
apps/pin_ifw_sync	pin_ifw_sync_oracle.pl
apps/pin_monitor	pin_monitor_balance
apps/pin_bulk_adjust	pin_apply_bulk_adjustment
apps/partition_utils	partition_utils
apps/pin_sepa	pin_sepa
apps/pin_ra_check_thresholds	pin_ra_check_thresholds
apps/pin_event_extract	pin_event_extract
apps/pin_rerate	pin_rerate
apps/integrate_sync	pin_history_on

Table C-1 (Cont.) Supported Applications

Directory	Application
apps/storable_class_to_xml	storableclasstoxml
apps/load_config	load_config
apps/pin_inv	pin_inv_send pin_inv_export pin_inv_accts
apps/pin_remit	pin_remittance pin_remit_month
apps/pin_export_price	pin_export_price
apps/load_price_list	loadpricelist
apps/cmt	pin_cmt
apps/partition	partitioning.pl

D
Supported Load Utilities for Configurator Jobs

This appendix lists the load utilities that are supported by the configurator job in your Oracle Communications Billing and Revenue Management (BRM) cloud native deployment.

The configurator job facilitates the running of load utilities on demand without entering into a pod. You can use the configurator job to run these load utilities:

- **load_ara_config_object**
- **load_channel_config**
- **load_config_dist**
- **load_config_item_tags**
- **load_config_item_types**
- **load_config_provisioning_tags**
- **load_content_srvc_profiles**
- **load_edr_field_mapping**
- **load_event_map**
- **load_localized_strings**
- **load_pin_ach**
- **load_pin_ar_taxes**
- **load_pin_batch_suspense_override_reason**
- **load_pin_batch_suspense_reason_code**
- **load_pin_beid**
- **load_pin_billing_segment**
- **load_pin_bill_suppression**
- **load_pin_business_profile**
- **load_pin_calendar**
- **load_pin_config_auth_reauth_info**
- **load_pin_config_batchstat_link**
- **load_pin_config_business_type**
- **load_pin_config_controlpoint_link**
- **load_pin_config_export_gl**
- **load_pin_config_ood_criteria**
- **load_pin_config_ra_alerts**
- **load_pin_config_ra_flows**
- **load_pin_config_ra_thresholds**

- **load_pin_customer_segment**
- **load_pin_dealers**
- **load_pin_device_permit_map**
- **load_pin_device_state**
- **load_pin_event_record_map**
- **load_pin_excluded_logins**
- **load_pin_impact_category**
- **load_pin_glchartaccts**
- **load_pin_glid**
- **load_pin_invoice_data_map**
- **load_pin_network_elements**
- **load_pin_notify**
- **load_pin_num_config**
- **load_pin_order_state**
- **load_pin_payment_term**
- **load_pin_recharge_card_type**
- **load_pin_remittance_flds**
- **load_pin_remittance_spec**
- **load_pin_rerate_flds**
- **load_pin_rtp_trim_flist**
- **load_pin_rum**
- **load_pin_service_framework_permitted_service_types**
- **load_pin_sim_config**
- **load_pin_snowball_distribution**
- **load_pin_spec_rates**
- **load_pin_sub_bal_contributor**
- **load_pin_suspense_editable_flds**
- **load_pin_suspense_edr_fld_map**
- **load_pin_suspense_override_reason**
- **load_pin_suspense_params**
- **load_pin_suspense_reason_code**
- **load_pin_telco_provisioning**
- **load_pin_telco_service_order_state**
- **load_pin_telco_tags**
- **load_pin_uniqueness**
- **load_pin_verify**
- **load_pin_voucher_config**

- **load_suspended_batch_info**
- **load_tax_supplier**
- **load_transition_type**
- **load_usage_map**
- **pin_bus_params**
- **pin_deploy**
- **pin_load_invoice_events**
- **pin_uei_deploy**
- **testnap**