Oracle® Communications Billing and Revenue Management

Cloud Native System Administrator's Guide



Release 16.26.0
G54447-01
October 2026

ORACLE

Oracle Communications Billing and Revenue Management Cloud Native System Administrator's Guide, Release 16.26.0

G54447-01

Copyright © 2021, 2026, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

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.

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

Contents

About This Content

Part I Basic System Administration of BRM Cloud Native

1 Managing Pods and PVCs in BRM Cloud Native

Setting up Autoscaling of BRM Pods 1
Automatically Rolling Deployments by Using Annotations 3
Restarting BRM Pods 3
Setting Minimum and Maximum CPU and Memory Values 4
Using Static Volumes 5
Assigning Pods to Nodes Using nodeSelector and affinity 6
Using nodeSelector 6
Using Affinity 7
Managing Labels and Annotations 7
Example Resource Labels and Annotations 8
Example Kind Labels and Annotations 9
Example Global Labels and Annotations 9
About Customizing and Extending Pods 9
Example Specifying Node Tolerations 10
Example Distributing Pods Across Nodes 10

2 Running Applications and Utilities Outside Pods

Running Load Utilities Through Configurator Jobs 1
Running pin_bus_params and load_pin_device_state 2
Running Load Utilities on Multischema Systems 3
Running Applications and Utilities Through brm-apps Jobs 4
Configuring MTA Performance Parameters 4
Running Custom Applications and Utilities Through brm-apps 6
Running Business Operations Through pin_job_executor Service 7

Copyright © 2021, 2026, Oracle and/or its affiliates.

3 Exposing Directories as ConfigMaps

Configuring a CM ConfigMap Directory 1
Exposing the taxcode_map File Example 1
Configuring an EAI Publisher ConfigMap 2

4 Managing a Helm Release

About Helm Releases 1
Tracking a Release's Status 1
Updating a Helm Release 1
Checking a Release's Revision 2
Rolling Back a Release To a Previous Revision 2

5 Managing Passwords in BRM Cloud Native

Using External Kubernetes Secrets for Certificates and Wallets 1
Rotating the BRM Root Password 2
Rotating the BRM Root Key 5
Rotating the BRM Password 6
Rotating BRM Role Passwords 6

6 Managing Database Partitions

Converting Nonpartitioned Classes to Partitioned Classes 1
Adding Partitions to Your BRM Cloud Native Database 2
Dropping Partitions from Your BRM Cloud Native Database 2

7 Improving Performance in BRM Cloud Native

Deploying the CM and DM Containers in the Same Pod 1
Tuning Your Application Connection Pools 2
Configuring Application Connection Pools with Multiple CM Replicas 3
Configuring Multiple Replicas of Batch Controller 3
Deploying Paymentech Data Manager in HA Mode 4

8 Managing a BRM Cloud Native Multischema System

Running Billing Against a Specified Schema 1
Adding Schemas to a Multischema System 2
Migrating Accounts from One Schema to Another 7
Migrating Accounts Using Custom Search Criteria 10
Modifying Database Schema Priorities 12

Copyright © 2021, 2026, Oracle and/or its affiliates.

Modifying Database Schema Status 13
Synchronizing /uniqueness Objects Between Schemas 15

9 Migrating Legacy Data to BRM Cloud Native

About Migrating Legacy Data 1
Loading Legacy Data into the BRM Database 1

10 Creating Custom Fields and Storable Classes

Creating, Editing, and Deleting Fields and Storable Classes using BRM SDK Opcodes 1
Making the Data Dictionary Writable 1
Running the BRM SDK Opcodes 2
Making Custom Fields Available to Your PCM and C++ Applications 5
Making Custom Fields Available to Your Java PCM Applications 7
Moving Field and Storable Class Definitions Between BRM Servers with pin_deploy 7
Extracting Field and Storable Class Definitions with pin_deploy 8
Importing Field and Storable Class Definitions with pin_deploy 9

11 Using Logs to Monitor Components

Overview of BRM Cloud Native Logging 1
Viewing Component Logs 2
Configuring Log Collection and Forwarding 2
Using the Sample Fluentd Configuration Files 2
Configuring the Log Management System 3
Setting the Reporting Level for Logging Messages 3

Part II Monitoring BRM Cloud Native Services

12 Monitoring BRM Cloud Native Services

About Monitoring BRM Cloud Native Services 1
Setting Up Monitoring for BRM Cloud Native Services 1
Configuring BRM Cloud Native to Collect Metrics 2
Configuring Perflib for BRM Opcode Monitoring 4
Configuring OracleDB_Exporter to Scrape Database Metrics 7
Configuring Grafana for BRM Cloud Native 8
BRM Opcode Metric Group 9

Copyright © 2021, 2026, Oracle and/or its affiliates.

13 Monitoring and Autoscaling Business Operations Center Cloud Native

About Monitoring and Autoscaling in Business Operations Center Cloud Native 1
Setting Up Monitoring and Autoscaling in Business Operations Center 1
- Configuring WebLogic Monitoring Exporter to Scrape Metric Data 2
- Configuring webhook to Enable Autoscaling 3
- Configuring Standalone Prometheus for Business Operations Center 5
- Configuring Prometheus Operator for Business Operations Center 7
- Creating Grafana Dashboards for Business Operations Center 9

Sample Prometheus Alert Rules for Business Operations Center 9

14 Monitoring and Autoscaling Billing Care Cloud Native

About Monitoring and Autoscaling in Billing Care Cloud Native 1
Setting Up Monitoring and Autoscaling in Billing Care and Billing Care REST API 1
- Configuring WebLogic Monitoring Exporter to Scrape Metric Data 2
- Configuring Webhook to Enable Autoscaling 3
- Configuring Standalone Prometheus for Billing Care 6
- Configuring Prometheus Operator for Billing Care 9
- Creating Grafana Dashboards for Billing Care and Billing Care REST API 11

Sample Prometheus Alert Rules for Billing Care and Billing Care REST API 11

15 Monitoring and Autoscaling Pipeline Configuration Center Cloud Native

About Monitoring and Autoscaling Pipeline Configuration Center Cloud Native 1
Setting Up Monitoring and Autoscaling in Pipeline Configuration Center 1
- Configuring WebLogic Monitoring Exporter to Scrape Metric Data 2
- Configuring webhook to Enable Autoscaling 3
- Configuring Standalone Prometheus for Pipeline Configuration Center 5
- Configuring Prometheus Operator for Pipeline Configuration Center 7
- Creating Grafana Dashboards for Pipeline Configuration Center 9

Sample Prometheus Alert Rules for Pipeline Configuration Center 9

16 Monitoring BRM REST Services Manager Cloud Native

About Monitoring BRM REST Services Manager Cloud Native 1
Setting Up Monitoring for BRM REST Services Manager 1
Creating Grafana Dashboards for BRM REST Services Manager 4
Modifying Prometheus and Grafana Alert Rules After Deployment 5
About REST Endpoints for Monitoring BRM REST Services Manager 5

Copyright © 2021, 2026, Oracle and/or its affiliates.

17 Tracing BRM REST Services Manager Cloud Native

About BRM REST Services Manager Tracing 1
Securing Communication with Zipkin 1
Enabling Tracing in BRM REST Services Manager 1

18 Tracing Opcode Processes

Enabling Selective Opcode Tracing 1

Part III Integrating with BRM Cloud Native

19 Integrating with Your BRM Cloud Native Deployment

Integrating with Thick Clients 1
Configuring TLS Certificates for BRM Cloud Native 2
How BRM Selects the TLS Wallet 2
Preparing the Required TLS Wallets 3
Configuring TLS Certificates 3
Configuring BRM Client Applications for Two-Way TLS 5
Configuring an External Load Balancer for Connection Manager SSL/TLS Connections 5
Integrating with JCA Resource Adapter 6
Integrating with Kafka Servers 8
Integrating with Oracle Analytics Publisher 10
Generating Invoices in Oracle Analytics Publisher 14

20 Setting Up Payment Processing with Paymentech

About Setting Up Payment Processing with Paymentech 1
Using SFTP for Batch Payment Transactions 1
Configuring Authentication for SFTP Batch Transactions 2
Configuring dm_fusa with SFTP Settings 2
Using TCP/IP for Batch Payment Transactions 4
Configuring TCP/IP for Online Payment Transactions 4
Configuring the Paymentech Simulator for Testing 5

Part IV Administering PDC Cloud Native Services

Copyright © 2021, 2026, Oracle and/or its affiliates.

21 Administering PDC Cloud Native Services

Creating PDC Users 1
Using Resource Limits in PDC Domain Pods 2

22 Running PDC Applications

About Running the PDC Utilities 1
Importing Pricing and Setup Components with ImportExportPricing 1
- Importing from a Single XML File 1
- Importing Multiple XML Files from a Directory 3

Exporting Pricing and Setup Components with ImportExportPricing 4
Checking the Status of a Changeset with ImportExportPricing 6
Using SyncPDC to Synchronize Setup Components 8

23 Monitoring PDC in a Cloud Native Environment

About Monitoring PDC Cloud Native 1
Setting Up Monitoring in PDC Cloud Native 1
- Creating Grafana Dashboards for Pricing Design Center 2

24 Monitoring PDC REST Services Manager

About PDC REST Services Manager Logs 1
- Accessing the PDC REST Services Manager Logs 2
- Changing the Log Levels 3
  - Changing the Log Levels Using Helm 3
  - Changing the Log Levels Using Kubernetes 4
- Changing the Default Log Manager Using Helm 4

About PDC REST Services Manager Tracing 5
- Enabling Tracing in PDC REST Services Manager 5
- Using Trace Tags to Troubleshoot Issues 6

About PDC REST Services Manager Metrics 7
- Checking Access to PDC REST Services Manager Metrics 8

About Monitoring PDC REST Services Manager System Health 9
- Verifying the PDC REST Services Manager Pod Status 9
- Using the PDC REST Services Manager Health Endpoint 9

25 Rotating PDC Log Files

About Rotating PDC Log Files 1

Copyright © 2021, 2026, Oracle and/or its affiliates.

26 Managing Language Packs in PDC Pods

Enabling Language Packs in PDC Pods 1

Part V Administering ECE Cloud Native Services

27 Administering ECE Cloud Native Services

Running SDK Jobs 1
Error Handling for SDK Jobs 3
Changing the ECE Configuration During Runtime 4
Creating a JMX Connection to ECE Using JConsole 4
Reloading ECE Application Configuration Changes 5
Reloading the Grid Log Level 6
Rotating Kafka and JMS Credentials at Runtime 7
JMS Credential Rotation Considerations 7
Kafka Credential Rotation Considerations 7
Monitoring Credential Refresh 7
Configuring Subscriber-Based Tracing for ECE Services 8
Using Third-Party Libraries and Custom Mediation Specifications 10
Loading RADIUS Network Mapping Files in ECE Cloud Native 11
Setting Up ECE Cloud Native in Firewall-Enabled Environments 12
Enabling Federation in ECE 14
Enabling Parallel Pod Management in ECE 16
Customizing SDK Source Code 17

28 Securing ECE Communications

Enabling SSL Communication When BRM and ECE Run in Separate Clusters 1
Using Customer-Provided TLS Certificates for BRM Communication 3
Securing Communication Between the CHF and NRF, PCF, and SMF 4
Securing Communication Using KeyStores Mounted in the Helm Chart 4
Securing Communication Using External Kubernetes Secrets 5

29 Managing ECE Journal Storage

About Coherence Elastic Data Storage 1
Managing Coherence Journal Space 1

Copyright © 2021, 2026, Oracle and/or its affiliates.

30 Managing Persisted Data in the Oracle Database

Enabling Persistence in ECE 1
Rotating Persistence Database Passwords 3
Avoiding Missing Pricing Data During ECE Cold Restarts 4
Re-Creating the ECE Schema After Deployment 6
Loading Only Partial Data into ECE Cache 6
Incremental Customer Loading in ECE Cache 6

31 Configuring Disaster Recovery in ECE Cloud Native

Setting Up Active-Active Disaster Recovery for ECE 1
Processing CDR Generation in an Active-Active System 11
Processing Usage Requests on Site Receiving Request 11
Stopping ECE from Routing to a Failed Site 12
Adding Fixed Site Back to ECE System 12
Activating a Secondary Rated Event Formatter Instance 13
About Conflict Resolution During the Journal Federation Process 14
Using startWithSync to Resynchronize an ECE’s Cache Contents 15

32 Managing ECE Pods

Scaling Kubernetes Pods 1
Setting Up Autoscaling of ECE Pods 1
Rolling Restart of ECE Pods 4

33 Monitoring ECE in a Cloud Native Environment

About Monitoring ECE in a Cloud Native Environment 1
Setting Up Alerts with the ECE Alert Configuration Template 2
Modifying Alert Rules and Thresholds 3
Enabling ECE Metric Endpoints 4
Best Practices and Important Notes 5
Sample Prometheus Operator Configuration 5
ECE Cloud Native Metrics 7
BRS Metrics 7
BRS Queue Metrics 10
CDR Formatter Metrics 11
Coherence Metrics 11
Coherence Federated Service Metrics 11
Diameter Gateway Metrics 12
ECE Federated Service Metrics 13
ECE Notification Metrics 13

Copyright © 2021, 2026, Oracle and/or its affiliates.

EM Gateway Metrics 14
HTTP Gateway Metrics 14
JVM Metrics 15
Kafka JMX Metrics 16
Kafka Client Metrics 16
Micrometer Executor Metrics 17
RADIUS Gateway Metrics 18
Rated Event Formatter (REF) Metrics 18
Rated Events Metrics 19
Session Metrics 19
ECE Cloud Native Alerts 19
CDR Formatter Alerts 20
CDR Gateway Alerts 20
Coherence Alerts 20
Diameter Gateway Alerts 22
EM Gateway Alerts 23
HTTP Gateway Alerts 23
JVM Alerts 23
Kubernetes Alerts 24
Rated Event Formatter (REF) Alerts 24

Part VI Troubleshooting BRM Cloud Native Services

34 Troubleshooting BRM Cloud Native

Authorization Error When Running pin_update_balances Utility 1
Order Processing Fails with Kafka DM Connection Errors 2
Grafana Dashboards are Empty 3
SSL/TLS Handshake Error When Connecting to CM Through an External Load Balancer 4

35 Troubleshooting ECE Cloud Native

Troubleshooting Incorrect Charges for Mid-Session Product Purchase 1
Configuring Tariff Split Behavior During Installation or Before Restart 2
Configuring Tariff Split Behavior at Runtime 2
Troubleshooting Missing Login Identifier for Existing Accounts After Enabling ECE 2
Configuring Login Identifier Loading During Installation or Before Restart 3
Configuring Login Identifier Loading at Runtime 3
Troubleshooting Long Cycle Grant Validity Splitting 3
Configuring Long Cycle Validity Splitting During Installation or Before Restart 4
Configuring Long Cycle Validity Splitting at Runtime 5

Copyright © 2021, 2026, Oracle and/or its affiliates.

Reloading the Charging Configuration Reload Fails Due to Missing Cluster Port 5

36 Troubleshooting PDC Cloud Native

Troubleshooting ImportExportPricing Errors 1
Unable to Import a Large Number of XML Files 1
SyncPDC Fails After Cloning BRM and PDC Databases 2
Publishing Pricing Components to ECE Fails 4
RRE Transformation Fails After Importing Pricing Objects 5

A WebLogic-Based Application Metrics

WLS Server Metrics Group A-1
Application Runtime Metric Group A-2
Servlets Metric Group A-2
JVM Runtime Metric Group A-3
Execute Queue Runtime Metric Group A-3
Work Manager Runtime Metric Group A-4
Thread Pool Runtime Metric Group A-4
JDBC Service Runtime Metric Group A-4
JTA Runtime Metric Group A-6
WLS Scrape MBean Metric Group A-7
Persistent Store Runtime MBean Metric Group A-7

B Supported Scripts and Utilities

C Supported Utilities and Applications for brm-apps Jobs

D Supported Load Utilities for Configurator Jobs

Copyright © 2021, 2026, Oracle and/or its affiliates.

About This Content

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 those involved in installing and maintaining an BRM Cloud Native Deployment.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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:

- Managing Pods and PVCs in BRM Cloud Native
- Running Applications and Utilities Outside Pods
- Exposing Directories as ConfigMaps
- Managing a Helm Release
- Managing Passwords in BRM Cloud Native
- Managing Database Partitions
- Improving Performance in BRM Cloud Native
- Managing a BRM Cloud Native Multischema System
- Migrating Legacy Data to BRM Cloud Native
- Creating Custom Fields and Storable Classes
- Using Logs to Monitor Components

Copyright © 2021, 2026, Oracle and/or its affiliates.

1

Managing Pods and PVCs in BRM Cloud Native

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

Topics in this document:

- Setting up Autoscaling of BRM Pods
- Automatically Rolling Deployments by Using Annotations
- Restarting BRM Pods
- Setting Minimum and Maximum CPU and Memory Values
- Using Static Volumes
- Assigning Pods to Nodes Using nodeSelector and affinity
- Managing Labels and Annotations
- About Customizing and Extending Pods

> **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 down the number of BRM pod replicas in your deployment based on a pod's CPU or memory utilization.

For more information about:

- Kubernetes Horizontal Pod Autoscaler, see "Horizontal Pod Autoscaling" in the Kubernetes documentation
- Kubernetes requests and limits, see "Resource Management for Pods and Containers" in the Kubernetes documentation

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

- batch-controller
- brm-rest-services-manager
- cm
- dm-eai
- dm-kakfa

Copyright © 2021, 2026, Oracle and/or its affiliates.

- dm-oracle
- realtime-pipe
- rel-daemon
- rated-event-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. 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.
4. For each BRM pod, set these keys to the appropriate values for your system:
   - **ocbrm.*BRMPod*.resources.limits.cpu**: Set this to the maximum number of CPU cores the pod can utilize.
   - **ocbrm.*BRMPod*.resources.requests.cpu**: Set this to the minimum number of CPU cores required in a Kubernetes node to deploy a pod.

     The pod is set to **Pending** if the minimum CPU amount is unavailable.

     > **Note**
     >
     > The node must have enough CPUs available for the CPU requests of all containers of the pod. For example, the cm pod would need to have enough CPUs for the cm container, eai_js container, and perflib container (if enabled).

   - **ocbrm.*BRMPod*.resources.limits.memory**: Set this to the maximum amount of memory a pod can utilize.
   - **ocbrm.*BRMPod*.resources.requests.memory**: Set this to the minimum memory required for a Kubernetes node to deploy a pod.

     The pod is set to **Pending** if the minimum amount is unavailable 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. No changes are made if the number of pod replicas is already at the minimum.
   - **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 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**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **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**.

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

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile --
   namespace 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.

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 use 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:

Copyright © 2021, 2026, Oracle and/or its affiliates.

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

   kubectl get pods -n *NameSpace*

   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*

Setting Minimum and Maximum CPU and Memory Values

You can specify the minimum and maximum CPU and memory resources BRM cloud native containers can use. Setting minimum values ensures containers can deploy successfully while setting maximum values prevents containers from consuming excessive resources, which could lead to system crashes.

> **Note**
>
> For a pod to be scheduled on a node, the node must have enough CPUs available for the CPU requests of all containers of the pod. For example, in case of the cm pod, the node would need to have enough CPUs for the cm container, eai_js container, and perflib container (if enabled).
>
> You should also tune the JVM parameter for heap memory when tuning container-level resources for Java-based containers. You do this adjustment through component-level keys.

To set the minimum and maximum amount of CPUs and memory for containers, include the following keys in your **override-values.yaml** file for **oc-cn-helm-chart**, **oc-cn-init-db-helm-chart**, **oc-cn-op-job-helm-chart**, **oc-cn-ece-helm-chart**:

```
componentName:
   resources:
      requests:
         cpu: value
         memory: value
      limits:
         cpu: value
         memory: value
```

where:

- *componentName*: Specifies the component name in the **values.yaml** file, such as cm, rel_daemon, and vertex_dm.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **limits.cpu**: Specifies the maximum number of CPU cores the container can utilize, such as 1000m.
- **limits.memory**: Specifies the maximum amount of memory a container can utilize, such as 2000Mi.
- **requests.cpu**: Specifies the minimum number of CPU cores reserved in a Kubernetes node to deploy a container, such as 50m.
- **requests.memory**: Specifies the minimum amount of memory a container can utilize, such as 256Mi.

You must perform a Helm install or Helm upgrade after making any changes.

For more information about requests and limits, see "Resource Management for Pods and Containers" in the Kubernetes documentation.

Using Static Volumes

By default, the BRM cloud native pods use dynamic volume provisioning. However, you can modify one or more pods to use static volumes instead to meet your business requirements. To do so, you add **createOption** keys to the **override-values.yaml** file for each pod that you want to use static volumes and then redeploy your Helm charts.

To change a pod to use dynamic volumes, remove the **createOption** keys from your **override-values.yaml** file and then redeploy your Helm charts.

To change one or more pods to use static volumes, do the following:

1. Open the **override-values.yaml** file for the appropriate Helm chart: **oc-cn-op-job-helm-chart**, **oc-cn-helm-chart**, and **oc-cn-ece-helm-chart**.
2. Under the appropriate pod's **volume** section, update the **createOption** keys.

   For example, to use a hostPath-based volume, you would update the **createOption** key as shown below:

   ```
   volume:
     createOption:
       hostPath:
         path: pathOnNode
         type: Directory
   ```

   where *pathOnNode* is the location on the host system of the external PV.

   > **Note**
   >
   > The batchpipe, rated-event-manager, and rel_daemon pods require a separate volume for each schema in a multischema system. In this case, use *pathOnNode***/SCHEMA**. When you perform a helm upgrade or install, it replaces **SCHEMA** with the schema number. For example, the Helm chart replaces **SCHEMA** with **1** for schema 1, **2** for schema 2, and so on.

3. Save and close your **override-values.yaml** file.
4. Redeploy your Helm charts. For more information, see "Deploying BRM Cloud Native Services" in *BRM Cloud Native Deployment Guide*.

The following shows sample **override-values.yaml** keys for changing the brm-sdk, batchpipe, and batch-controller pods to use a static hostPath-based volume:

Copyright © 2021, 2026, Oracle and/or its affiliates.

```
ocbrm:
  brm_sdk:
    volume:
      storage: 50Mi
      createOption:
        hostPath:
          path: /sample/vol
          type: Directory
  batchpipe:
    volume:
      output:
        storage: 100mi
        createOption:
          hostPath:
            path: /sample/vol/out/SCHEMA
            type: Directory
      reject:
        storage: 100mi
        createOption:
          hostPath:
            path: /sample/vol/reject/SCHEMA
            type: Directory

  batch-controller:
    volume:
      input:
        storage: 50mi
        createOption:
          hostPath:
            path: /sample/vol/input
            type: Directory
```

Assigning Pods to Nodes Using nodeSelector and affinity

You can control where BRM cloud native pods run within your Kubernetes cluster by using the **nodeSelector** and **affinity** keys. Use these keys to ensure specific pods are scheduled only on suitable nodes, or to place certain pods together or apart for operational or compliance purposes. You can isolate workloads by scheduling critical components on dedicated nodes or optimize resource usage on specialized hardware, such as SSDs or GPU nodes.

BRM Helm charts expose **nodeSelector** and **affinity** as user-editable keys in the **override-values.yaml** file. This approach eliminates the need to modify chart templates.

For more information, see "Assigning Pods to Nodes" in the Kubernetes documentation.

Using nodeSelector

The **nodeSelector** key lets you specify that a pod runs only on nodes with specific labels, such as a particular hardware type or geographic region.

To assign a BRM pod to specific nodes with **nodeSelector**:

1. List node labels to find suitable scheduling criteria:

   ```
   kubectl get nodes --show-labels
   ```

2. In your **override-values.yaml** file for the appropriate Helm chart, add a **nodeSelector** section to the desired BRM component. For example, to add it to the CM:

Copyright © 2021, 2026, Oracle and/or its affiliates.

```
cm:
  nodeSelector:
    disktype: ssd
```

This configures the cm pod to run only on nodes labeled with **disktype** set to **ssd**.

3. Run the **helm upgrade** command for the appropriate Helm chart.

Using Affinity

The affinity key provides advanced scheduling controls:

- **Node affinity**: Prefer or require scheduling on nodes with specific labels.
- **Pod affinity**: Prefer or require placement with other specific pods.
- **Pod anti-affinity**: Prefer or require scheduling away from other specific pods.

To control pod placement using affinity or anti-affinity

1. Choose whether you want pods to run together (affinity) or apart (anti-affinity), and on what criteria (such as labels or topology).
2. In your **override-values.yaml** file for the appropriate Helm chart, add an **affinity** section to the desired BRM component. For example, to add it to the realtime-pipe pod:

```
realtime_pipe:
  affinity:
    podAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
                - key: app
                  operator: In
                  values:
                    - cm
            topologyKey: "kubernetes.io/hostname"
```

3. Run the **helm upgrade** command for the appropriate Helm chart.

Managing Labels and Annotations

You can add and manage Kubernetes labels and annotations for BRM resources created by Helm charts. This capability lets you integrate with cluster management tools, improve operations, and enable features such as webhooks or sidecar injectors that rely on metadata. You can add this metadata without modifying chart templates.

Table 1-1 describes how BRM cloud native uses labels and annotations.

Table 1-1 About Labels and Annotations

Term	Description	Examples
Labels	Key-value pairs that identify and group resources.	env=prod team=sre
Annotation	Key-value pairs used by tools read to determine behavior or actions.	Sidecar injection Webhook triggers Credential injection

Copyright © 2021, 2026, Oracle and/or its affiliates.

BRM Helm charts automatically add two standard labels to each resource:

- **app.kubernetes.io/name**: The application name for the resource.
- **app.kubernetes.io/part-of**: The BRM application or component it belongs to, such as brm or bcws.

When you configure custom labels, the system adds them in addition to these defaults.

You can extend labels and annotations for most BRM resources created by the charts, such as ConfigMaps, Secrets, Jobs, and Domain resources.

You can define metadata at three levels. If a key exists at multiple levels, BRM applies precedence as shown below:

1. **Resources**: Applies to a single resource by its **metadata.name**. For resources with ordinal suffixes, such as **my-app-1** or **my-app-2**, use the base name, such as **my-app**.
2. **Kind**: Applies to all resources of a specific Kubernetes kind. Use the exact name of the kind, such as Deployment, ConfigMap, or ServiceAccount.
3. **Global**: Applies to all resources in the chart.

To add labels or annotations:

1. Edit your **override-values.yaml** for the appropriate chart. Use the **metadata.labels** and **metadata.annotations** sections with the **global**, **kind**, and **resource** subkeys.
2. Run the **helm upgrade** command for the appropriate chart.
3. Validate the labels and annotations on the resources using **kubectl**. For example:
   - To view labels on a deployment:

     ```
     kubectl get deployment cm -n BrmNamespace --show-labels
     ```

   - To view annotations on a ConfigMap:

     ```
     kubectl get configmap cm-dep-plan -n BrmNamespace -o
     jsonpath='{.metadata.annotations}'
     ```

   - To list resources with a specific label:

     ```
     kubectl get pods -n BrmNamespace -l env=prod
     ```

Example Resource Labels and Annotations

This example demonstrates how to configure resource labels and annotations in your **override-values.yaml** file:

```
metadata:
   labels:
      resources:
         cm-dep-plan: # exact metadata.name (base name if ordinals are added at
runtime):
            component: "bcws"
            managedBy: "helm"
   annotations:
      resources:
         cm-dep-plan:
            checksum/config: "abc123"  # example for rolling restarts based on config
checksum
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

Example Kind Labels and Annotations

This example demonstrates how to configure kind labels and annotations in your **override-values.yaml** file:

```
metadata:
   labels:
      kind:
         Deployment:
            workload: "stateless"
         ConfigMap:
            configRole: "primary"
   annotations:
      kind:
         ServiceAccount: eks.example.com/role-arn: "arn:example:iam::123456789012:role/
brm-sa-role"
```

Example Global Labels and Annotations

This example demonstrates how to configure global labels and annotations in your **override-values.yaml** file:

```
metadata:
   labels:
      global:
         env: "prod"
         team: "sre"
         annotations: global:
         owner: "name@example.com"
         purpose: "ops-visibility"
```

About Customizing and Extending Pods

You can add or override settings in BRM cloud native pod specifications without modifying Helm chart templates or waiting for a product update. By adding values under the **addOnPodSpec** key in your **override-values.yaml** file, you can enable new features, enforce custom security policies, or change pod deployment behavior from a single configuration location.

During a Helm install or Helm upgrade, the Helm chart checks for any **addOnPodSpec** values. Settings defined under this key are merged into the pod specification at deployment and override identical settings specified elsewhere in the chart.

> **Note**
>
> If a setting appears both elsewhere in the chart and under **addOnPodSpec**, the value in **addOnPodSpec** takes precedence.

Before customizing or extending pods, carefully review the following guidelines:

- **Use Direct Fields First**: Where possible, use dedicated **values.yaml** keys, such as **affinity** or **nodeSelector**, for common pod configuration requirements.
- **Use addOnPodSpec for Extensions or Overrides**: Use **addOnPodSpec** for custom settings not available as dedicated keys, or to override default behaviors.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **Document Your Customizations**: Record any changes made in **addOnPodSpec** for troubleshooting, maintenance, and auditing purposes.

The general steps for customizing or extending a pod:

1. Open the **override-values.yaml** file for your Helm chart.
2. Add the settings under the **addOnPodSpec** key.

   For example, to add a **securityContext** to a pod:

   ```
   addOnPodSpec:
     securityContext:
       runAsUser: 1200
       fsGroup: 1000
   ```

   This specifies the user and groups under which a pod runs.
3. Deploy or update your BRM environment using the **helm install** or **helm upgrade** command with your updated **override-values.yaml** file.

Example Specifying Node Tolerations

This example demonstrates using the **addOnPodSpec** key to control on which nodes the pods can run:

```
addOnPodSpec:
  tolerations:
    - key: "example.com/role"
      operator: "Equal"
      value: "special"
      effect: "NoSchedule"
```

Example Distributing Pods Across Nodes

This example demonstrates using the **addOnPodSpec** key to distribute pods evenly across nodes, improving resilience and availability:

```
addOnPodSpec:
  topologySpreadConstraints:
    - maxSkew: 1
      topologyKey: kubernetes.io/hostname
      whenUnsatisfiable: DoNotSchedule
      labelSelector:
        matchLabels:
          app: your-app
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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 and brm-apps jobs.

Topics in this document:

- Running Load Utilities Through Configurator Jobs
- Running Load Utilities on Multischema Systems
- Running Applications and Utilities Through brm-apps Jobs
- Running Custom Applications and Utilities Through brm-apps
- Running Business Operations Through pin_job_executor Service

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 only 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
   ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile --
   namespace 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.

   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 --namespace 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 --
   namespace BrmNameSpace
   ```

5. Restart the CM because **pin_bus_params** requires it.

   a. Set these keys in the **override-values.yaml** file:

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **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 --namespace 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 only 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 --
namespace 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 --namespace BrmNameSpace
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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 run 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 --
   namespace 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.

   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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 backends, and most backends are dedicated to processing transactions.

- **mtaPerBatch**: Specifies the number of payment transactions 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 sends four batches to dm_fusa (each batch containing 5,000 payment transactions).

  > **Note**
  >
  > This key impacts the performance of the **pin_collect** application only. It has minimal impact on other applications.

- **mtaPerStep**: Specifies how much data to store in dm_oracle when the application performs a step search. It does not significantly impact performance but governs 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 parent accounts. When running applications on only the children (**pay_type 10007**), you would set **mtaFetchSize** to 500,000 to retrieve all 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".

Copyright © 2021, 2026, Oracle and/or its affiliates.

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:16.26.0.x.x
       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 image by entering this command:

   ```
   podman build --format docker --tag brm_apps:16.26.0.x.x-custom --file
   dockerfile_custom_brm_apps .
   ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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 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*.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 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 --
   namespace 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.

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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 --
   namespace 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**: This configuration file synchronizes BRM with the ECE rating engine.
- **payloadconfig_ifw_sync.xml**: This configuration file synchronizes BRM with the batch and real-time rating engine.
- **payloadconfig_kafka_sync**: This configuration file is for publishing business events from BRM to the Kafka server.

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 --
   namespace 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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
- Tracking a Release's Status
- Updating a Helm Release
- Checking a Release's Revision
- Rolling Back a Release To a Previous Revision

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. Only some objects are 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 --namespace Namespace
```

where:

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

Updating a Helm Release

To update any **override-values.yaml** key value after creating a release, run the following command. This command updates or re-creates the impacted Kubernetes objects without impacting other objects in the release. It also makes a new revision of the release.

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **Note**
>
> Before updating the release, you can check for issues by running the **helm upgrade** command and appending the **--dry-run** parameter.

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

where:

- *ReleaseName* is the name you assigned to this installation instance.
- *Namespace* is the namespace 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 --namespace Namespace
```

where:

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

Copyright © 2021, 2026, Oracle and/or its affiliates.

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:

- Using External Kubernetes Secrets for Certificates and Wallets
- Rotating the BRM Root Password
- Rotating the BRM Root Key
- Rotating the BRM Password
- Rotating BRM Role Passwords

Using External Kubernetes Secrets for Certificates and Wallets

To increase system security and simplify certificate and wallet maintenance, you can store keystore certificates and Oracle wallets in precreated Kubernetes secrets. Create the required certificate or wallet material, add it to a Kubernetes secret, and configure the appropriate external secret key in your **override-values.yaml** file.

For TLS connections to BRM, use certificates issued by a certificate authority (CA) approved by your organization. BRM cloud native container images do not include populated TLS wallets, private keys, or sample CA certificates. You provide and manage the TLS certificates, private keys, and trust material that your deployment requires. You can provide BRM TLS wallets through Kubernetes secrets or, where supported, through customer-provided wallet files in the Helm chart.

When you perform a Helm install or Helm upgrade, the Helm chart checks the configured external secret keys. If you specify an external secret, the chart uses the precreated Kubernetes secret. If you do not specify an external secret, the behavior depends on the component and Helm chart. Where a chart-file source is supported, the chart creates a Kubernetes secret from the customer-provided certificate or wallet files. The Helm chart does not generate the certificate or wallet material.

For BRM Server TLS wallets, if you enable TLS and do not provide a supported wallet source, the Helm installation fails. For information about the supported wallet sources and their selection order, see "How BRM Selects the TLS Wallet".

You can use external Kubernetes secrets for the following:

- Billing Care
- Billing Care REST API
- BRM REST Services Manager
- BRM Server
- Business Operations Center
- Collections Configuration Center
- Elastic Charging Engine

Copyright © 2021, 2026, Oracle and/or its affiliates.

- Pipeline Configuration Center
- Pricing Design Center (PDC)
- PDC REST Services Manager
- Web Services Manager

To use external Kubernetes secrets in your BRM cloud native deployment:

1. Create the required keystore certificates and Oracle wallets, and store them in Kubernetes secrets in your Kubernetes cluster.

   For TLS wallets, ensure that the certificates and private keys required by the component and TLS mode.

   For information about creating Kubernetes secrets, see "Managing Secrets" in the Kubernetes documentation.

2. In the **override-values.yaml** file for the appropriate Helm chart, set the external secret key for each certificate or wallet to the name of the Kubernetes secret that you created.

   For BRM Server TLS wallets, set the appropriate component-specific **extUserSSLWalletSecret** key to the name of the precreated Kubernetes secret. BRM also supports shared wallet sources.

   Depending on the component, configure the appropriate external secret keys in:

   - **oc-cn-init-db-helm-chart**
   - **oc-cn-helm-chart**
   - **oc-cn-op-job-helm-chart**
   - **oc-cn-ece-helm-chart**

   > **Note**
   >
   > The precedence between an external Kubernetes secret and other supported certificate or wallet sources depends on the component.

3. Perform a Helm install or Helm upgrade.

Rotating the BRM Root Password

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

Changing the BRM root password 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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

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

   `helm upgrade` *`BrmReleaseName`* **`oc-cn-helm-chart --values`** *`OverrideValuesFile`* **`--namespace`** *`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.

   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.

3. 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_pass	*RootPassword2*	Provide the updated password for the **root.0.0.0.1** service.

4. 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).

Copyright © 2021, 2026, Oracle and/or its affiliates.

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**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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

5. Delete the PDC and Pipeline Configuration Center deployments:

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

6. 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
   ```

7. Update the BRM root password in your ECE pods by doing this:

   a. Connect to any of the charging server (ecs) pods through JConsole. See "Creating a JMX Connection to ECE Using JConsole" for more information.

   b. Expand the **ECE Configuration** node.

   c. Navigate to the BRM Connection node.

   d. Expand **Operations**.

   e. Enter the new BRM root password (*RootPassword2*) along with the existing wallet password in the **setPassword** method and then run it.

   f. Perform a test connection to validate that the connection is successful.

   g. 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 the BRM root key regularly to increase security.

> **Note**
>
> In BRM cloud native, the root key wallet is stored in the BRM database. The root key wallet is not stored in BRM container images or mounted in BRM pods.

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 --
   namespace 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- *BrmNameSpace* is the namespace 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 one is generated in the Oracle wallet. You can set **ocbrm.root_key_rotate** to **false** to avoid rotating the root key again.

After you rotate the root key once, use the following method to rotate the root key again:

1. Delete the dm-oracle deployment.
2. In your **override-values.yaml** file for **oc-cn-helm-chart**, ensure that the **ocbrm.root_key_rotate** key is set to **true**.
3. Run the **helm upgrade** command to update your Helm release:

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

Rotating the BRM Password

To rotate the BRM password, stop and 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 --
   namespace 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.
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 --
   namespace BrmNameSpace
   ```

Rotating BRM Role Passwords

You set the initial passwords for each role in your BRM cloud native system when you deploy or upgrade your BRM cloud native server and database schema. These passwords are stored in the Oracle wallet. After your system is deployed, you should rotate your role passwords regularly.

To rotate your BRM role passwords after deployment:

Copyright © 2021, 2026, Oracle and/or its affiliates.

1. Open your **override-values.yaml** file for **oc-cn-helm-chart**.
2. Set the **ocbrm.rotate_brm_role_passwords** key to **true**.
3. Specify the old and new passwords for each role password that you want to rotate using this format:

```
ocbrm:
   brm_role_pass:
      old_roleName.0.0.0.1: oldPassword
      roleName.0.0.0.1: newPassword
```

where:

- *oldPassword* is the old password in Base64-encoded format.
- *newPassword* is the new password in Base64-encoded format.
- *roleName* is one of the following:
  - **acct_recv** for the Accounts Receivable role. Users with this role can run the AR utilities such as **pin_refund** and **pin_monitor_balance**.
  - **bc_client** for the Billing Care role. Users with this role can run the Billing Care application.
  - **bill_inv_pymt_sub** for the Invoice Payments role. Users with this role can run the invoicing utilities such as **pin_inv_accts** and **pin_upd_assoc_bus_profile**.
  - **billing** for the Billing role. Users with this role can run billing applications such as **pin_bill_accts** and **pin_rollover**.
  - **boc_client** for the Business Operations Center role. Users with this role can run the Business Operations Center application.
  - **collections** for the Collections role. Users with this role can run the collections utilities such as **pin_collections_process** and **pin_collections_send_dunning**.
  - **crypt_utils** for the Encryption role. Users with this role can run the encryption utilities such as **pin_crypt_upgrade** and **pin_crypt_upgrade_keys**.
  - **cust_mgnt** for the Customer Management role. Users with this role can run the customer management utilities such as **pin_contracts** and **pin_state_change**.
  - **invoicing** for the Invoicing role. Users with this role can run the invoicing utilities such as **pin_inv_accts** and **pin_inv_doc_gen**.
  - **java_client** for the Java Applications role. Users with this role can run Java applications such as Account Migration Manager and Conversion Manager.
  - **load_utils** for the Load Utilities role. Users with this role can run the load utilities such as **load_config** and **load_pin_calendar**.
  - **payments** for the Payments role. Users with this role can run the payment utilities such as **pin_installments** and **pin_recover**.
  - **pcc_client** for the Pipeline Configuration Center role. Users with this role can run the Pipeline Configuration Center application.
  - **rerating** for the Rerating role. Users with this role can run the rerating utilities such as **pin_rerate** and **pin_rate_change**.
  - **rsm** for the REST Services Manager role. Users with this role can call the BRM REST Services Manager API operations.
  - **super_user** for the Super User role. Users with this role can create, update, and delete other roles.

Copyright © 2021, 2026, Oracle and/or its affiliates.

  - **ui_client** for the Thick Clients role. Users with this role can run the thick clients such as Number Administration Center.
  - **ece** for the ECE role. Users with this role can run Elastic Charging Engine (ECE).

For example, to rotate the Business Operations Center and Pipeline Configuration Center roles, you would enter the following:

```
ocbrm:
   brm_role_pass:
      old_boc_client.0.0.0.1: oldBOCPassword
      boc_client.0.0.0.1: newBOCPassword
      old_pcc_client.0.0.0.1: oldPCCPassword
      pcc.0.0.0.1: newPCCPassword
```

4. Keep all other role passwords in the file.
5. Run the **helm upgrade** command to update your Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile --
   namespace 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.

If successful, BRM cloud native:

- Rotates the passwords for role names prefixed with **old**_.
- Generates the new role passwords in the Oracle wallet.

All other passwords remain the same. You can set **ocbrm.rotate_brm_role_passwords** to **false** to avoid rotating the role passwords again.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 BRM Cloud Native Database
- Dropping Partitions from Your BRM Cloud Native 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:

Copyright © 2021, 2026, Oracle and/or its affiliates.

- *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.

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 BRM Cloud Native 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. Add the appropriate lines for running the **partition_utils** utility to the **oc-cn-helm-chart/brmapps_scripts/loadme.sh** script.

   For example, these lines add delayed event partitions for 12 months:

   ```
   #!/bin/sh

   cd /oms/apps/partition; partition_utils -o enable -t delayed -c /event/delayed/session%
   cd /oms/apps/partition; partition_utils -o add -t delayed -s StartDate -u month -q 12 -f

   exit 0;
   ```

   where *StartDate* is 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.
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.
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
   ```

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

Dropping Partitions from Your BRM Cloud Native Database

You can drop a subset of real-time, delayed-event, or prepaid partitions that occur between a specified start date and end date. By default, partitions that contain objects associated with open items are not removed. You can use the **-f** parameter to remove partitions that contain objects associated with open items, but this parameter must be used with care.

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **Note**
>
> Before you remove partitions that contain objects associated with open items, verify that removing them does not conflict with your business practices.

To drop partitions from the database in your BRM cloud native environment:

1. Add the appropriate lines for running the **partition_utils** utility to the **oc-cn-helm-chart/brmapps_scripts/loadme.sh** script. Include the **-f** option to remove partitions even if they contain objects associated with open items.

   ```
   #!/bin/sh

   cd /oms/apps/partition; partition_utils -o remove -s StartDate -e EndDate [-f]

   exit 0;
   ```

   where:

   - *StartDate* is the start of the date range for the objects you want to remove. The format is *MMDDYYYY*.
   - *EndDate* is the end of the date range for the objects to remove. The format is *MMDDYYYY*.

2. By default, you can only remove partitions that are older than 45 days. To change the 45-day limitation, edit the following entries in the pin-conf-config-brm-apps-1 ConfigMap (**configmap_pin_conf_brm_apps_1.yaml**):

   - **$MAX_REMOVE_REAL{'older_than'}**
   - **$MAX_REMOVE_DELAYED{'older_than'}**
   - **$MAX_REMOVE_PREPAID{'older_than'}**

   For more information, see "Customizing Partition Limitations" in *BRM System Administrator's Guide*.

3. The brm-apps-partition-cfg ConfigMap (**configmap_partition_cfg.yaml**) controls partition parameters, including the database name and the partition logging directory. If you need to change the partition parameters, edit them in the file.

4. Enable the brm-apps job. In your **override-values.yaml** file for **oc-cn-helm-chart**, set **ocbrm.brm_apps.job.isEnabled** to **true**.

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

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

The brm_apps job runs a series of partitioning scripts that remove the partitions.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 communication performance between CM and Oracle DM by deploying the CM and Oracle DM containers in the same Kubernetes pod.

To deploy CM and Oracle DM in the same pod:

1. In the **oc-cn-helm-chart/templates** directory, rename the **dm_oracle.yaml** file to **_dm_oracle.yaml**.

   Renaming the file prevents the Helm chart from creating Oracle DM as a separate pod.

2. From **_dm_oracle.yaml**, copy the dm-oracle container definition into the **containers** section of **cm.yaml**.

   Copy all configuration that the dm-oracle container references, including its environment variables, ports, probes, volume mounts, and corresponding volume definitions.

   If you enable TLS for Oracle DM, configure its TLS wallet by using the standard BRM TLS wallet settings. For example, you can set **ocbrm.dm_oracle.extUserSSLWalletSecret** to the Kubernetes secret that contains the Oracle DM wallet. For information about BRM TLS wallet provisioning and wallet-source selection, see "Configuring TLS Certificates for BRM Cloud Native".

3. In the copied dm-oracle container definition, set **SERVICE_FQDN** to **localhost**:

   ```
   - name: SERVICE_FQDN
     value: "localhost"
   ```

   CM and Oracle DM run in the same pod and communicate by using the loopback interface.

4. Copy the Oracle DM annotations from **_dm_oracle.yaml** to the corresponding annotations section in **cm.yaml**.

   Include the annotations required for the Oracle DM configuration so that changes to its ConfigMaps trigger the appropriate pod update.

5. In the **cm-pin-conf-config** ConfigMap, change the **dm_pointer** entry so that CM connects to Oracle DM through **localhost** instead of the dm-oracle Kubernetes service. For example:

   - cm dm_pointer *databaseNumber* **ip localhost 12950**

Copyright © 2021, 2026, Oracle and/or its affiliates.

where *databaseNumber* is the BRM database number.

6. Save your changes to **cm.yaml**.
7. Run the **helm upgrade** command to update your Helm release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile --
namespace 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.

8. Verify that the CM pod contains both the CM and dm-oracle containers and that both containers are running.

Tuning Your Application Connection Pools

You can improve an application's performance by tuning the number of threads available to connect with the CM.

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

The request is queued if an incoming request cannot be assigned a thread immediately. The request waits for a thread to become available for a configurable period. 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 your **override-values.yaml** file.
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.
infranet.connectionpool.maxsize	The maximum number of threads the application can spawn for accepting requests from the CM.
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.
infranet.connectionpool.maxidletime	The time, in milliseconds, that an unused thread remains in the connection pool before it is removed. **Important**: If the value is too low, threads might be removed and restored too frequently. This can degrade system performance.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 7-1 (Cont.) Connection Pool Parameters

Entry	Description
infranet.connectionpool.maxrequestlistsize	The maximum number of requests that can be held in the pending request queue.
infranet.connectionpool.reconnectfrequency	The interval at which the application refreshes its connections to the CM.

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 --
   namespace 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.

Configuring Application Connection Pools with Multiple CM Replicas

In a cloud native deployment, an application connects to the CM Kubernetes service instead of connecting directly to a specific CM pod. The CM service distributes the application's requests among the available CM pods.

The **infranet.connectionpool.minsize** and **infranet.connectionpool.maxsize** entries apply to each client application replica, not to each CM pod. For example, if a client application has five replicas and **infranet.connectionpool.maxsize** is set to 40, the application can create up to 200 connections across the five replicas.

You configure the pool size based on the application's concurrent workload and the capacity of the CM deployment. There is no need to configure a separate load balancer between an application, such as a REST adapter, and the CM pods.

Use the CM Kubernetes service to distribute opcode requests to the CM pods. As CM pods are added, removed, or restarted, use **infranet.connectionpool.reconnectfrequency** to refresh the application's connections. When connections are refreshed or re-established, the CM Kubernetes service can distribute them to available CM pods.

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:

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 the batch-controller pod.
2. Run the **helm upgrade** command to update your Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile --
   namespace 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.

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, each 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 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.

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

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

   where:

Copyright © 2021, 2026, Oracle and/or its affiliates.

- *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.

Using the Paymentech Simulator

For testing purposes, a second deployment of the Paymentech 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 --
namespace BrmNameSpace
```

The deployment scripts and configuration files for the Paymentech Simulator are only provided for testing. 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**

Copyright © 2021, 2026, Oracle and/or its affiliates.

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
- Adding Schemas to a Multischema System
- Migrating Accounts from One Schema to Another
- Migrating Accounts Using Custom Search Criteria
- Modifying Database Schema Priorities
- Modifying Database Schema Status
- Synchronizing /uniqueness Objects Between Schemas

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 runs 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 --
   namespace BrmNameSpace
   ```

   where:

Copyright © 2021, 2026, Oracle and/or its affiliates.

- *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.

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 an **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 host name 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: Base64EncodedPassword
            schematablespace: pin02
            indextablespace: pinx02
         secondary2:
            deploy: false
            schemauser: pin03
            schemapass: Base64EncodedPassword
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
      schematablespace: pin03
      indextablespace: pinx03
   secondary3:
      deploy: true
      schemauser: pin04
      schemapass: Base64EncodedPassword
      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 an **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 host name 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.
- **tnsAdmin.connectionDescriptor**: Use this key when the database connection requires a RAC address list or another custom descriptor. The chart adds the **pindb** alias. When this value is empty, **oc-cn-helm-chart** generates the existing connection descriptor from the database host, port, service name, and SSL settings.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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: Base64EncodedPassword
            schematablespace: pin02
            indextablespace: pinx02
            tnsAdmin:
               connectionDescriptor: OracleNetConnectionDescriptor
         secondary2:
            deploy: true
            schemauser: pin03
            schemapass: Base64EncodedPassword
            schematablespace: pin03
            indextablespace: pinx03
            tnsAdmin:
               connectionDescriptor: OracleNetConnectionDescriptor
         secondary3:
            deploy: true
            schemauser: pin04
            schemapass: Base64EncodedPassword
            schematablespace: pin04
            indextablespace: pinx04
            tnsAdmin:
               connectionDescriptor: OracleNetConnectionDescriptor
```

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

```
helm upgrade 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 where BRM Kubernetes objects for the BRM Helm chart were previously deployed.
- *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 creates new dm-oracle, amt, and rel-daemon 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. 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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, see "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

path='realpath config_dist.conf'
cp $path /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 --namespace 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**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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

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

The CM is restarted.

4. Reset BRM POID 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 $ORACLE_HOME/lib/ojdbc11.jar:$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 --namespace BrmNameSpace
```

5. 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 --namespace 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**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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

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

The CM is restarted.

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

```
#!/bin/sh
cd /oms/setup/scripts; perl pin_amt_install.pl
cd /oms/setup/scripts; perl pin_amt_install.pl -m
exit 0;
```

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

```
helm upgrade BrmReleaseName oc-cn-helm-chart –-namespace BrmNameSpace --values
OverrideValuesFile
```

   d. Restart the amt pod.
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 **router** prefix and an **.edr** suffix.
   - 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 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. Enable Account Migration Manager in your BRM database by setting the **ocbrm.isAmt** key to your **override-values.yaml** file.
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**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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, set this to an integer between 50 and 100 for optimal performance. 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
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

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 8-1 (Cont.) Account Search Parameters

Parameter	Description	Required
bill_day_of_month	Migrates accounts that have the specified billing day of the 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 --
namespace 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.

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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 specific American state or belonging to a particular service provider.

To migrate accounts using custom search criteria:

1. Enable Account Migration Manager in your BRM database by setting **ocbrm.isAmt** to **true** in your **override-values.yaml** file.
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. Surround search variables with curly braces "**{ }**" and ensure they match an entry under the **account_search.cfg** section.

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

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

Copyright © 2021, 2026, Oracle and/or its affiliates.

For example, this SQL fragment enables AMM to search for accounts 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);
         }
       }
       ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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 --
namespace 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 are 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

```
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 --
   namespace 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 --namespace BrmNameSpace
      ```

      The CM is restarted.

Modifying Database Schema Status

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

- **Open:** Open schemas are available for account creation.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **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. It continues creating 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 have Multidatabase Manager automatically switch 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**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile --
   namespace 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 --namespace 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 --
   namespace 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

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **ocbrm.config_jobs.run_apps**: Set this to **false**

b. Update the Helm release again:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values
OverrideValuesFile --namespace 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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
- Loading Legacy Data into the BRM Database

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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 --
      namespace 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 --
      namespace 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
      ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
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 --
namespace 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 --
namespace 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

10

Creating Custom Fields and Storable Classes

Learn how to create custom fields and storable classes in Oracle Communications Billing and Revenue Management (BRM) cloud native using the BRM SDK opcodes or the **pin_deploy** utility.

Topics in this document:

- Creating, Editing, and Deleting Fields and Storable Classes using BRM SDK Opcodes
- Moving Field and Storable Class Definitions Between BRM Servers with pin_deploy

The storable class structure is described in "Understanding Storable Classes" in *BRM Developer's Guide*.

Creating, Editing, and Deleting Fields and Storable Classes using BRM SDK Opcodes

You can use the BRM SDK opcodes to create, modify, delete, or retrieve storable class and field specifications from the BRM database. For more information about the BRM SDK, see "Using BRM SDK" in *BRM Developer's Guide*.

To manage field and storable class specifications using the BRM SDK opcodes:

1. Enable changes to the data dictionary. See "Making the Data Dictionary Writable".
2. Create, edit, or delete your custom storable classes and fields by running the BRM SDK opcodes. See "Running the BRM SDK Opcodes".
3. Make your custom fields and storable classes available to BRM applications by generating source and header files.
   - For BRM applications written in PCM C and C++, see "Making Custom Fields Available to Your PCM and C++ Applications".
   - For BRM applications written in Java PCM, see "Making Custom Fields Available to Your Java PCM Applications".

Making the Data Dictionary Writable

Ensure the data dictionary is writable before adding or changing fields and storable classes.

Perform the following for each database in your system:

1. Enable changes to the data dictionary by doing the following:
   a. Open the **dm-oracle-pin-conf-config** ConfigMap.
   b. Ensure the following fields are set to **1**:

```
data:
  pin.conf: |
    - dm dd_write_enable_fields 1
    - dm dd_write_enable_objects 1
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

c. Save and close the file.

2. (Optional) To increase the size of the CM cache for the data dictionary, do the following:

   a. Open the **cm-pin-conf-config** ConfigMap.

   b. Increase the *cacheSize* value in the following entries:

```
data:
  pin.conf: |
    - cm_cache cm_data_dictionary_cache numberOfEntries, cacheSize, hashSize
    - cm_cache fm_utils_data_dictionary_cache numberOfEntries, cacheSize,
hashSize
```

   c. Save and close the file.

3. (Optional) To configure whether the DM runs Data Definition Language (DDL) when updating object types in the data dictionary tables, do the following:

   a. Open the **dm-oracle-pin-conf-config** ConfigMap.

   b. Set the following entry:

```
data:
  pin.conf: |
    - dm sm_oracle_ddl value
```

   where *value* is **1** to run DDLs when updating object types or **0** not to run them. Setting *value* to **1** ensures that database objects are mapped to the correct tables.

   c. Save and close the file.

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 namespace in which to create BRM Kubernetes objects for the BRM Helm chart.

Running the BRM SDK Opcodes

You can run the BRM SDK opcodes in the BRM cloud native system without entering a pod by running the **testnap** utility through a configurator job. For more information about:

- The **testnap** utility's syntax and parameters, see "testnap" in *BRM Developer's Guide*.
- Configurator jobs, see "Running Load Utilities Through Configurator Jobs".

To run the BRM SDK opcodes:

1. Create a **testnap** script with the following content:

```
r << XXX 1
flistContent
XXX
xop opcodeNumber bufferNumber
```

   where:

Copyright © 2021, 2026, Oracle and/or its affiliates.

- *flistContent* is the input flist for the BRM SDK opcode you want to run.
- *opcodeNumber* is the number of the BRM SDK opcode to run. For the list of opcode numbers for the BRM SDK opcodes, see Table 10-1 and Table 10-2.
- *bufferNumber* is the internal buffer number used to load the opcode's input flist file.

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

```
#!/bin/sh

cd /oms/sys/data/config; testnap testnapScript
exit 0;
```

   where *testnapScript* is the name and path to the **testnap** script you created.

3. Move *testnapScript* to the **oc-cn-helm-chart/config_scripts** directory.
4. In the **override-values.yaml** file for **oc-cn-helm-chart**, set **ocbrm.config_jobs.run_apps** to **true**.
5. Run the **helm upgrade** command for **oc-cn-helm-chart**:

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

Table 10-1 describes the BRM SDK opcodes to run for creating, modifying, deleting, and retrieving storable classes.

Table 10-1 BRM SDK Opcodes for Storable Classes

Opcode Name	Opcode Number	Description
PCM_OP_SDK_SET_OBJ_SPECS	578	Creates or modifies a storable class specification in the data dictionary of all databases in your BRM system. It takes the following as input: POID, storable class name, and storable class type. If the transaction is successful, the opcode returns the POID of the created or modified storable class and a results array containing an SQL description of any table changes. **Note:** If you change a storable class after it has been instanced and populated with data, your database will be corrupted.
PCM_OP_SDK_GET_OBJ_SPECS	577	Retrieves the storable class specifications specified in the input flist. You can retrieve specific levels or types of objects by using the wildcard (*****) character. When no storable classes are specified, the opcode returns all storable class specifications in the BRM database. If the transaction is successful, the opcode returns the specified storable class specifications or all storable class specifications if the input flist does not specify a storable class.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 10-1 (Cont.) BRM SDK Opcodes for Storable Classes

Opcode Name	Opcode Number	Description
PCM_OP_SDK_DEL_OBJ_SPECS	583	Deletes storable class specifications from the data dictionary only. To drop the actual table that was created by PCM_OP_SDK_SET_OBJ_SPECS, you must drop it manually. The opcode deletes storable class specifications from the data dictionary of all databases in your BRM system. If the transaction is successful, the opcode returns the POID of the deleted storable class and a results array containing an SQL description of any table changes. **Note:** Deleting a storable class that has already been instantiated corrupts your database. For example, never delete the **/account** storable class. Because of this danger, Oracle recommends not using this opcode on a production system.

Table 10-2 describes the BRM SDK opcodes to run for creating, modifying, deleting, and retrieving field definitions.

Table 10-2 BRM SDK Opcodes for Field Definitions

Opcode Name	Opcode Number	Description
PCM_OP_SDK_SET_FLD_SPECS	576	Creates or modifies the specified field specification in the data dictionary of all databases in your BRM system. It takes the following as input: a partial POID (database number plus **/dd/fields**), field name, and field type. **Note:** The POID is the only mandatory field on the input flist. However, to implement the field, you must at least specify the field name and type. The opcode returns the POID of the created or modified data dictionary field if the transaction is successful. If the opcode cannot create or modify the field, the opcode returns the field's POID, along with the PIN_FLD_ACTION field set to **NOOP**.
PCM_OP_SDK_GET_FLD_SPECS	575	Retrieves all field specifications specified on the input flist. When no fields are specified, this opcode returns all field specifications in the BRM database. **Note:** Returning all field specifications can take a long time.
PCM_OP_SDK_DEL_FLD_SPECS	585	Deletes the specified field specification from the data dictionary of all databases in your BRM system. It takes the following as input: the partial POID (database number plus **/dd/fields**) and the name of the field to delete. If the transaction is successful, the opcode returns the POID of the deleted field specification.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Making Custom Fields Available to Your PCM and C++ Applications

After you create custom fields and storable classes, you must make them available to your BRM cloud native applications. The first step is to create a C header file, and then you generate a Java package with the custom storable classes.

To make custom fields available to your BRM cloud native applications written in PCM C or PCM C++:

1. Create a C header file for your custom fields. For information about the syntax to use in a header file, view the *BRM_home*/**include/pin_flds.h** file in the brm-sdk pod.
2. Enable the brm-sdk pod if you haven't already done so.
   a. Set these keys in your **override-values.yaml** file for **oc-cn-helm-chart**:

      ```
      brm_sdk:
         isEnabled: true
         deployment:
            imageName: brm_sdk
            imageTag: 16.26.0.x.x
         pvc:
            storage: 50Mi
      ```

   b. Run the **helm upgrade** command for **oc-cn-helm-chart**:

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

      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 **values.yaml** file for **oc-cn-helm-chart**.
      - *BrmNameSpace* is the namespace in which BRM Kubernetes objects were created for **oc-cn-helm-chart**.
3. Copy your C header file to the brm-sdk pod:

   ```
   cp fileName oc-cn-helm-chart/brm_sdk_scripts/
   helm upgrade BrmReleaseName oc-cn-helm-chart --namespace BrmNameSpace --values
   OverrideValuesFile
   ```

   where *fileName* is the name of your C header file.

   For example, if the C header file is named **cust_flds.h**:

   ```
   cp cust_flds.h oc-cn-helm-chart/brm_sdk_scripts/
   helm upgrade BrmReleaseName oc-cn-helm-chart --namespace BrmNameSpace --values
   OverrideValuesFile
   ```

   Afterward, the files from **oc-cn-helm-chart/brm_sdk_scripts/** are available at **/oms/load** in the brm-sdk pod.

Copyright © 2021, 2026, Oracle and/or its affiliates.

4. Find the name of the brm-sdk pod:

   kubectl get pods -n *BrmNameSpace* **| grep brm-sdk**

   You should see something similar to this:

   ```
   NAME                          READY    STATUS     RESTARTS    AGE
   brm-sdk-f67b95777-bf8j5       1/1      Running    0           18m
   ```

   In this case, the brm-sdk pod name (*brmSDKPodName*) is **brm-sdk-f67b95777-bf8j5**.

5. Run the **kubectl exec** command to get a shell to the running brm-sdk container:

   kubectl exec -it -n *BrmNameSpace brmSDKPodName* **bash**

   For example:

   kubectl exec -it -n *BrmNameSpace* brm-sdk-f67b95777-bf8j5 bash

6. In your shell, go to the **/oms/load** directory and run the **parse_custom_ops_fields.pl** script with the custom source file:

   perl parse_custom_ops_fields -L pcmc -I *filename* **-O** *custFlds* **-P** *javaPackageName*

   where:

   - *custFlds*: Specifies the name and the location of the memory-mapped output file to create.
   - *javaPackageName*: Specifies the name of the Java package in which to create the generated classes.

   For example, if the C header file is named **cust_flds.h**:

   perl parse_custom_ops_fields -L pcmc -I cust_flds.h -O *outputFile* -P *javaPackageName*

   For information about the parameters and syntax for **parse_custom_ops_fields**, see "parse_custom_ops_fields" in *BRM Developer's Guide*.

7. Configure one or more BRM applications to access your custom fields by doing the following for each BRM application:

   a. Open the application's **oc-cn-helm-chart/templates/configmap_pin_conf_*.yaml** file.

   b. Add the following entry, replacing *custFlds* with the file name and location of the memory-mapped output file that **parse_custom_ops_fields** created:

      - - ops_fields_extension_file *custFlds*

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **Note**
>
> Do not add more than one **ops_fields_extension_file** entry. The custom fields source file and the extension file that results from it contain information about all the custom fields in the data dictionary, so a single reference to that file is sufficient.

c. Include the **cust_flds.h** header file in the applications and FMs that use the custom fields.

> **Note**
>
> Default BRM fields are defined with their numbers in the *BRM_home***/include/pin_flds.h** file in the brm-sdk pod. While it is possible to add custom fields directly to **pin_flds.h**, you should not do so. Placing custom field definitions in the separate **cust_flds.h** file allows you to upgrade to new releases without having to edit **pin_flds.h**.

Making Custom Fields Available to Your Java PCM Applications

> **Note**
>
> Developer Center is a Java application. To ensure that custom fields are displayed properly in flists in Object Browser and Opcode Workbench, you must follow the procedures for making fields available to Java applications.

Configure one or more BRM applications to access your custom fields by doing the following:

1. Create a Java source file for each custom field.
2. Do the following for each application:
   a. Open the application's **oc-cn-helm-chart/templates/configmap_infranet_properties_*.yaml** file.
   b. Add properties for configuring your custom field and source file.
   c. Save and close the file.
3. Compile the Java source files you created in Step 1.
4. (Optional) Compress the compiled classes into a JAR file.
5. In your CLASSPATH environment variable, add the location of the JAR files or compiled Java classes.

Moving Field and Storable Class Definitions Between BRM Servers with pin_deploy

You can transport definitions for storable classes and fields from one BRM server to another using the **pin_deploy** utility. For example, you could move them from your development environment to your production environment. The **pin_deploy** utility exports storable class and

Copyright © 2021, 2026, Oracle and/or its affiliates.

field definitions to a Portal Object Definition Language (PODL) file and loads them into the BRM data dictionary.

Moving field and storable class definitions from one BRM server to another involves these high-level tasks:

1. Extracting Field and Storable Class Definitions with pin_deploy
2. Importing Field and Storable Class Definitions with pin_deploy

The **pin_deploy** utility is available on all BRM platforms, can be scripted, and can use **stdin** and **stdout**. It has several modes of operation to ensure atomic operations and consistency. It provides the following advantages:

- Streamlines the process of putting all storable class and field definitions into source code management
- Enables you to print out a storable class or field definition for review
- Reduces the possibility of damaging the BRM production database data dictionary

See "pin_deploy" in *BRM Developer's Guide* for more information about the utility's syntax and parameters.

Extracting Field and Storable Class Definitions with pin_deploy

To extract field and storable class definitions from the source BRM cloud native server:

1. To extract definitions for both fields and storable classes, add the following lines to the **oc-cn-helm-chart/config_scripts/loadme.sh** script:

> **Note**
>
> You must run **pin_deploy** from the *BRM_home*/**sys/test** directory because it must be run with the **root.0.0.0.1** role.

```
#!/bin/sh
cd BRM_home/sys/test; pin_deploy field [-cp] fieldName1 fieldName2
fieldNameN
cd BRM_home/sys/test; pin_deploy class [-mnscp] className1 className2
classNameN
exit 0;
```

where:

- *fieldNameN* specifies the name of the field to export.
- *classNameN* specifies the name of the storable class to export.
- **-m** specifies to export the storable class implementation.
- **-n** specifies to export the storable class interface.
- **-s** specifies to include all subclasses of specified storable class.
- **-c** specifies to include field definitions for all customer-defined fields within storable classes.
- **-p** specifies to include field definitions for all BRM-defined fields within storable classes.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile --
   namespace 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.

Importing Field and Storable Class Definitions with pin_deploy

To import field and storable class definitions into the destination BRM server:

> **Note**
>
> The **pin_deploy** utility cannot determine the space requirement in the BRM database. If you run out of disk space before the deployment is complete, you must manually drop the tables that were created, make more space, and try again.

1. Add up the implementation definitions (for example, **initial clause**) of the PODL files you want to import to verify that you have enough disk space. The lines start with this text:

   ```
   SQL_STORAGE =
   ```

2. Configure BRM cloud native to do the following:
   - Run **pin_deploy** in **verify** mode to determine if there are any conflicts with existing storable class and field definitions.
   - Commit the storable class and field definitions to the BRM database.

   To do so, add the following lines to the **oc-cn-helm-chart/config_scripts/loadme.sh** script:

   > **Note**
   >
   > You must run **pin_deploy** from the *BRM_home***/sys/test** directory because it must be run with the **root.0.0.0.1** role.

   ```
   #!/bin/sh
   cd BRM_home/sys/test; pin_deploy verify filename.podl
   cd BRM_home/sys/test; pin_deploy command filename.podl
   exit 0;
   ```

   where:

Copyright © 2021, 2026, Oracle and/or its affiliates.

- *filename* is the name of the PODL file that contains your storable class or field definitions. If there are multiple PODL files, separate the file names using a space as a delimiter.
- *command* is either:
  - **create** to preserve old storable class and field definitions that conflict with new ones.
  - **replace** to overwrite storable class and field definitions that conflict with existing ones.

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
   ```

   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.

In all cases, the utility imports the entire PODL file. Nothing from the file is loaded if the utility cannot load the entire file. For example, if it loads a storable class that includes custom fields, they must exist in the data dictionary or in the PODL file for the storable class to load.

Copyright © 2021, 2026, Oracle and/or its affiliates.

11

Using Logs to Monitor Components

Learn how to use logs to monitor your Oracle Communications Billing and Revenue Management (BRM) cloud native deployment.

Topics in this document:

- Overview of BRM Cloud Native Logging
- Viewing Component Logs
- Configuring Log Collection and Forwarding
- Configuring the Log Management System
- Setting the Reporting Level for Logging Messages

Overview of BRM Cloud Native Logging

BRM cloud native services write log messages to standard output and standard error. Kubernetes captures these container output streams and makes them available as pod logs. However, Kubernetes will rotate and remove log files when they get to a certain size/number, so for longer-term storage, you will want a log management system. See https://kubernetes.io/docs/concepts/cluster-administration/logging/#log-rotation in the Kubernetes documentation for information on setting the size and number of these files.

The logging process is as follows:

1. The BRM service writes log messages to standard output and standard error.
2. The container runtime captures the container's stdout/stderr, and the kubelet stores the resulting logs on the worker node under **/var/log/containers/** by default.
3. While the logs are stored in the container runtime, you can view them using Kubernetes commands such as **kubectl logs**. See "Viewing Component Logs" for information.
4. A log forwarder, such as Fluentd, collects the container logs from the worker node, parses multiline BRM log entries, enriches the records with Kubernetes metadata, and forwards them to the log management system. See "Configuring Log Collection and Forwarding" for information.
5. A log management system stores, searches, visualizes, and generates alerts for the logs. See "Configuring the Log Management System" for information.

> **Note**
>
> BRM includes Fluentd as the pre-integrated log forwarder, but also supports other log forwarders. BRM supports external log management systems, but does not provide pre-integration for a specific system.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Viewing Component Logs

You can use Kubernetes commands in Table 11-1 to view logs for BRM pods.

Table 11-1 Kubernetes Commands for Viewing BRM Component Logs

Command	Description
`kubectl get pods -n BrmNamespace`	Lists the pods in the BRM namespace.
`kubectl logs PodName -n BrmNamespace`	Displays logs for the specified BRM pod.
`kubectl logs -f PodName -n BrmNamespace`	Displays logs for the specified BRM pod and continues to stream new log messages.
`kubectl logs PodName -c ContainerName -n BrmNamespace`	Displays logs for a specific container in the specified BRM pod.

where:

- *PodName* is the name of the BRM pod.
- *ContainerName* is the name of the container in the pod.
- *BrmNamespace* is the namespace in which you deployed BRM.

Configuring Log Collection and Forwarding

To manage the logs and store them for longer time periods, configure a log forwarding system like Fluentd and a log management system.

The log forwarding system performs the following functions:

1. Collects logs from container output streams.
2. Parses logs by using configured patterns for BRM components.
3. Enriches log records with Kubernetes metadata, such as namespace, pod name, and container name.
4. Forwards logs to a log management system.

Using the Sample Fluentd Configuration Files

The BRM cloud native Helm chart provides the following sample Fluentd configuration files:

oc-cn-helm-chart/Fluentd-klog-patterns-configmap.yaml: Configures Fluentd to process Kubernetes container logs in K-log format.

oc-cn-helm-chart/Fluentd-json-patterns-configmap.yaml: Configures Fluentd to process logs in JSON format.

Each sample file includes a source configuration that reads Kubernetes container logs for the BRM namespace. In the <source> section, update the **path** and **tag** keys to refer to the BRM namespace. For more information, see https://docs.fluentd.org/configuration/config-file#list-of-directives.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Use the <match> sections in the file to configure the destination for the log files for your selected log management system, for example:

```
<match **>
  @type forward
  <server>
    host log-collector.example.com
    port 24224
  </server>
</match>
```

> **Note**
>
> Review and update these sample configuration files for your environment before you use them.

Configuring the Log Management System

> **Caution**
>
> The container runtime rotates and deletes log files on worker nodes. Configure a log management system if you need long-term log retention, search capabilities, alerting, audit records, or troubleshooting history.

Configure a log management system to manage BRM logs that your log forwarding system sends.

The log management system performs the following functions:

- Receives logs from the configured log forwarding system.
- Stores and indexes log records.
- Provides search and visualization capabilities.
- Generates alerts based on configured rules, such as warning or error patterns.
- Applies retention, access control, and compliance policies.

You can use any log management system that meets your operational, storage, security, and compliance requirements.

Configure the log management system to receive logs from the log forwarding system.

Setting the Reporting Level for Logging Messages

By default, BRM components report error messages, and BRM applications report both error and warning messages (level 2). The four levels of error reporting are:

- **0** = No logging
- **1** = Log error messages only
- **2** = Log error messages and warnings

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **3** = Log error, warning, and debugging messages

> **Note**
>
> To avoid performance degradation, use log level 3 only for debugging.

A common implementation is to set the reporting level to 2 and use the configured log management system to detect warning messages and generate alerts.

To change the severity level for logging:

1. Open your **override-values.yaml** file for **oc-cn-helm-chart**.
2. Edit the log level entry for the BRM component or application. For information about product-specific logging configuration settings, see "Configuring BRM Server and PDC Services" in *BRM Cloud Native Deployment Guide*.
3. Save and close the file.
4. Stop and restart the component or application.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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:

- Monitoring BRM Cloud Native Services
- Monitoring and Autoscaling Business Operations Center Cloud Native
- Monitoring and Autoscaling Billing Care Cloud Native
- Monitoring and Autoscaling Pipeline Configuration Center Cloud Native
- Monitoring BRM REST Services Manager Cloud Native
- Tracing BRM REST Services Manager Cloud Native
- Tracing Opcode Processes

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".

Copyright © 2021, 2026, Oracle and/or its affiliates.

12

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
- 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 or Prometheus Operator 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 Compatibility Matrix*.

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **Note**
>
> Standalone Prometheus does not automatically add Kubernetes metadata labels such as pod and namespace to collected metrics. Because the provided Grafana dashboards use these labels in dashboard queries, add the required relabeling rules to the **prometheus.yaml** file when using standalone Prometheus.
>
> When using Prometheus Operator, ensure that the ServiceMonitor CRDs are installed and that ServiceMonitor resources exist for CM and dm-oracle metrics. Also verify that the Prometheus **serviceMonitorSelector** matches the ServiceMonitor labels so the targets are discovered and scraped.

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

   For the list of compatible software versions, see *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 to view 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**, ensure that the **ocbrm.cm.deployment.perflib_enabled** key is set to **true**. (The default is **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_PROCESS_METRICS** key to **true**.

4. To collect metrics for the dm-oracle pod, do the following:

Copyright © 2021, 2026, Oracle and/or its affiliates.

- In your **override-values.yaml** file for **oc-cn-helm-chart**, ensure that the **ocbrm.dm_oracle.deployment.perflib_enabled** key is set to **true**. (The default is **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 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**.

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

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

7. 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 12-1 for more information.

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

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile --namespace 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.

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

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

Copyright © 2021, 2026, Oracle and/or its affiliates.

- Oracle DM shared memory, front-end processes, and back-end processes
- 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: true
    jmx_exporter:
        enable: true
ocbrm:
    cm:
        deployment:
            perflib_enabled: true
    dm_oracle:
        deployment:
            perflib_enabled: true
    perflib:
        deployment:
            persistPerflibLogs: 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 12-1 describes the environment variables you can use to configure Perflib for the CM and DMs.

Table 12-1 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.
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**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 12-1 (Cont.) Perflib Environment Variables

Environment Variable	Description
PERFLIB_MAX_LOG_SIZE	The maximum number of opcodes that can be logged in one log file. You can use this to prevent huge log files if detailed tracing is used for long periods. • **0**: Creates a single file with no limits. This is the default. • *Number*: Defines the maximum number of opcodes to log before opening a new file.
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 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 Perflib uses 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 are 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 Linux time number). • **%T**: Substituted with the current time stamp (as a *YYYYMMDDHHMMSS* string). The default is **/oms_logs/perflib_data.dat**.
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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 12-1 (Cont.) Perflib Environment Variables

Environment Variable	Description
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. 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.
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 on different objects. For PCM_OP_EXEC_SPROC, the latest versions of Perflib will record the name of the stored procedure that was run. • **0**: Do not collect object type data. • **1**: Collect object type data and record it in real time or batch trace files. This is the default.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 12-1 (Cont.) Perflib Environment Variables

Environment Variable	Description
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 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_PROCESS_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.
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 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 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 to display the metric data for your BRM cloud native services.

Alternatively, you can use the sample dashboards included in the **oc-cn-docker-files-16.26.0.*x*.*x*.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.

Table 12-2 describes each sample dashboard.

Table 12-2 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-dm-kafka-dashboard.json	Allows you to view JVM-related metrics for the Kafka DM.
ocbrm-cm-dashboard.json	Allows you to view CPU and opcode-level metrics for the CM.
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-esp-dashboard.json	Allows you to view metrics for the Event Stream Processor (ESP).

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 12-2 (Cont.) Sample Grafana Dashboards

File Name	Description
ocbrm-overview-dashboard.json	Allows you to view metrics for BRM services at the pod, container, network, and input-output level.
ocbrm-performance-dashboard.json	Allows you to view performance-related metrics for BRM cloud native.
ocbrm-rel-dashboard.json	Allows you to view JVM-related metrics for Rated Event (RE) Loader.
ocbrm-rem-dashboard.json	Allows you to view metrics for Rated Event Manager (REM).
ocbrm-remtable-dashboard.json	Allows you to view table metrics for Rated Event Manager (REM).
ocbrm-wsm-helidon-server-dashboard.json	Allows you to view metrics for Web Services Manager.

> **Note**
>
> For the sample dashboard to work properly, the data source 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 12-3 lists the metrics in this group.

Table 12-3 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
brm_opcode_errors_total	Counter	The total number of errors when executing a BRM opcode.	cm dm-oracle
brm_opcode_exec_time_total	Counter	The total time taken to run a BRM opcode.	cm dm-oracle
brm_opcode_user_cpu_time_total	Counter	The total CPU time taken to run the BRM opcode in user space.	cm dm-oracle
brm_opcode_system_cpu_time_total	Counter	The total CPU time taken to run the BRM opcode in OS Kernel space.	cm dm-oracle

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 12-3 (Cont.) BRM Opcode Metrics

Metric Name	Metric Type	Metric Description	Pod
brm_opcode_records_total	Counter	The total number of records returned by the BRM opcode execution.	cm dm-oracle
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 watermark 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
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)

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 12-3 (Cont.) BRM Opcode Metrics

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

Copyright © 2021, 2026, Oracle and/or its affiliates.

13

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
- Setting Up Monitoring and Autoscaling in Business Operations Center
- Sample Prometheus Alert Rules for Business Operations Center

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 MBean metrics, such as memory usage and session count, required to monitor and maintain the Business Operations Center application.
- **Prometheus**: Use this open-source toolkit to scrape Business Operations Center metric data from WebLogic Monitoring Exporter and store it in a time-series database. It can also be used to scale up or 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 all Business Operations Center metric data that is stored in Prometheus on a graphical dashboard.

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 Compatibility Matrix*.
2. Install Grafana. See "Install Grafana" in the Grafana documentation.

Copyright © 2021, 2026, Oracle and/or its affiliates.

For the list of compatible software versions, see *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 to display 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 *WebLogic Server MBean Reference* 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 --
namespace 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.

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
     ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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 13-1 lists the possible values for each parameter.

Table 13-1 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**.
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 --
namespace BrmNameSpace
```

The webhook application starts waiting for alerts from Prometheus Alertmanager.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
      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 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 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**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 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 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
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
    rules:
  - alert: clusterScaleUp
    annotations:
      message: WLS cluster has less than 2 running servers for more than 2
minutes.
    expr: sum by(weblogic_domainUID)
(up{serviceType="SERVER",weblogic_clusterName="cluster-1",weblogic_domainUI
D="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_domainUI
D="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:
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
    - 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 to display your Business Operations Center metric data. Alternatively, you can use the sample dashboard JSON model included in the **oc-cn-docker-files-16.26.0.*x*.*x*.tgz** package.

> **Note**
>
> For the sample dashboard to work properly, the data source 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=~".+",weblogic
_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=~".+",weblogic
_domainUID="boc-domain",weblogic_serverName=~".+"}[2m])) * 100 > 70 and on()
100 -
avg(avg_over_time(wls_jvm_heap_free_percent{weblogic_clusterName=~".+",weblogi
c_domainUID="boc-domain",weblogic_serverName=~".+"}[2m])) > 70
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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=~".+",weblogic
_domainUID="boc-domain",weblogic_serverName=~".+"}[2m])) * 100 < 40 and on()
100 -
avg(avg_over_time(wls_jvm_heap_free_percent{weblogic_clusterName=~".+",weblogi
c_domainUID="boc-domain",weblogic_serverName=~".+"}[2m])) < 40 and on() sum
by(weblogic_domainUID)
(up{weblogic_clusterName="cluster-1",weblogic_domainUID="boc-domain"}) == 5
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

14

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 session count, required to monitor and maintain the Billing Care and Billing Care REST API applications.
- **Prometheus**: Use this open-source toolkit to scrape metric data from WebLogic Monitoring Exporter and store it in a time-series database. It can also be used to scale up or 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 all Billing Care and Billing Care REST API metric data stored in Prometheus on a graphical dashboard.

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 Compatibility Matrix*.
2. Install Grafana. See "Install Grafana" in the Grafana documentation.

   For the list of compatible software versions, see *BRM Compatibility Matrix*.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 to display 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 is **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 format. For the list of possible MBeans, see *WebLogic Server MBean Reference* in the Oracle WebLogic Server documentation.
   - For Billing Care: **ocbc.bc.monitoring.queries**
   - For the Billing Care REST API: **ocbc.bcws.monitoring.queries**

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **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 --namespace 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.

   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**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **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.

   > **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 14-1 lists the possible values for each parameter.

Table 14-1 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**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 14-1 (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 --namespace 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"}
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
  - 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* are 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
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
            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
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
        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 servers for more than
2 minutes.
      expr: sum by(weblogic_domainUID)
(up{serviceType="SERVER",weblogic_clusterName="cluster-1",weblogic_domainUI
D="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_domainUI
D="billingcare-domain"}) > 3
      for: 2m
      labels:
        domain_uid: billingcare-domain
        severity: critical
    - alert: clusterScaleUp
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
      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_domainUI
D="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_domainUI
D="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'
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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-16.26.0.*x*.*x*.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 data source 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_domain
UID="bcws-domain",weblogic_serverName=~".+"}[2m]))*100> 70
```

- For a Billing Care domain:

```
avg(avg_over_time(wls_jvm_process_cpu_load{weblogic_clusterName=~".+",weblogic_domain
UID="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
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

- For a Billing Care domain:

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

- For a Billing Care domain:

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


Copyright © 2021, 2026, Oracle and/or its affiliates.

15

Monitoring and Autoscaling Pipeline Configuration Center Cloud Native

Learn how to use external applications, such as Prometheus and Grafana, to monitor and autoscale Oracle Communications Billing and Revenue Management (BRM) Pipeline Configuration Center in a cloud native environment.

Topics in this document:

- About Monitoring and Autoscaling Pipeline Configuration Center Cloud Native
- Setting Up Monitoring and Autoscaling in Pipeline Configuration Center
- Sample Prometheus Alert Rules for Pipeline Configuration Center

About Monitoring and Autoscaling Pipeline Configuration Center Cloud Native

You set up the monitoring of Pipeline Configuration 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 run-time information from Pipeline Configuration Center cloud native and then expose the metric data in Prometheus format. It exposes different WebLogic MBean metrics, such as memory usage and session count, required to monitor and maintain the Pipeline Configuration Center application.
- **Prometheus**: Use this open-source toolkit to scrape Pipeline Configuration Center metric data from WebLogic Monitoring Exporter and store it in a time-series database. It can also be used to scale up or down your Pipeline Configuration 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 all Pipeline Configuration Center metric data that is stored in Prometheus on a graphical dashboard.

Setting Up Monitoring and Autoscaling in Pipeline Configuration Center

To set up monitoring and autoscaling in Pipeline Configuration 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 Compatibility Matrix*.
2. Install Grafana. See "Install Grafana" in the Grafana documentation.

Copyright © 2021, 2026, Oracle and/or its affiliates.

For the list of compatible software versions, see *BRM Compatibility Matrix*.

3. Configure WebLogic Monitoring Exporter to scrape metric data from Pipeline Configuration Center in your cloud native environment. See "Configuring WebLogic Monitoring Exporter to Scrape Metric Data".
4. Configure the Prometheus webhook to autoscale the Pipeline Configuration 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 Pipeline Configuration Center".
   - Prometheus Operator. See "Configuring Prometheus Operator for Pipeline Configuration Center".
6. Configure Grafana to display Pipeline Configuration Center metric data. See "Creating Grafana Dashboards for Pipeline Configuration Center".

Configuring WebLogic Monitoring Exporter to Scrape Metric Data

You configure WebLogic Monitoring Exporter to scrape metric data for Pipeline Configuration 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 Pipeline Configuration Center cloud native:

1. Open your **override-values.yaml** file for **oc-cn-helm-chart**.
2. Set the **ocpcc.pcc.monitoring.isEnabled** key to **true**.
3. Set the **ocpcc.pcc.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 Pipeline Configuration Center. To do so, set the **ocpcc.pcc.monitoring.queries** key to the full array of WebLogic Server MBeans to monitor in YAML structure. For the list of possible MBeans, see *WebLogic Server MBean Reference* 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 **ocpcc.pcc.monitoring** as needed.

Copyright © 2021, 2026, Oracle and/or its affiliates.

For information about the other keys under **ocpcc.pcc.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 --
namespace 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.

WebLogic Monitoring Exporter is started within the Pipeline Configuration Center WebLogic Server pod and begins scraping metric data for Pipeline Configuration 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 Pipeline Configuration 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 Pipeline Configuration 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
     ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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 15-1 lists the possible values for each parameter.

Table 15-1 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**.
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=pcc-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 --
   namespace BrmNameSpace
   ```

The webhook application starts waiting for alerts from Prometheus Alertmanager.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Example: Configuring webhook to Autoscale Pipeline Configuration Center Pods

The following shows sample **override-values.yaml** entries for setting up the webhook application to perform autoscaling on your Pipeline Configuration 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 Pipeline Configuration Center

To configure a standalone version of Prometheus for Pipeline Configuration Center cloud native:

1. Open your **override-values.yaml** file for Prometheus.
2. Configure Prometheus to collect your Pipeline Configuration 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: pcc-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]
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
      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 Pipeline Configuration 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 Pipeline Configuration Center-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 Pipeline Configuration Center".

```
serverFiles:
  alerts:
    groups:
      - name: node_rules
        rules:
          - alert: clusterScaleUp
            for: 2m
            expr: sum by(weblogic_domainUID, weblogic_clusterName)
(up{weblogic_domainUID="pcc-domain"}) < 2
            labels:
              domain_uid: pcc-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="pcc-domain"}) > 3
            labels:
              domain_uid: pcc-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**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 Pipeline Configuration Center

To configure Prometheus Operator for Pipeline Configuration 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 Pipeline Configuration 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 Pipeline Configuration Center-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 Pipeline Configuration Center".

```
## Provide custom recording or alerting rules to be deployed into the
cluster.
##
additionalPrometheusRulesMap:
  - rule-name: Custom-rule
    groups:
  - name: custom-alert.rules
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
    rules:
  - alert: clusterScaleUp
    annotations:
      message: WLS cluster has less than 2 running servers for more than 2
minutes.
    expr: sum by(weblogic_domainUID)
(up{serviceType="SERVER",weblogic_clusterName="cluster-1",weblogic_domainUI
D="pcc-domain"}) < 2
    for: 2m
    labels:
      domain_uid: pcc-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_domainUI
D="pcc-domain"}) > 3
    for: 2m
    labels:
      domain_uid: pcc-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:
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
    - 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 Pipeline Configuration Center

Create a dashboard in Grafana to display your Pipeline Configuration Center metric data. Alternatively, you can use the sample dashboard JSON model included in the **oc-cn-docker-files-16.26.0.*x*.*x*.tgz** package.

> **Note**
>
> For the sample dashboard to work properly, the data source name for the WebLogic Domain must be **Prometheus**.

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

Sample Prometheus Alert Rules for Pipeline Configuration Center

You can use custom expressions for your Prometheus alert rules when setting up autoscaling in Pipeline Configuration 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=~".+",weblogic
_domainUID="pcc-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="pcc-
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=~".+",weblogic
_domainUID="pcc-domain",weblogic_serverName=~".+"}[2m])) * 100 > 70 and on()
100 -
avg(avg_over_time(wls_jvm_heap_free_percent{weblogic_clusterName=~".+",weblogi
c_domainUID="pcc-domain",weblogic_serverName=~".+"}[2m])) > 70
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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=~".+",weblogic
_domainUID="pcc-domain",weblogic_serverName=~".+"}[2m])) * 100 < 40 and on()
100 -
avg(avg_over_time(wls_jvm_heap_free_percent{weblogic_clusterName=~".+",weblogi
c_domainUID="pcc-domain",weblogic_serverName=~".+"}[2m])) < 40 and on() sum
by(weblogic_domainUID)
(up{weblogic_clusterName="cluster-1",weblogic_domainUID="pcc-domain"}) == 5
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

16

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 all BRM REST Services Manager metric data stored in Prometheus on a graphical dashboard.

  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
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
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 --namespace 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 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:
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
    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-16.26.0.0.0
    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, **16.26.0.0.0**

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 to display 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. Alternatively, you can use the sample dashboard JSON model that is included in the **oc-cn-docker-files-16.26.0.*x*.*x*.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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 --namespace
  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 --
  namespace 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Use a browser to send HTTP/HTTPS requests to the endpoints listed in Table 16-1, where *hostname* and *port* are the URL and port for your BRM REST Services Manager server.

Table 16-1 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",
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
                "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

Copyright © 2021, 2026, Oracle and/or its affiliates.

17

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
- Securing Communication with Zipkin
- Enabling Tracing in BRM REST Services Manager

About BRM REST Services Manager Tracing

You can trace the flow of REST API calls made to BRM REST Services Manager using Zipkin, 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 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**:

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **isTracingEnabled**: Set this to **true**.
- **zipkinHostName**: Set this to the host name 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 --
   namespace 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

18

Tracing Opcode Processes

Learn how to enable selective opcode tracing in a cloud native Oracle Communications Billing and Revenue Management (BRM) environment, as part of the initiative to provide more granularity in error-code, class, and location reporting by Perflib.

Topics in this document:

- Enabling Selective Opcode Tracing

Enabling Selective Opcode Tracing

Use the selective opcode tracing feature to control which opcodes are traced for flist and summary tracing. By default, all opcodes are traced. You can enable selective opcode tracing in a cloud native environment using a configuration file or during runtime using the **pstatus** application.

To enable selective opcode tracing in a cloud native environment:

1. Open the CM and Oracle DM Perflib environment files in a text editor:
   - CM: **configmap_env_perf.yaml**
   - Oracle DM: **configmap_env_perf_dm.yaml**
2. Set the **PERFLIB_VAR_TRACE_OPCODES** parameter to a comma-separated list of opcodes to trace when using Perflib's full or summary flist tracing mode:

   ```
   PERFLIB_VAR_TRACE_OPCODES: "opcodeNameOrNumber"
   ```

   where *opcodeNameOrNumber* is either the name or number of a BRM opcode. If you list multiple opcodes, use a comma as a delimiter. If the parameter is empty or missing, Perflib traces all opcodes by default.

   For example, this entry specifies to use flist tracing for only the PCM_OP_CUST_COMMIT_CUSTOMER and PCM_OP_SUBSCRIPTION_PURCHASE_DEAL (108) opcodes:

   ```
   PERFLIB_VAR_TRACE_OPCODES: "PCM_OP_CUST_COMMIT_CUSTOMER,108"
   ```
3. Run the **helm upgrade** command to update the release.
4. Restart the CM and Oracle DM services to initialize the new configuration.

To enable selective opcode tracing at runtime, enter the CM or Oracle DM pod directly using a shell and run the **pstatus** application.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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:

- Integrating with Your BRM Cloud Native Deployment
- Setting Up Payment Processing with Paymentech

Copyright © 2021, 2026, Oracle and/or its affiliates.

19

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 Analytics Publisher.

Topics in this document:

- Integrating with Thick Clients
- Configuring TLS Certificates for BRM Cloud Native
- Configuring an External Load Balancer for Connection Manager SSL/TLS Connections
- Integrating with JCA Resource Adapter
- Integrating with Kafka Servers
- Integrating with Oracle Analytics Publisher

Integrating with Thick Clients

You can integrate BRM cloud native with thick clients that connect to Connection Manager (CM). When you enable TLS, provide the thick client with a customer-managed TLS wallet that contains the certificates required for the configured TLS mode.

To integrate a thick client with CM:

1. In the **override-values.yaml** file for **oc-cn-helm-chart**, configure the CM TLS settings:
   - **ocbrm.cm.serviceFqdn**: Set this key to the Subject Alternative Name (SAN) in the TLS certificate, such as dns:node1.brm.com.
   - **ocbrm.isSSLEnabled**: Set this to **true**.
   - **ocbrm.sslMode**: Set this key to the TLS authentication mode:
     - **ONE_WAY**: The thick client validates the certificate presented by CM.
     - **TWO_WAY**: The thick client and CM authenticate each other by presenting and validating certificates.
2. Provide a TLS wallet for the thick client.

   Configure the wallet according to the TLS mode:
   - For one-way TLS, include the trusted certificate authority (CA) certificates that the thick client uses to validate the certificate presented by CM.
   - For two-way TLS, include the thick client's identity certificate and private key in addition to the trusted CA certificates that it uses to validate the CM certificate.

   Use certificates issued by a CA approved by your organization. Manage the thick-client wallet independently of the CM wallet. Do not copy the CM TLS wallet to the thick client.
3. If you configure two-way TLS and the thick client connects to CM over PCP, enable PCP client authentication in the client's Infranet configuration:

   ```
   infranet.pcp.ssl.client_auth=true
   ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

For more information, see "Configuring BRM Client Applications for Two-Way TLS".

4. If thick clients connect to CM through an external load balancer, ensure that the client wallet trusts the certificate presented at the TLS endpoint.

   The host name or IP address in the certificate must match the host name or IP address that the thick client uses to connect to the load balancer. See "Configuring an External Load Balancer for Connection Manager SSL/TLS Connections".

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

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile --
   namespace 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.

Configuring TLS Certificates for BRM Cloud Native

You can secure TLS connections between BRM components and external applications by using Oracle wallets.

BRM cloud native container images do not include populated TLS wallets, private keys, or sample CA certificates. Before you enable TLS, obtain the required certificates from a certificate authority (CA) approved by your organization and create the Oracle wallets required for your deployment. You are responsible for managing and maintaining your certificates and wallets, including renewing and rotating certificates as needed.

How BRM Selects the TLS Wallet

When TLS is enabled, BRM selects the TLS wallet from the first available source in the following order:

1. Component-specific **extUserSSLWalletSecret**
2. **ocbrm.extCustomSSLWalletSecret**
3. The **custom-ssl-wallet** secret created from **brm_custom_wallet/cwallet.sso** in the Helm chart

If none of these wallet sources is available when TLS is enabled, the Helm installation fails.

> **Note**
>
> The **ocbrm.customSSLWallet** key is no longer used. Do not include this key in your **override-values.yaml** file.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Preparing the Required TLS Wallets

Before configuring TLS in your BRM cloud native deployment, obtain the CA certificates and identity certificates required by your organization's security policies and create the appropriate Oracle wallets. Use certificates issued by a certificate authority (CA) approved by your organization.

When you create a certificate for a BRM component, include all DNS names that clients use to connect to the component in the certificate's Subject Alternative Name (SAN). For example, if clients connect to the Connection Manager (CM) by using both its Kubernetes service name and an external host name, include both DNS names in the CM certificate.

The certificates required in each wallet depend on the TLS mode:

- **One-way TLS**: The client must trust the certificate presented by the server. Ensure that the client wallet contains the appropriate trusted certificates and that the server wallet contains the server's identity certificate and private key. A BRM component that acts only as a TLS client does not require an identity certificate or private key.
- **Two-way TLS**: both the client and server authenticate each other. Ensure that the wallets for both endpoints contain the appropriate trusted certificates, identity certificates, and private keys. A BRM component that acts as a TLS client therefore also requires its own identity certificate and private key.

Configuring TLS Certificates

To configure TLS certificates for your BRM cloud native deployment:

1. Ensure that each wallet contains the certificates required for the component and the configured TLS mode:
   - For one-way TLS, a component acting as a server requires its identity certificate and private key. A component acting as a client requires the trusted certificates needed to validate the server certificate.
   - For two-way TLS, both the client and server require the appropriate identity certificate, private key, and trusted certificates.
2. Provide the TLS wallets required by your BRM components.

   You can provide TLS wallets by using pre-created Kubernetes secret or, where supported, by providing **brm_custom_wallet/cwallet.sso** in the Helm chart.

   If a component requires its own identity certificate, provide a separate wallet for that component so that its pod receives only the certificate and private key that it requires.

   BRM supports component-specific and shared TLS wallet sources and selects the wallet according to a defined precedence order. See "How BRM Selects the TLS Wallet" for information about the supported wallet sources and their selection order.
3. Open your **override-values.yaml** file for **oc-cn-helm-chart**.
4. Configure TLS by setting the following keys:
   - **ocbrm.isSSLEnabled**: Set this to **true**.
   - **ocbrm.sslMode**: Specify the TLS authentication mode:
     - **ONE_WAY**: The client validates the certificate that the server presents.
     - **TWO_WAY**: The client and server authenticate each other by presenting and validating certificates. For BRM client applications that connect to CM over PCP,

Copyright © 2021, 2026, Oracle and/or its affiliates.

also set **infranet.pcp.ssl.client_auth=true**. See "Configuring BRM Client Applications for Two-Way TLS".

- **ocbrm.cmSSLTermination**: Set this key to **true** if you want CM to terminate TLS and you do not want to use TLS between CM and downstream DMs and EMs.

5. Configure the TLS wallet source for each BRM component that requires a wallet.

   To use a component-specific Kubernetes Secret, set the component's **extUserSSLWalletSecret** key to the name of the Secret.
   For example, to specify a TLS wallet Secret for CM:

   ```
   ocbrm:
      isSSLEnabled: true
      sslMode: "ONE_WAY"
      cm:
         extUserSSLWalletSecret: CmTLSWalletSecret
   ```

   Set the corresponding **extUserSSLWalletSecret** key for each additional component that requires its own TLS wallet.

   To use a shared external Kubernetes Secret, set **ocbrm.extCustomSSLWalletSecret** to the name of the Secret.

   Alternatively, where supported, provide **brm_custom_wallet/cwallet.sso** in the Helm chart. The Helm chart creates the **custom-ssl-wallet** Secret from this wallet.

   > **Note**
   >
   > If you enable TLS and do not provide a supported wallet source, the Helm installation fails.

6. Install BRM cloud native services by entering the following command from the **helmcharts** directory:

   > **Note**
   >
   > If BRM cloud native is already installed, run the **helm upgrade** command instead.

   ```
   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.
   - *BrmNameSpace* is the namespace in which to create BRM Kubernetes objects.
   - *OverrideValuesFile* is the file name and path of your **override-values.yaml** file.

7. Verify that the BRM pods are running and that the TLS connections are succeed.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Before you install or upgrade BRM cloud native, you can verify the contents of an Oracle wallet by running:

```
orapki wallet display -wallet WalletPath -summary
```

where *WalletPath* is the path to the Oracle wallet.

Verify that the wallet contains the identity and trusted certificates required for the configured TLS mode.

For information about integrating clients outside the Kubernetes cluster with the CM, see "Integrating with Thick Clients".

Configuring BRM Client Applications for Two-Way TLS

When you set **ocbrm.sslMode** to **TWO_WAY**, BRM clients and servers authenticate each other by using TLS certificates. Configure each BRM client application that uses PCP to present its client certificate during the TLS handshake.

To configure a BRM client application for two-way TLS:

1. Ensure that the application's TLS wallet contains:
   - The application's identity certificate and private key
   - The trusted certificates required to validate the certificate presented by the BRM server
2. Enable PCP client authentication in the application's Infranet configuration:

   ```
   infranet.pcp.ssl.client_auth=true
   ```
3. Configure the application to use the TLS wallet that contains the required certificates and private key.
4. Restart or redeploy the application so that the configuration changes take effect.

> **Note**
>
> For one-way TLS, do not set **infranet.pcp.ssl.client_auth** to **true**. A BRM component that acts only as a client needs the trusted certificates required to validate the server certificate. The component does not need a client identity certificate or private key.

Configuring an External Load Balancer for Connection Manager SSL/TLS Connections

You can configure clients to connect to CM through an external load balancer. In this configuration, the client connects to the load balancer, and the load balancer forwards the connection to the Kubernetes worker node on which CM is available.

When CM is configured to use SSL or TLS, a handshake error can occur if the load balancer or the BRM client is not configured with the required certificate information.

You can configure the load balancer for SSL/TLS in one of these ways:

- Configure SSL/TLS between the BRM client and the load balancer only.

Copyright © 2021, 2026, Oracle and/or its affiliates.

In this configuration, the BRM client connects to the load balancer using SSL/TLS. The load balancer connects to CM without SSL.

Ensure that the certificate configured on the load balancer meets one of these requirements:

- The certificate is issued by a certificate authority (CA) that is trusted in the Oracle wallet used by the BRM client.
- The certificate configured on the load balancer is added as a trusted certificate in the Oracle wallet used by the BRM client.

Also ensure that the host name or IP address in the certificate matches the load balancer host name or IP address that the BRM client uses to connect.

- Configure SSL/TLS between the BRM client and the load balancer, and between the load balancer and CM.

  In this configuration, the BRM client connects to the load balancer using SSL/TLS. The load balancer must also start an SSL connection to CM.

  Configure the load balancer to connect to the CM back end using SSL instead of a plain TCP connection. Also configure the load balancer with the certificate information that it requires to trust the CM certificate.

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

> **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".

To deploy JCA Resource Adapter in your BRM cloud native deployment:

1. Enable the brm-sdk pod by setting these keys in your **override-values.yaml** file for **oc-cn-helm-chart**:

   ```
   brm_sdk:
      isEnabled: true
      deployment:
         imageName: brm_sdk
         imageTag: 16.26.0.x.x
      pvc:
         storage: 50Mi
   ```

2. Run the **helm upgrade** command for **oc-cn-helm-chart**:

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

3. Find the name of the brm-sdk pod:

Copyright © 2021, 2026, Oracle and/or its affiliates.

```
kubectl get pods -n BrmNameSpace | grep brm-sdk
```

You should see something similar to this:

```
NAME                       READY    STATUS     RESTARTS    AGE
brm-sdk-f67b95777-bf8j5    1/1      Running    0           18m
```

4. Enter the brm-sdk pod:

   ```
   kubectl exec -n BrmNameSpace -it BrmSdkPodName bash
   ```

   where *BrmSdkPodName* is the name of the pod from step 3.

   For example:

   ```
   kubectl exec -n MyNameSpace -it brm-sdk-f67b95777-bf8j5 bash
   ```

5. Go to the **apps/brm_integrations/jca_adapter/** directory and copy the **OracleBRMJCA15Adapter.rar** file to a new staging directory:

   ```
   cd apps/brm_integrations/jca_adapter/
   mkdir staging
   cp OracleBRMJCA15Adapter.rar staging/
   ```

6. Go to your staging directory and extract the files from the **OracleBRMJCA15Adapter.rar** archive file:

   ```
   cd staging
   jar xvf OracleBRMJCA15Adapter.rar
   ```

7. Update the following parameters in your **META-INF/weblogic-ra.xml** file:

   - **ConnectionString**: Set this to the string for connecting to the BRM service in the format:

     protocol host port

     For example: **ip cm 11960**.

   - **Password**: Set this to the password for the BRM **root** user.
   - **JavaPcmSSL**: Set this to **true** if SSL is enabled for BRM.
   - **SslWalletLocation**: Set this to the location of the Oracle wallet that contains the BRM client TLS certificate. This can be copied from the BRM installation.

8. From your staging directory, delete the **OracleBRMJCA15Adapter.rar** file and then build a new archive file using the updated **META-INF/weblogic-ra.xml** file:

   ```
   rm OracleBRMJCA15Adapter.rar
   jar cvf OracleBRMJCA15Adapter.rar  .
   ```

9. Copy the new **OracleBRMJCA15Adapter.rar** into the brm-sdk PVC storage class:

   ```
   cp OracleBRMJCA15Adapter.rar /oms/ext/
   ```

10. Copy the client wallet to the brm-sdk PVC storage class:

    ```
    cp -r /oms/wallet/client/ /oms/ext/wallet/
    ```

11. Exit the brm-sdk pod.

12. Retrieve the name of the PVC volume for the brm-sdk pod:

    ```
    kubectl get pvc -n BrmNameSpace | grep brm-sdk
    ```

    You should see something similar to this:

    ```
    NAME      STATUS   VOLUME                                     CAPACITY   ACCESS
    MODES   STORAGE CLASS  AGE
    ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
brm-sdk   Bound    pvc-094feae0-4d11-4887-83a0-b47a0fc6a3f4   50Mi
RWX            myclass        24h
```

13. Copy the **OracleBRMJCA15Adapter.rar** archive file from the PVC to the working directory

    cp *NfsMountPath/BrmNameSpace/BrmSdkPvc/***OracleBRMJCA15Adapter.rar** **.**

    where *BrmSdkPvc* is the name of the PVC volume from step 12.

    For example:

    ```
    cp -r /mnt/oke_test/MyNameSpace/pvc-094feae0-4d11-4887-83a0-b47a0fc6a3f4/
    OracleBRMJCA15Adapter.rar .
    ```

14. Copy the client wallet from the PVC to the working directory:

    cp -r *NfsMountPath/BrmNameSpace/BrmSdkPvc/WalletFolder* **.**

    For example:

    ```
    cp -r /mnt/oke_test/MyNameSpace/pvc-094feae0-4d11-4887-83a0-b47a0fc6a3f4/wallet .
    ```

15. Copy the client wallet to the appropriate path and deploy **OracleBRMJCA15Adapter.rar** on AIA.

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 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**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **ocbrm.dm_kafka.kafkaAsyncMode**: Specify whether to use asynchronous mode (**true**), in which the Kafka DM records all business events that fail to publish to a log file. In synchronous mode (**false**), the Kafka DM returns errors to BRM when a business event fails to publish to the Kafka server.
- **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**.

  > **Note**
  >
  > Set **ocbrm.dm_kafka.deployment.poolSize** to the thread pool size that the EAI Java Server (JS) uses. A mismatch can lead to thread exhaustion or connection failures over time. The CM log can contain intermittent PIN_ERR_DM_CONNECT_FAILED errors. The connection failures can cause order processing to fail. For more information, see "Configuring Thread Pooling for the Kafka DM" in *BRM Developer's Guide*.

- **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_ifw_kafka_sync_merged.xml**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 --namespace 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.

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
        kafkaAsyncMode: 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_ifw_sync.xml
```

Integrating with Oracle Analytics Publisher

You can optionally integrate your BRM cloud native deployment with invoicing software such as Oracle Analytics 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:

Copyright © 2021, 2026, Oracle and/or its affiliates.

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

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

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

   The steps for installing the package on BRM cloud native are similar to those on BRM on-premises. For more information, see "Installing the BRM-Oracle Analytics 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.

   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 Oracle Analytics Publisher instance, which Billing Care uses to show invoices.
   - **ocbc.bc.configEnv.bipUserId**: The name of the user with access to the Oracle Analytics Publisher instance for viewing invoices from Billing Care.
   - **ocbc.bc.secretVal.bipPassword**: The Base64-encoded password for the Oracle Analytics Publisher user.
   - **ocbc.bcws.configEnv.bipUrl**: The URL for **PublicReportService_v11** from your 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 Oracle Analytics Publisher instance for accessing invoices from the Billing Care REST API.
   - **ocbc.bcws.secretVal.bipPassword**: The Base64-encoded password for the Oracle Analytics 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.

   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 is installed.
   - **bipPort**: The port number for Oracle Analytics Publisher.
   - **bipUsername**: The name of the user with access to the Oracle Analytics Publisher instance.
   - **bipPassword**: The Base64-encoded password for the Oracle Analytics 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **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 --
   namespace 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**).

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **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**

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 --
    namespace 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 --namespace BrmNameSpace
    ```

When configuring Oracle Analytics 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

After integration is complete, you can generate your customers' invoices in Oracle Analytics 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 templates by running the **pin_inv_doc_gen** utility through a brm-apps job. See "Running Applications and Utilities Through brm-apps Jobs".

Copyright © 2021, 2026, Oracle and/or its affiliates.

20

Setting Up Payment Processing with Paymentech

Learn how Oracle Communications Billing and Revenue Management (BRM) cloud native integrates with Paymentech to process your customers' credit card and direct debit payments.

Topics in this document:

- About Setting Up Payment Processing with Paymentech
- Using SFTP for Batch Payment Transactions
- Using TCP/IP for Batch Payment Transactions
- Configuring TCP/IP for Online Payment Transactions
- Configuring the Paymentech Simulator for Testing

About Setting Up Payment Processing with Paymentech

Paymentech validates, authorizes, and processes credit card and debit card payments from your customers. For more information, see "Processing Credit Card and Debit Card Payments with Paymentech" in *BRM Configuring and Collecting Payments*.

To process payments with Paymentech in BRM cloud native:

1. Configure **dm_fusa** to send batch payment transactions to Paymentech using either SFTP or TCP/IP:
   - To use SFTP, see "Using SFTP for Batch Payment Transactions".
   - To use TCP/IP, see "Using TCP/IP for Batch Payment Transactions".
2. Configure **dm_fusa** to use TCP/IP for online payment transactions. See "Configuring TCP/IP for Online Payment Transactions".
3. Configure merchant accounts as described in "Setting Up Merchant Accounts" in *BRM Configuring and Collecting Payments*.
4. Set up BRM to process PINless debit payments as described in "Configuring PINless Direct Debit Payment Processing" in *BRM Configuring and Collecting Payments*.
5. Test payment processing using the Paymentech Simulator. See "Configuring the Paymentech Simulator for Testing".

Using SFTP for Batch Payment Transactions

You can configure **dm_fusa** to use SFTP for sending batch payment transactions to the Paymentech payment processor. Batch transactions include batch payments, multiple verifications, multiple authorizations, deposits, and refunds.

To use SFTP for batch payment transactions:

1. Configuring Authentication for SFTP Batch Transactions

Copyright © 2021, 2026, Oracle and/or its affiliates.

2. Configuring dm_fusa with SFTP Settings

Configuring Authentication for SFTP Batch Transactions

To configure authentication for SFTP batch transactions:

1. Generate an RSA public/private key pair.

   For example, to generate a 4096-bit RSA key pair, run:

   ```
   ssh-keygen -t rsa -b 4096
   ```

2. (For production environments only) Copy the RSA public key to the Paymentech SFTP server:

   ```
   ssh-copy-id -i pathPublicKey  userName@hostNamePaymentech
   ```

   If prompted, enter the **SFTP** user password.

   where:

   - *pathPublicKey*: Path and file name of the public key created in step 1.
   - *userName*: SFTP user name on the Paymentech SFTP server.
   - *hostNamePaymentech*: Host name of the Paymentech SFTP server.

3. (For testing with the Paymentech Simulator only) In your **override-values.yaml** file for **oc-cn-helm-chart**, add the RSA public key in Base64-encoded format to **ocbrm.dm_fusa.secret.sftpAuthorizedKeys**.

   After updating your configuration, run the **helm upgrade** command for **oc-cn-helm-chart**.

Configuring dm_fusa with SFTP Settings

To configure **dm_fusa** to use SFTP for sending batch payment transactions to Paymentech:

1. In your **override_values.yaml** file for **oc-cn-helm-chart**, set the following keys:

   - **ocbrm.dm_fusa.isEnabled**: Set this to **true**.
   - **ocbrm.dm_fusa.batchProto**: Set this to **sftp**.

2. In the SSH Daemon (sshd) client configuration file (**configmap_dm_fusa_sftp_ssh_config.yaml**), configure server connection details, including server, port, algorithms, and ciphers.

   Below shows the file's default values:

   ```
   data:
      config: |
         Host CServer
         HostName fusa-simulator
         User omsuser
         Port 3333
         IdentityFile /home/omsuser/.ssh/.sftp_key
         #KexAlgorithms ecdh-sha2-nistp521,ecdh-sha2-nistp384,ecdh-sha2-nistp256,diffie-
   hellman-group14-sha256
         #KexAlgorithms diffie-hellman-group14-sha256
         #Ciphers aes128-ctr,aes192-ctr,aes256-ctr
         #HostKeyAlgorithms ecdsa-sha2-nistp521,ssh-rsa,rsa-sha2-512
         #MACs hmac-sha2-512,hmac-sha2-256
         #LogLevel DEBUG3
         #LogLevel VERBOSE
   ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

3. In your **dm_fusa** ConfigMap (**configmap_pin_conf_dm_fusa.yaml**), set the following keys:

```
- dm_fusa sftp_pkey_pwd passPhrase
- dm_fusa sftp_host hostName
- dm_fusa sftp_indir inputPath
- dm_fusa sftp_outdir outPath
- dm_fusa sftp_rfrfile rfrFileName
- dm_fusa sftp_retrys retryValue
- dm_fusa sftp_retry_interval intervalValue
```

Table 20-1 describes the variables in the **dm_fusa** ConfigMap.

Table 20-1 Variables Descriptions

Parameter	Description
passPhrase	The SFTP private key passphrase. If omitted, the passphrase is retrieved from the wallet. By default, this key is commented out.
hostName	The host section name in the SSH client configuration file. The default is **CServer**.
inputPath	(For testing with the Paymentech Simulator only) The absolute path to the input directory where batch files are uploaded using SFTP. The default is **/oms/sftp_in**.
outputPath	(For testing with the Paymentech Simulator only) The absolute path to the output directory where response files are downloaded using SFTP. The default is **/oms/sftp_out**.
rfrFileName	The name of the RFR file to retrieve from the output directory. If this parameter is not specified, **dm_fusa** downloads any unprocessed file found in remote output directory.
retryValue	The number of times to retry connections to Paymentech. The default is **4**.
intervalValue	The interval, in seconds, before retrying the connection to Paymentech. The default is **4**.

4. In your **override-values.yaml** file, set the following keys:
   - **ocbrm.dm_fusa.secret.sftpKey**: Set this to the private key in Base64-encoded format.
   - **ocbrm.dm_fusa.secret.sftpKeyPassPhrase**: Set this to the passphrase in Base64-encoded format.
5. Run the **helm upgrade** command to update the release:

```
helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile --namespace
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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Using TCP/IP for Batch Payment Transactions

To configure **dm_fusa** to use TCP/IP for batch payment transactions:

1. In your **override_values.yaml** file for **oc-cn-helm-chart**, set the following keys:
   - **ocbrm.dm_fusa.isEnabled**: Set this to **true**.
   - **ocbrm.dm_fusa.batchProto**: Set this to **socket**.
2. In the **dm_fusa** environment ConfigMap (**configmap_env_dm_fusa.yaml**), set the following keys:
   - **DMF_BATCH_PORT**: Set this to the Paymentech TCP/IP port number. The default is **8780**.
   - **DMF_BATCH_SRVR**: Set this to the name of the Paymentech TCP/IP server. The default is **fusa-simulator**.
3. Run the **helm upgrade** command to update the release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile --namespace 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.

Configuring TCP/IP for Online Payment Transactions

You configure **dm_fusa** to send online payment transactions to Paymentech using TCP/IP. Online transactions include single authorizations and verifications.

To configure TCP/IP for online payment transactions:

1. In the **configmap_env_dm_fusa.yaml** file, set the following keys:
   - **data.DMF_ONLINE_SRVR**: Set this to the IP address or host name of the Paymentech server for online payment transactions.
   - **data.DMF_ONLINE_PORT**: Set this to the port where Paymentech receives online payment transactions.
2. Run the **helm upgrade** command to update the release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile --namespace 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Configuring the Paymentech Simulator for Testing

You can use Paymentech simulators to test credit card and debit card transactions:

- Use **answer_s** to simulate online transactions.
- Use **answer_b** to simulate batch transactions.

For more information, see "Testing Paymentech Credit Card Processing" in *BRM Configuring and Collecting Payments*.

To configure the Paymentech simulator in your cloud native environment:

1. In the **configmap_pin_conf_fusa_simulator.yaml** file, set the following keys:

   ```
   - answer_b batch_proto sftp
   - answer_b sftp_indir inputPath
   - answer_b sftp_outdir outputPath
   ```

   where:

   - *inputPath* is the absolute path to the input directory where batch files are uploaded using SFTP.
   - *outputPath* is the absolute path to the output directory where response files are downloaded using SFTP.

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

   - **ocbrm.dm_fusa.secret.sftpKey**: The private key in Base64-encoded format.
   - **ocbrm.dm_fusa.secret.sftpKeyPassPhrase**: The Base64-encoded passphrase for the private key.
   - **ocbrm.dm_fusa.secret.sftpAuthorizedKeys**: The Base64-encoded public key accepted by the server for log in.

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

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


Copyright © 2021, 2026, Oracle and/or its affiliates.

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:

- Administering PDC Cloud Native Services
- Running PDC Applications
- Monitoring PDC in a Cloud Native Environment
- Monitoring PDC REST Services Manager
- Rotating PDC Log Files
- Managing Language Packs in PDC Pods

Copyright © 2021, 2026, Oracle and/or its affiliates.

21

Administering PDC Cloud Native Services

Learn how to perform common system administration tasks on your Oracle Communications Pricing Design Center (PDC) cloud native services.

Topics in this document:

- Creating PDC Users
- Using Resource Limits in PDC Domain Pods

Creating PDC Users

PDC is a role-based application that authenticates and authorizes users based on the group to which they belong. The role-based functionality of PDC is supported by the following WebLogic Server groups, which are created when you deploy **oc-cn-op-job-helm-chart**.

- **PricingDesignAdmin**: This group's users have administrative privileges on PDC. They can perform operations on all PDC UI screens, pricing components, and setup components.
- **PricingAnalyst**: This group's users have administrative privileges for pricing components and view-only privileges for setup components.
- **PricingReviewer**: This group's users have view-only privileges for all pricing and setup components.

When you create PDC users, add them to one of these groups based on the role of the user in the organization.

To create PDC users in the PDC domain:

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

```
ocpdc:
   wop:
      users:
         name: UserName
         description: Description
         password: EncodedPassword
         groups: Group
```

   where:

   - *UserName* is the name of the user.
   - *Description* is a brief description of the user.
   - *EncodedPassword* is the Base64-encoded password for the user.
   - *Group* is the name of the group that the user belongs to. The available PDC groups are **PricingAnalyst**, **PricingDesignAdmin**, and **PricingReviewer**.

2. Deploy or redeploy PDC by running the **helm install** command for **oc-cn-op-job-helm-chart**:

Copyright © 2021, 2026, Oracle and/or its affiliates.

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

where:

- *OpJobReleaseName* is the release name assigned to your existing **oc-cn-op-job-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.

Using Resource Limits in PDC Domain Pods

You can optimize the PDC system's CPU and memory usage for requests and limits during runtime. To do so:

1. In your **oc-cn-helm-chart/pdc/vpa_values.yaml** file, set the keys listed in Table 21-1.

Table 21-1 Keys for the BRM Helm Chart vpa_values.yaml File

Pods	Keys
PDC Domain	To set the request and limit values for the PDC domain pod: • **pdcDomainPOD.requests.cpu**: Set this to the default CPU request value. • **pdcDomainPOD.requests.memory**: Set this to the default memory requests value. • **pdcDomainPOD.limits.cpu**: Set this to the maximum number of CPU cores the pod can utilize. • **pdcDomainPOD.limits.memory**: Set this to the maximum amount of memory a pod can utilize. The default is 26i.
Real-Time Rating, Batch Rating, and the SyncPDC utility	To set the request and limit values for the PDC domain pod: • **pdcDomainPOD.requests.cpu**: Set this to the default CPU request value. • **pdcDomainPOD.requests.memory**: Set this to the default memory requests value. • **pdcDomainPOD.limits.cpu**: Set this to the maximum number of CPU cores the pod can utilize. • **pdcDomainPOD.limits.memory**: Set this to the maximum amount of memory a pod can utilize.
The ImportExportPricing utility	To set the request and limit values for the ImportExportPricing pod: • **pdcDomainPOD.requests.cpu**: Set this to the default CPU request value. • **pdcDomainPOD.requests.memory**: Set this to the default memory requests value.. • **pdcDomainPOD.limits.cpu**: Set this to the maximum number of CPU cores the pod can utilize. • **pdcDomainPOD.limits.memory**: Set this to the maximum amount of memory a pod can utilize.

2. In your **oc-cn-op-job-helm-chart/pdc/vpa_values.yaml** file, set the request and limit values for the PDC domain job:
   - **pdcDomainJOB.requests.cpu**: Set this to the default CPU request value.
   - **pdcDomainJOB.requests.memory**: Set this to the default memory requests value.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **pdcDomainJOB.limits.cpu**: Set this to the maximum number of CPU cores the pod can utilize.
- **pdcDomainJOB.limits.memory**: Set this to the maximum amount of memory a pod can utilize.

3. Run the helm upgrade command to update the release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile --
   namespace 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

22

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
- Importing Pricing and Setup Components with ImportExportPricing
- Exporting Pricing and Setup Components with ImportExportPricing
- Using SyncPDC to Synchronize Setup Components

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 pricing and setup components by defining them in one or more XML files and importing them into the PDC database with the **ImportExportPricing** utility.

By default, the **ImportExportPricing** utility generates an XML status report for when the pricing and setup components are being imported, and stores it with the **ImportExportPricing** logs. This report includes the chageset status and the status report of each transformation engine. For more information, see " Checking the Status of a Changeset with ImportExportPricing ".

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:

Copyright © 2021, 2026, Oracle and/or its affiliates.

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.pdcBRMHostPath**
   - **pdc-brm-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 --namespace 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:
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
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 objects 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.pdcBRMHostPath**
   - **pdc-brm-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 appropriate read and write permissions, and set ownership of the input directory, its subdirectories, your import XML files, and **import_order.cfg** file to **chown** *runAsUser*:**0**.
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**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **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 --namespace 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 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **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*.
>
> By default, the **ImportExportPricing** utility generates an XML status report for when the pricing and setup components are being exported, and stores it with the **ImportExportPricing** logs. This report includes the chageset status and the status report of each transformation engine. For more information, see " Checking the Status of a Changeset with ImportExportPricing ".

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-created 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 --
   namespace 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- *BrmNameSpace* is the namespace 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.pdcBRMHostPath** 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
```

Checking the Status of a Changeset with ImportExportPricing

Use the **ImportExportPricing** utility to check the status of a changeset after you import pricing or setup data. A successful import into PDC does not necessarily mean that PDC successfully published the data to the transformation engines.

The utility reports the overall changeset status and the status of each transformation engine.

A changeset has one of the following statuses:

- SUCCEEDED: PDC processed the changeset successfully.
- FAILED: PDC could not process the changeset successfully. Check the XML output and logs for the failed engine status.
- PENDING: PDC has not finished processing the changeset. One or more transformation engines may still be processing the request.

An engine has one of the following statuses:

- COMPLETED: The engine processed the request successfully.
- FAILED: The engine could not process the request successfully.
- DISPATCHED: PDC sent the request to the engine, but the engine has not finished processing it.

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **Note**
>
> A changeset status of SUCCEEDED or FAILED is final. An engine status of COMPLETED or FAILED is final.

To check the status of a changeset and its transformation engines:

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 **changeSetStatus**.
   - **extraCmdLineArgs**: Set this to one of the following values. The value must be surrounded by quotes:
     - "**-name** *changesetName*"
     - "**-n** *changesetName*"
     - "**-id** *idNumber*"

   For more information about the utility's commands, see "changeSetStatus Mode" in *PDC Creating Product Offerings*.
4. Save and close the file.
5. Run the **helm upgrade** command to update the release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart --values OverrideValuesFile --namespace 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 and returns the status in XML format. If the changeset status is PENDING or an engine status is DISPATCHED, check the status again after the engine finishes processing the request.

Example: Checking the Status of a Changeset

This example checks a changeset named **PricingUpdate**:

```
ocpdc:
   configEnv:
      importExport:
         IE_Operation: changeSetStatus
         extraCmdLineArgs: "-name PricingUpdate"
```

In this case, PDC cloud native runs the following command:

```
./ImportExportPricing -changeSetStatus -name PricingUpdate
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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 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 14:00. The valid values are:
  - **startAt**: The utility runs at the time the job is submitted.
  - **"*HH*:*MM*"**: The utility runs at the specified time, where *HH* is an hour between 0 and 23, and *MM* is the minutes between 0 and 59. For example, enter **"12:00"** to schedule the utility to run at noon.
- **ocpdc.configEnv.syncPDC.SyncPDCInterval**: Specifies the frequency at which to run the **SyncPDC** process, such as daily or every 2 hours. Enter a value in the format **"*N*:*U*"**, where *N* is a valid number and *U* is one of these units: **D** (Daily), **H** (Hourly), or **M** (Minute). For example, enter **"2:D"** to run the utility every other day.

> **Note**
>
> A value of "24:H" is not the same as "1:D" due to daylight saving time (DST).

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. It 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **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 --
   namespace 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

23

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
- Setting Up Monitoring in PDC Cloud Native

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 session 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 all PDC metric data stored in Prometheus on a graphical dashboard.

  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 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 Compatibility Matrix*.
4. Enabling monitoring in your PDC cloud native deployment:

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 --namespace 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.

WebLogic Monitoring Exporter is installed in your cloud native environment.

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

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

6. Configure Grafana to display Pricing Design Center metric data. See "Creating Grafana Dashboards for Pricing Design Center".

Creating Grafana Dashboards for Pricing Design Center

Create a dashboard in Grafana to display your Pricing Design Center metric data. Alternatively, you can use the sample dashboard JSON model included in the **oc-cn-docker-files-16.26.0.*x*.*x*.tgz** package.

> **Note**
>
> For the sample dashboard to work properly, the data source name for the WebLogic Domain must be **Prometheus**.

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

Copyright © 2021, 2026, Oracle and/or its affiliates.

24

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
- About PDC REST Services Manager Tracing
- About PDC REST Services Manager Metrics
- About Monitoring PDC REST Services Manager System Health

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 shutdown 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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-
   ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
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: "16.26.0.x.x"
       deployment:
           deadlineSeconds: 60
           revisionHistLimit: 10
           imageName:
             pdcrsm: "oracle/pdcrsm"
           # For non-empty tag, ":" MUST be prepended
           imageTag: ":16.26.0.x.x"
           imagePullPolicy: IfNotPresent
           rootLoglevel: INFO
           appLogLevel: INFO
   ```

2. Update your Helm release. See "Updating a Helm Release".

Copyright © 2021, 2026, Oracle and/or its affiliates.

Changing the Log Levels Using Kubernetes

Change the log levels using Kubernetes for short-term troubleshooting.

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: "16.26.0.x.x"
       deployment:
           deadlineSeconds: 60
           revisionHistLimit: 10
           imageName:
             pdcrsm: "oracle/pdcrsm"
           # For non-empty tag, ":" MUST be prepended
           imageTag: ":16.26.0.x.x"
           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**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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: "16.26.0.x.x"
    deployment:
        deadlineSeconds: 60
        revisionHistLimit: 10
        imageName:
          pdcrsm: "oracle/pdcrsm"
        # For non-empty tag, ":" MUST be prepended
        imageTag: ":16.26.0.x.x"
        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 using Zipkin, 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 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 --
   namespace BrmNameSpace
   ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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.

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. The example below shows 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",
  "timeOccurred": "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/
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
testInit4Offer",
          "name": "100Minutes",
          "version": "1.0",
          "@referredType": "ProductOfferingOracle"
        }
      ]
    }
  }
}
```

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 24-1 shows the available PDC REST Services Manager metrics.

Table 24-1 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.
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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 24-1 (Cont.) PDC REST Services Manager Metrics

Integration Point	Metric	Description
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 "Metrics" in the Helidon documentation for more information: https://helidon.io/docs/v4/mp/guides/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
     ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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": [
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
            {
                "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
                }
            }
        ]
    }
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

25

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

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 log files' maximum file size in bytes. 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 25-1 lists the PDC application's default log file rotation settings.

Table 25-1 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
RRE/BRE Transformation Prime Log	WARNING	50000	50	N/A

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 25-1 (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 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 prime log and transaction logs:

```
ocpdc:
   configEnv:
      transformation:
         logLevel: INFO
         logFileSize: 50000
         logFileCount: 100
         persistTransactionLogs: failed
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

26

Managing Language Packs in PDC Pods

Learn how to change the language displayed in your Oracle Communication Pricing Design Center (PDC) UI screens, XML import files, and XML export files.

Topics in this document:

- Enabling Language Packs in PDC Pods

Enabling Language Packs in PDC Pods

To enable language packs in PDC pods:

1. Create a Dockerfile for building an Oracle Linux Fusion Middleware image.

   For example, to create a Dockerfile for image **container-registry.oracle.com/middleware/fmw-infrastructure_cpu:14.1.2.0-jdk21-ol9**:

   ```
   FROM container-registry.oracle.com/middleware/fmw-infrastructure_cpu:14.1.2.0-jdk21-ol9
   USER root
   RUN locale -a
   RUN microdnf install glibc-all-langpacks && \
       microdnf clean all && \
       rm -rf /var/cache/dnf/* /var/cache/yum/*
   RUN locale -a
   ```

   > **Note**
   >
   > The **RUN locale -a** line is optional.

2. Build the image by running this command:

   ```
   podman build --format=docker --force-rm=true --no-cache=true --tag imageName:imageVersion .
   ```

   For example:

   ```
   podman build --format=docker --force-rm=true --no-cache=true --tag fmw_with_lang:1 .
   ```

   Wait for the image to build.

3. In your **override-values.yaml** file for both **oc-cn-op-job-helm-chart** and **oc-cn-helm-chart**, set the following keys:

   - **ocpdc.deployment.fmw.imageRepository**: Set this to the image repository where the fmw_with_lang:1 image resides.
   - **ocpdc.deployment.fmw.imageName**: Set this to the name of the image you built. For the above example, you would set it to **fmw_with_lang**.
   - **ocpdc.deployment.fmw.imageTag**: Set this to the tag for the image you built. For the above example, you would set it to **1**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **ocpdc.lang**: Set this to the language to use, such as LV_LV.UTF-8 for Latvian or en_US.UTF-8 for American English.

4. Deploy or redeploy PDC cloud native in your environment:

   a. Direct WebLogic Kubernetes Operator to monitor the BRM namespace:

   ```
   helm upgrade weblogic-operator  weblogic-operator/weblogic-operator \
   --namespace Operator \
   --reuse-values \
   --set "domainNamespaces={BrmNameSpace}" \
   --wait
   ```

   where:

   - *Operator* is the namespace you created for WebLogic Kubernetes Operator as part of the prerequisite tasks.
   - *BrmNameSpace* is the namespace in which to create BRM Kubernetes objects for the BRM Helm chart.

   b. Create WebLogic domains by running this command from the **helmcharts** directory:

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

   c. Install PDC cloud native services by entering this command from the **helmcharts** directory:

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


Copyright © 2021, 2026, Oracle and/or its affiliates.

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:

- Administering ECE Cloud Native Services
- Securing ECE Communications
- Managing ECE Journal Storage
- Managing Persisted Data in the Oracle Database
- Configuring Disaster Recovery in ECE Cloud Native
- Managing ECE Pods
- Monitoring ECE in a Cloud Native Environment

Copyright © 2021, 2026, Oracle and/or its affiliates.

27

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
- Changing the ECE Configuration During Runtime
- Rotating Kafka and JMS Credentials at Runtime
- Configuring Subscriber-Based Tracing for ECE Services
- Using Third-Party Libraries and Custom Mediation Specifications
- Loading RADIUS Network Mapping Files in ECE Cloud Native
- Setting Up ECE Cloud Native in Firewall-Enabled Environments
- Enabling Federation in ECE
- Enabling Parallel Pod Management in ECE
- Customizing SDK Source Code

Running SDK Jobs

You can run sample scripts for ECE cloud native services by running an SDK job.

> **Note**
>
> The SDK job runs from the ECE SDK directory in the SDK job pod. The SDK sample scripts, configuration files, source code, and related files are included in the ECE image. If you plan to build custom extension code, include the code in the ECE image by using the supported image-layering customization method before you run the 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. 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>
   ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

The SDK job then goes into a **Completed** state.

3. Check the logs printed by the job by running this command:

   kubectl logs *sdkJobName*

4. 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 namespace, 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*

5. 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*.

Copyright © 2021, 2026, Oracle and/or its affiliates.

6. To view the logs created by the SDK script, run the **kubectl** logs command for the SDK job pod.

Error Handling for SDK Jobs

If an error occurs while an SDK job runs, the job enters an error state. For example, an SDK job can fail when the SDK command contains invalid parameter values.

To troubleshoot an SDK job:

1. Check the SDK job pod logs:

   kubectl logs *sdkJobName*

   where *sdkJobName* is the name of the SDK job pod.

2. If you need to inspect files that the SDK job creates, inspect them in the SDK job pod. The ECE SDK is available under**/home/charging/opt/ECE/ocecesdk**.

   > **Note**
   >
   > Files that the SDK job generates in the pod are not persisted when Kubernetes deletes or re-creates the SDK job pod. If you need to retain generated files for troubleshooting or later use, copy them to configured persistent storage before the pod is removed.

3. If the log output contains a message similar to the following, verify that your custom extension code and build scripts are included in the customized ECE image at the locations expected by the SDK job. Also verify that **job.sdk.command** specifies the correct build script or directory.

   ```
   cp: cannot stat '/home/charging/config/ece.properties': No such file or
   directory
   ```

   See "Customizing SDK Source Code" for information about providing custom SDK source code through the ECE image.

4. After you correct the error, run the **helm upgrade** command with a new SDK job name. See "Running SDK Jobs".

If you don't provide SDK commands while running the **helm upgrade** command, the SDK job prints:

```
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, ECE uses the default job name of **sdk**. Kubernetes does not allow a completed job to run again with the same job resource. Delete the previous **sdk** job or specify a new job name before you run the SDK job again.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Changing the ECE Configuration During Runtime

After you deploy your ECE cloud native services, changes to the ECE application configuration require a rolling update of the ECE pods unless you use one of the runtime methods described in this topic. These runtime methods do not require restarting the ECE pods.

To update the ECE configuration at runtime, use one of the following methods:

- Modify ECE configuration MBeans through a JMX editor. See "Creating a JMX Connection to ECE Using JConsole".
- Reload the ECE application configuration by running a Kubernetes job. See "Reloading ECE Application Configuration Changes".
- Reload the grid log level for an ECE component by running a Kubernetes job. See "Reloading the Grid Log Level".

For information about how ECE handles Kafka and JMS credential-only rotations at runtime, see "Rotating Kafka and JMS Credentials at Runtime".

> **Note**
>
> You can run a Kubernetes job to reload either the ECE application configuration or the grid log level, but not both at the same time.

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 ecs-0 ece-jmx=ece-jmx-external
   ```

3. Retrieve the worker node's IP address by running this command:

   ```
   kubectl get pod ecs-0 -o wide
   ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

4. Update the **/etc/hosts** file in the remote machine with the worker node's IP by running this command:

   ```
   ipAddress ecs-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 ecs-0.ece-server.namespace.svc.cluster.local:jmxport
   ```

Afterward, you can start using JConsole to change ECE configuration MBeans. See "Managing Online Charging Sessions" in *ECE Implementing Charging*.

Reloading ECE Application Configuration Changes

You can change the ECE application configuration during runtime by running a Kubernetes job. The job automatically reloads the application's configuration into the ECE cloud native cache and the **charging-settings.xml** file.

To reload ECE application configuration changes:

1. Open your **override-values.yaml** file for the ECE Helm chart.
2. Modify the ECE configuration MBeans to meet your business needs.

   For example, changing the **charging.server.degradedModeThreshold** to **3**.
3. Set the **job.chargingConfigurationReloader.reloadAppConfig.runjob** key to **true**.

   This specifies to run a Kubernetes job.
4. Optionally, set the **job.chargingConfigurationReloader.reloadAppConfig.command** key to the location of the configuration MBean. For example, enter **charging.server** for the **degradedModeThreshold** Mbean, and enter **charging.notification** for the **rarNotificationMode** MBean.
5. Do not change the pod's specification-related keys that can trigger a restart of the pod during a Helm upgrade. For example, do not change the **restartCount**, **image**, or **jvmGCOpts** keys.
6. Run the **helm upgrade** command to update your Helm release:

   ```
   helm upgrade EceReleaseName oc-cn-ece-helm-chart --values
   OverrideValuesFile --namespace EceNameSpace
   ```

   The upgrade updates the **charging-settings.xml** file in the cache, updates the ECE **charging-settings-namespace** ConfigMap, and triggers the **charging-configuration-reloader** job.
7. Validate that the MBean attribute was modified by running the **query.sh** script in the ecs pod.

   See "Using the query Utility to Test ECE" in *ECE Implementing Charging* for more information.

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **Note**
>
> You do not need to restart the ecs, gateway, or ratedeventformatter pods for most ECE configuration changes. Restarts are required only for changes to database connection URL, Rated Event Formatter, Gateway-related, and Kafka-related application configuration parameters.

Reloading the Grid Log Level

You can change the grid log level for any ECE component at runtime by using a Kubernetes job.

To reload the grid log level during runtime:

1. Open your **override-values.yaml** file for the ECE Helm chart.
2. Set the **job.chargingConfigurationReloader.reloadLogging.runjob** key to **true**.
3. Set the **job.chargingConfigurationReloader.reloadLogging.command** key to the following:

   ```
   loggerOperation oracle.communication.brm.charging.loggerName loggerLevel
   ```

   where:

   - *loggerOperation*: The type of log operation, which can be **setGridLogLevel**, **setLogLevel**, **setGridLogLevelForFunctionalDomain**, **setLogLevelForFunctionalDomain**, or **updateSubscriberTraceConfiguration**.
   - *loggerName*: The name of the component logger or functional name.
   - *loggerLevel*: Specifies the log level, which can be **ALL**, **DEBUG**, **ERROR**, **INFO**, **TRACE**, or **WARN**.

   For example, to set the grid log level for the ECE application configuration to error:

   ```
   setGridLogLevel oracle.communication.brm.charging.appconfiguration ERROR
   ```

4. To persist the log level changes in the database, set the **log4j2.logger.***loggerName* key to the log level. The *loggerName* and *loggerLevel* must match the values from step 3.

   For example, if the **command** key is set to **setGridLogLevel oracle.communication.brm.charging.brmgateway INFO**, you must set the key as follows:

   ```
   log4j2.logger.brmgateway: INFO
   ```

5. Do not change the pod's specification-related keys that can trigger a restart of the pod during a Helm upgrade. For example, do not change the **restartCount**, **image**, or **jvmGCOpts** keys.
6. Run the **helm upgrade** command to update your Helm release:

   ```
   helm upgrade EceReleaseName oc-cn-ece-helm-chart --values
   OverrideValuesFile --namespace EceNameSpace
   ```

After the job completes, the logging level is reflected in the ECE grid pods.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Rotating Kafka and JMS Credentials at Runtime

ECE cloud native automatically recovers from stale Kafka and JMS credentials at runtime without requiring an ECE application or pod restart. You do not need to enable credential refresh separately. ECE uses your existing Kafka, JMS, wallet, and connection retry configuration.

Credential rotation does not immediately refresh an existing Kafka producer or JMS connection. After the updated credential is available in the ECE wallet, ECE refreshes the affected connection resources when a Kafka or JMS operation fails because of stale credentials.

Continue to use your existing procedures for secret and wallet credential rotation to propagate updated credentials to the ECE wallet.

JMS Credential Rotation Considerations

When you rotate JMS credentials, keep the WebLogic user password, WebLogic boot identity, and corresponding ECE wallet credential synchronized.

ECE uses the configured JMS connection retry count and retry interval when it attempts to reconnect. To allow recovery within the configured retry window, update the ECE wallet and ensure that the JMS provider is available before the retry window ends.

If the retry window ends before the updated credentials or JMS provider become available, you do not need to restart ECE. A subsequent JMS operation can establish a new connection using the current credentials in the ECE wallet.

Kafka Credential Rotation Considerations

When a Kafka operation fails because the existing producer can no longer use its credentials, ECE refreshes the producer and consumer using the current wallet credentials and retries the failed operation once.

If the retried operation also fails, ECE uses the existing Kafka publish failure handling and logs the failure.

Monitoring Credential Refresh

ECE cloud native records Kafka and JMS connection refresh activity in the logs of the ECE pod that uses the affected connection.

The logs can include the following information:

- A Kafka or JMS operation failure that triggers connection refresh
- Attempts to refresh the affected connection resources using the current wallet credentials
- Kafka stale-producer cleanup
- JMS reconnection attempts and retry activity
- Refresh or reconnection failures

Kafka refreshes the producer and retries the failed operation once. JMS performs reconnection attempts according to the configured JMS connection retry count and retry interval.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Use the following command to monitor an ECE pod for Kafka and JMS credential-refresh messages:

```
kubectl logs -f ecePodName -n BrmNameSpace | \
grep -E "Refreshing Kafka producer|Closed stale Kafka producer|Refreshing connection
using current wallet credentials|already refreshed"
```

where:

- *ecePodName* is the name of the ECE pod that uses the affected Kafka or JMS connection.
- *BrmNameSpace* is the Kubernetes namespace that contains the ECE pod.

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 distribution functions, do the following:

Copyright © 2021, 2026, Oracle and/or its affiliates.

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.i
nternal.ComponentLoggerImpl"/>
   <componentLogger
      loggerName="oracle.communication.brm.charging.rating.alteration"
      loggerLevel="DEBUG"
      config-
class="oracle.communication.brm.charging.subscribertrace.configuration.i
nternal.ComponentLoggerImpl"/>
</componentLoggerList>
```

d. Save and close your **charging-settings.yaml** ConfigMap 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 namespace 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 ecs-0 pod so that JMX can connect to it:

   kubectl -n namespace label pod ecs-0 ece-jmx=ece-jmx-external

6. Update the **/etc/hosts** file on the remote machine with the worker node of ecs-0:

   IP_OF_WORKER_NODE ecs-0.ece-server.namespace.svc.cluster.local

7. Connect to JConsole by entering this command:

   jconsole ecs-0.ece-server.namespace.svc.cluster.local:31022

   JConsole starts.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 appear.

> **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.

Using Third-Party Libraries and Custom Mediation Specifications

To use third-party libraries and custom mediation specifications with ECE cloud native:

1. Add the required files to a customized ECE image:
   - Add third-party JAR files under **/home/charging/ext/3rdparty_jars**.
   - Add custom mediation specifications under **/home/charging/ext/ece_custom_data**.
2. Build and publish the customized ECE image, and configure your ECE deployment to use that image.
3. Run the **helm install** command:

   ```
   helm install EceReleaseName oc-cn-ece-helm-chart --namespace BrmNameSpace
   --values OverrideValuesFile
   ```

   where:
   - *BrmNameSpace* is the namespace 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 after the ECE cluster is set up:

Copyright © 2021, 2026, Oracle and/or its affiliates.

1. Add the custom mediation specifications under **/home/charging/ext/ece_custom_data** when you build your customized ECE image.
2. Build and publish the updated ECE image, and update your ECE Helm configuration to reference that image.
3. Stop the configloader pod.

> **Note**
>
> The configloader pod loads the mediation specifications into the ECE cache.

4. Connect to JConsole. See "Creating a JMX Connection to ECE Using JConsole".
5. In JConsole, click the **MBeans** tab.
6. Expand the **ECE Configuration** node.
7. Expand **migration.loader**.
8. Expand **Attributes**.
9. Set the **configObjectsDataDirectory** attribute to the absolute path in the ECE container where the custom mediation specifications are available.

> **Note**
>
> To load only specific mediation specifications, set the **configObjectsDataDirectory** attribute to the absolute path in the ECE container where the specifications are available.

10. Exit JConsole.
11. 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 9.
12. Run the **helm upgrade** command to update the ECE Helm release:

```
helm upgrade EceReleaseName oc-cn-ece-helm-chart --values
OverrideValuesFile --namespace BrmNameSpace
```

Loading RADIUS Network Mapping Files in ECE Cloud Native

When you use RADIUS Gateway in a BRM cloud native deployment, the network mapping XML files generated by Pricing Updater must be available to the configloader pod before the pod starts.

> **Note**
>
> Starting with BRM cloud native release 16.26.0, network mapping XML files generated in the pricingupdater pod are not automatically available in the configloader pod.

Copyright © 2021, 2026, Oracle and/or its affiliates.

If Config Loader starts before the network mapping XML files are available to it, RADIUS Gateway usage requests can fail. The following procedure makes the files available to Config Loader and populates the cache.

To load the RADIUS network mapping files in ECE cloud native:

1. Stop the configloader pod if it is running.
2. In the Pricing Updater pod, make the **network_mappings** directory available in the pod as a subdirectory of the absolute path specified by the **configObjectsDataDirectory** parameter and ensure that the network mapping files are in that directory.
3. Copy the **network_mappings** directory from the pricingupdater pod into the absolute directory specified by the **configObjectsDataDirectory** parameter in the configloader pod using one of these methods:
   - Add the directory to the configloader container image layer.
   - Create a ConfigMap from the **network_mappings** directory in the pricingupdater pod, and mount the ConfigMap as read-only in the configloader container as the **network_mappings** subdirectory of the absolute path specified by the **configObjectsDataDirectory** parameter. For more information, see "Exposing Directories as ConfigMaps".
4. Start the configloader pod.

   When the configloader pod starts, it reads the network mapping XML files and loads them into its cache. RADIUS Gateway can then use the mappings to process usage requests.
5. Send a RADIUS accounting request and verify that RADIUS Gateway processes the request successfully.

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
      ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Enabling Federation in ECE

Enabling federation in ECE allows you to 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
- 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 clusters.

   - **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**:

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **Note**
>
> Set the **jvmCoherenceOpts** keys in each **charging.***coherenceMemberName* section with Coherence Federation parameters for the primary and secondary clusters.

- **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 ecs-0 pod:

      ```
      kubectl label -n NameSpace po ecs-0 ece-jmx=ece-jmx-external
      ```

   b. Update the **/etc/hosts** file on the remote machine with the worker node of ecs-0:

      ```
      IP_OF_WORKER_NODE ecs-0.ece-server.namespace.svc.cluster.local
      ```

   c. Connect to JConsole:

      ```
      jconsole ecs-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

Copyright © 2021, 2026, Oracle and/or its affiliates.

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.

> **Note**
>
> This procedure initializes federation from the primary production site to the secondary production sites. For an active-active disaster recovery deployment, also complete the secondary-to-primary federation startup described in "Setting Up Active-Active Disaster Recovery for ECE".

Enabling Parallel Pod Management in ECE

You can configure the Kubernetes StatefulSet controller to start all ecs pods simultaneously 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-chart**):

```
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`

Copyright © 2021, 2026, Oracle and/or its affiliates.

where *BrmNameSpace* is the namespace 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

ECE includes the SDK sample scripts, configuration files, source code, and related files in the ECE image. When an SDK job starts, ECE makes the SDK content available under **/home/charging/opt/ECE/ocecesdk** in the SDK job pod.

ECE does not use a dedicated SDK PVC. Changes that you make directly in the SDK job pod are pod-local and do not update the ECE image. To reuse customized SDK source code or scripts across SDK jobs, include the customizations in a customized ECE image before you run the SDK job.

To customize the ECE SDK source code:

1. Prepare the custom source files or scripts that you want to add to the SDK.
2. Add the custom source files or scripts to the SDK content in your customized ECE image so that they are available under **/home/charging/opt/ECE/ocecesdk** when the SDK job starts.
3. If you package custom extension code as a JAR file, add the JAR file to **/home/charging/ext/3rdparty_jars** when you build the customized ECE image.

   ECE adds JAR files from **/home/charging/ext/3rdparty_jars** to the ECE runtime libraries. Add the required JAR files to the image before you run the SDK job.
4. Build the customized ECE image and push it to a container image repository that your Kubernetes cluster can access.
5. Configure your ECE deployment to use the customized image.
6. In your **override-values.yaml** file, set the **job.sdk.command** key to the command that builds your custom extension code.

   To use the standard SDK build command, set the key to **launcher.sh** build.

   If you require additional build logic, include a custom script in the customized ECE image and set **job.sdk.command** to the name of that script. You can use **sample_gprs_session.sh** as a reference when you create the script.
7. Run the SDK job by following the instructions in "Running SDK Jobs".

When you run the SDK job with the build and run options, the job builds and runs the customized code from the SDK directory in the pod.

Copyright © 2021, 2026, Oracle and/or its affiliates.

28

Securing ECE Communications

Learn how to secure communications within your Oracle Communications Elastic Charging Engine (ECE) cloud native system and between ECE and external applications.

Topics in this document:

- Enabling SSL Communication When BRM and ECE Run in Separate Clusters
- Using Customer-Provided TLS Certificates for BRM Communication
- Securing Communication Between the CHF and NRF, PCF, and SMF

Enabling SSL Communication When BRM and ECE Run in Separate Clusters

If BRM and ECE run in different Kubernetes clusters or cloud native environments, configure 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.

   You can use **nodeSelector** to schedule the EM Gateway pod on specific worker nodes.

3. Provide the BRM client and server wallets that ECE requires for secure communication with BRM.

   You can provide each wallet in either of the following ways:

   - Add the complete client wallet to **oc-cn-ece-helm-chart/secrets/brmwallet/client/**.

     Add the complete server wallet to **oc-cn-ece-helm-chart/secrets/brmwallet/server/**.

   - Create Kubernetes secrets that contain the complete client and server wallets. In **override-values.yaml**, set **charging.extBRMClientWalletSecret** and **charging.extBRMServerWalletSecret** to the names of the corresponding secrets.

   If you use external Kubernetes secrets, create them in the namespace in which you install the ECE Helm chart. Each secret must contain all of the files required by its wallet.

4. Configure SSL and the BRM wallet settings in **override-values.yaml**. For example:

   ```
   emgateway:
     emGatewayList:
       - coherenceMemberName: "emgateway1"
         emGatewayConfiguration:
           sslEnabled: "1"
   ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
charging:
  extBRMClientWalletSecret: "client-wallet-secret"
  extBRMServerWalletSecret: "server-wallet-secret"
  brmWalletServerLocation: "cwallet.sso"
  connectionConfigurations:
    BRMConnectionConfiguration:
      sslEnabled: "1"
      brmwallet: "cwallet.sso"
```

If you provide wallet files through the Helm chart instead of external Kubernetes secrets, omit the corresponding **extBRMClientWalletSecret** or **extBRMServerWalletSecret** setting.

Specify only the wallet file name, such as **cwallet.sso**. Do not specify an absolute or relative wallet path. The ECE Helm chart constructs the complete runtime path and mounts the selected wallets in the appropriate pods.

When you enable SSL for **BRMConnectionConfiguration**, the ECE Helm chart mounts the client wallet read-only in BRM Gateway. When you enable SSL for at least one EM Gateway, the chart mounts the server wallet read-only in the EM Gateway pods. ECE does not create or generate BRM client or server wallets at runtime.

> **Note**
>
> RADIUS Gateway retrieves the BRM root-key wallet from the BRM database. You do not provide or configure a BRM root-key wallet for RADIUS Gateway in the ECE Helm deployment.

5. Provide the ECE wallet password and BRM Gateway password by using the supported secret configuration. In **override-values.yaml**, configure the required **secretEnv** settings, including:
   - **secretEnv.walletPassword**: Specifies the password for the ECE wallet.
   - **secretEnv.BRMGATEWAYPASSWORD**: Specifies the BRM Gateway password that ECE uses to connect to CM.

   If you use a precreated external Kubernetes secret for the ECE secret configuration, set **secretEnv.extECESecret** to the name of the secret. Ensure that the secret contains the values required by the ECE deployment.

   > **Note**
   >
   > The ECE secret configured by **secretEnv.extECESecret** is separate from the BRM client and server wallet secrets configured by **charging.extBRMClientWalletSecret** and **charging.extBRMServerWalletSecret**.

6. If you are deploying ECE for the first time, install the ECE Helm chart:

   ```
   helm install ECEReleaseName oc-cn-ece-helm-chart --namespace BRMNamespace --values
   overrideValuesFile
   ```

7. If ECE is already deployed, update the applicable wallet secrets or wallet files and run a Helm upgrade. Restart the affected ECE pods as required so that they use the updated wallet and secret data. See "Rolling Restart of ECE Pods" for more information.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Using Customer-Provided TLS Certificates for BRM Communication

You can use customer-provided TLS certificates to secure communication between ECE and BRM. ECE does not include or generate BRM client or server wallets. You provide the complete wallets that ECE uses for TLS communication.

You can provide each BRM wallet by adding the wallet files to the ECE Helm chart or by using an existing Kubernetes secret.

To configure customer-provided TLS certificates for BRM communication:

1. Create the BRM client and server wallets that your deployment requires.

   The client wallet contains the certificates that BRM Gateway uses when it connects to BRM. The server wallet contains the certificates that EM Gateway uses for its TLS listener.

2. Provide the wallets by using one of the following methods:

   - To provide wallet files through the ECE Helm chart, add the complete wallets to these directories:

     oc-cn-ece-helm-chart/secrets/brmwallet/client/
     oc-cn-ece-helm-chart/secrets/brmwallet/server/

   - To use existing Kubernetes secrets, create separate secrets for the client and server wallets. Each secret must contain all files required by its wallet and must exist in the namespace in which you install the ECE Helm chart.

3. If you use existing Kubernetes secrets, specify their names in the **override-values.yaml** file for **oc-cn-ece-helm-chart**:

   ```
   charging:
     extBRMClientWalletSecret: "client-wallet-secret"
     extBRMServerWalletSecret: "server-wallet-secret"
   ```

   If you provide a wallet through the Helm chart instead, omit the corresponding external secret setting.

4. Configure the wallet file names and enable TLS for the ECE components that use the wallets. The following example uses existing Kubernetes secrets for both wallets:

   ```
   emgateway:
     emGatewayList:
       - coherenceMemberName: "emgateway1"
         emGatewayConfiguration:
           sslEnabled: "1"

   charging:
     extBRMClientWalletSecret: "client-wallet-secret"
     extBRMServerWalletSecret: "server-wallet-secret"
     brmWalletServerLocation: "cwallet.sso"
     connectionConfigurations:
       BRMConnectionConfiguration:
         sslEnabled: "1"
         brmwallet: "cwallet.sso"
   ```

5. Install or upgrade the ECE Helm chart.

   When you enable TLS for **BRMConnectionConfiguration**, the Helm chart mounts the BRM client wallet read-only in BRM Gateway. When you enable TLS for at least one EM

Copyright © 2021, 2026, Oracle and/or its affiliates.

Gateway, the chart mounts the BRM server wallet read-only in the EM Gateway pods. ECE does not generate either wallet at runtime.

> **Note**
>
> RADIUS Gateway retrieves the BRM root-key wallet from the BRM database. You do not provide a root-key wallet or configure a root-wallet path for RADIUS Gateway in the ECE Helm deployment.

Securing Communication Between the CHF and NRF, PCF, and SMF

You can enable secure communication between the HTTP Gateway (CHF) and the NRF, PCF, and SMF in one of these ways:

- Using KeyStore certificates stored in the cloud native Helm charts. See "Securing Communication Using KeyStores Mounted in the Helm Chart".
- Using external Kubernetes Secrets. See "Securing Communication Using External Kubernetes Secrets".

Securing Communication Using KeyStores Mounted in the Helm Chart

To enable secure communication between the HTTP Gateway (CHF) and the NRF, PCF, and SMF using KeyStores mounted in the Helm chart:

1. Generate your SSL TrustStore and Identity KeyStore certificates for the NRF, PCF, and SMF.
2. Move the SSL TrustStore and Identity KeyStore certificate files to the **oc-cn-ece-helm-chart/secrets/httpgateway** directory.

   When you perform a Helm install or upgrade, the ECE Helm chart mounts the TrustStore and Identity KeyStores inside the httpgateway pods as a Kubernetes Secret.
3. Open your **override-values.yaml** file for **oc-cn-ece-helm-chart**.
4. Enable SSL-based communication between the CHF and the PCF, NRF, and SMF by setting the following keys under **httpgateway.httpgatewayList.httpGatewayConfiguration**:
   - **pcfSSLEnabled**: Set this to **true** to enable SSL-based communication between the CHF and PCF.
   - **nrfSSLEnabled**: Set this to **true** to enable SSL-based communication between the CHF and NRF.
   - **smfSSLEnabled**: Set this to **true** to enable SSL-based communication between the CHF and SMF.
   - **httpSSLType**: Specify the type of SSL communication: **oneway** or **twoway**.
5. Specify the name and location of the TrustStore and Identity KeyStore files by setting the following keys under **httpgateway**:
   - **httpIdentityKeystore**: Specify the path to the Identity KeyStore certificate files.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **httpIdentityKeystoreType**: Specify the type of SSL Identity KeyStore: **PKCS12** or **SSO**.
- **httpTruststore**: Specify the path to the SSL TrustStore files.
- **httpTruststoreType**: Specify the type of SSL TrustStore file: **PKCS12** or **SSO**.

6. If your ECE cloud native services route communication between the CHF and other network functions through an Oracle Services Communications Proxy (SCP), do the following:
   a. In the **httpgateway.httgatewayList.httpGatewayConfiguration.scpAuthorities** key, enter the URL of the primary and secondary SCP authority, delimited by commas.

      For example: scpAuthorities="scp1.example.com,scp2.example.com".

   b. Ensure that the **oc-cn-ece-helm-chart/secrets/httpgateway/** directory contains the SCP SSL KeyStore certificates rather than PCF and SMF certificates.

   See "Configuring Communication through SCP" in *ECE Implementing Charging* for more information about SCP.
7. Run the **helm install** or **helm upgrade** command to update the ECE Helm chart.

   The KeyStores are mounted inside the httpgateway pod as a Kubernetes Secret at **/home/charging/mnt/secrets/httpgateway/server_ssl**.

Securing Communication Using External Kubernetes Secrets

To enable secure communication between the HTTP Gateway (CHF) and the NRF, PCF, and SMF using external Kubernetes Secrets:

1. Create your KeyStore certificates for the NRF, PCF, and SMF as Secrets in your Kubernetes cluster.

   > **Note**
   >
   > If communication is routed through an Oracle Services Communications Proxy (SCP), create SCP-related certificates instead.

   For information about creating Kubernetes Secrets, see "Managing Secrets" in the Kubernetes documentation.
2. Open your **override-values.yaml** file for **oc-cn-ece-helm-chart**.
3. Specify the external Kubernetes Secrets by setting the following keys under **httpgateway**:
   - **extHttpIdentityKeystoreSecret**: Specify the name of the external Kubernetes Secret containing the HTTP Identity KeyStore.
   - **httpIdentityKeystore**: Specify the name of the Identity KeyStore certificate file contained in the external Kubernetes Secret.
   - **httpTruststore**: Specify the name of the SSL TrustStore file contained in the external Kubernetes Secret.
   - **extHttpTruststoreSecret**: Specify the name of the external Kubernetes Secret containing the HTTP TrustStore.
4. If your ECE cloud native services route communication between the CHF and other network functions through an Oracle Services Communications Proxy (SCP), set the

Copyright © 2021, 2026, Oracle and/or its affiliates.

httpgateway.httgatewayList.httpGatewayConfiguration.scpAuthorities key to the URL of the primary and secondary SCP authority, delimited by commas.

For example: scpAuthorities="scp1.example.com,scp2.example.com".

See "Configuring Communication through SCP" in *ECE Implementing Charging* for more information about SCP.

5. Perform a helm install of the ECE Helm chart:

```
helm install 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. It must be different from the one used for the BRM Helm chart.
- *BrmNameSpace* is the namespace in which BRM Kubernetes objects reside for the BRM Helm chart.
- *OverrideValuesFile* is the path to a YAML file that overrides the default configurations in the **oc-cn-ece-helm-chart/values.yaml** file.

Copyright © 2021, 2026, Oracle and/or its affiliates.

29

Managing ECE Journal Storage

Learn how to manage the size of the journal storage in Oracle Communications Elastic Charging Engine (ECE) cloud native deployments to meet your business needs.

Topics in this document:

- About Coherence Elastic Data Storage
- Managing Coherence Journal Space

About Coherence Elastic Data Storage

ECE cloud native utilizes Oracle Coherence Elastic Data to manage data in both the ECE cache and the federated cache. Coherence Elastic Data stores data across memory and disk-based devices, employing a journaling mechanism to store object state changes. Journals record values for a specific key, and an in-memory map specifies which journal file contains the most recent value for a key.

Coherence Elastic Data contains two types of journals:

- **RAM Journal**: This journal stores data in memory.
- **Flash Journal**: This journal stores data on disk-based devices, such as Solid State Disks (SSDs). The disk can be local or shared and is expected to perform similarly to SSDs.

The flash journal functions as overflow memory for the RAM journal. Coherence writes data to the RAM journal until it reaches a specified maximum, after which it writes data to the flash journal. After the RAM journal's usage drops below the maximum threshold, Coherence returns to writing data to the RAM journal.

Over time, outdated data can accumulate in a journal file. Coherence Elastic Data uses an internal garbage collection algorithm to track whether a threshold is reached. When the threshold is met, garbage collection threads are activated to remove outdated values, and the files are recycled and reused for future data storage.

In ECE cloud native, Coherence Elastic Data uses two per-JVM storage buffers, with one dedicated to each journal. The Elastic Data serves as a single resource shared by all services running on a cluster member.

Managing Coherence Journal Space

You can control the amount of space the flash and RAM journals use to meet your business needs. By default, ECE cloud native creates journal space for small-to-medium-sized deployments with up to 20,000 TPS. For larger deployments, you may need to increase the size of the journal space.

To manage the amount of space Coherence Elastic Data uses to store journal data, do the following:

1. Open your **override-values.yaml** file for **oc-ece-cn-helm-chart**.
2. Configure Coherence Elastic Data for ECE cloud native by modifying these entries under **charging.journalingConfig**:

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **journalManagerDirectoryPath**: Specifies the path for storing the flash journal files. Ensure this directory already exists before deploying.
- **flashJournalMaxSize**: Specifies the maximum size per ECS JVM for a flash journal file. The default is 10 GB.
- **ramJournalMaxSize**: Specifies the maximum size of the RAM journal as a percentage of the Heap memory, such as 10% or 20%. When the maximum percentage is reached, Coherence Elastic Data writes data to the flash journal. The default is 10% and should not be increased without justification.

3. The federation service uses an internal journal record cache to track cache data changes that have not yet been federated to other federation participants. Specify the maximum size of this cache by configuring the following keys under **charging.federatedCacheScheme.journalCacheHighUnits**:

> **Note**
>
> Keep the relative sizing between the HighUnits values. In addition, the sum of all four HighUnits values should not exceed **flashJournalMaxSize** + **ramJournalMaxSize**.

- **brmJournalCacheHighUnits**: Specifies the maximum size of the federation journal record cache for the BRM federated service. The default is 8 GB.
- **xreffederatedJournalCacheHighUnits**: Specifies the maximum size of the federation journal record cache for the XREF federated service. The default is 500 MB.
- **replicatedfederatedJournalCacheHighUnits**: Specifies the maximum size of the federation journal record cache for the replicated federated service. The default is 500 MB.
- **offerProfileFederatedJournalCacheHighUnits**: Specifies the maximum size of the federation journal record cache for the offer profile federated service. The default is 500 MB.

When the limit is reached, the federation service moves the destination participants to the ERROR state and removes all pending entries from the journal record cache.

4. Save and close your **override-values.yaml** file.
5. Redeploy your ECE Helm chart by running this command:

```
helm install EceReleaseName oc-cn-ece-helm-chart --namespace BrmNameSpace --values
override-values.yaml
```

where:

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

Copyright © 2021, 2026, Oracle and/or its affiliates.

30

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
- Loading Only Partial Data into ECE Cache
- Incremental Customer Loading in ECE Cache

Enabling Persistence in ECE

You can configure ECE to persist its cache data in the Oracle database. Persistence provides a durable copy of the cache data that ECE can use to recover from events such as a node failure or the loss of a cache partition. When ECE needs to recover persisted data, it automatically reloads the data from the persistence database.

When you enable persistence, ECE components such as Customer Updater, Pricing Updater, and ConfigLoader persist the following data:

- Data that BRM and PDC publish to the ECE cache
- Mediation specification data that ECE loads into the cache
- Data that ECE synchronizes with or receives from BRM
- Other persisted data, including balances, top-up history, recurring bundle history, rated events, and Portal object IDs (POIDs)

> **Note**
>
> - If **pricingLoadFromPersistence** is set to **false** and the ECE state is not **USAGE_PROCESSING**, Pricing Updater cleans the existing ECE schema at startup and expects PDC to republish the pricing data.
> - Similarly, if **customerLoadFromPersistence** is set to **false** and the ECE state is not **USAGE_PROCESSING**, Customer Updater attempts to clean the ECE schema at startup before it reloads customer data from BRM.
> - When **pricingLoadFromPersistence** is set to **false** for an ECE cold restart, ensure that PDC republishes the pricing data and that the data is available on all ECE nodes before Customer Updater loads customer data. For more information, see "Avoiding Missing Pricing Data During ECE Cold Restarts".

During installation, upgrade, auto-recovery, and pod restart, ECE uses the Kubernetes REST API to:

Copyright © 2021, 2026, Oracle and/or its affiliates.

- Update the **charging-settings-*namespace*** ConfigMap to enable ECE to reload cache data from the persistence database
- Retrieve metadata from ECE StatefulSets and pods
- 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. Configure the **ece-*namespace*** service account to authenticate with the Kubernetes API server.

   For information about the rules defined in the **ece-*namespace*** role-based access control (RBAC) configuration, see the **ece-clusterrole-sa.yaml** file in **oc-cn-ece-helm-chart**.

2. Enable and configure persistence. In the **override-values.yaml** file for **oc-cn-ece-helm-chart**, configure the required persistence credentials and connection settings.

   > **Note**
   >
   > Specify all persistence tablespace names in uppercase.

```
secretEnv:
  PERSISTENCEDATABASEPASSWORD:
    - schema: 1
      PASSWORD: EncryptedPassword

  PERSISTENCEDBAPASSWORD:
    - schema: 1
      PASSWORD: EncryptedPassword    # SYSDBA user

  PERSISTENCEDATABASEKEYPASS:
    - schema: 1
      PASSWORD: EncryptedPassword

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"
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
        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"
```

> **Note**
>
> - When **configLoadFromPersistence** is set to **true**, ConfigLoader loads configuration data from the persistence database. The **configDataDirectory** log message appears before ConfigLoader selects the data source and does not indicate that ECE loaded mediation specification files, such as **diameter_mediation.spec**, from the file system.
> - When ConfigLoader loads mediation specification files from the file system, the logs include file-processing messages such as **processing file** or **ConfigLoader Load Completed From Files**.

When you deploy **oc-cn-ece-helm-chart** with persistence enabled, the Helm chart creates the schema user if it does not already exist, creates the ECE tables and indexes, and runs the required stored procedures.

Rotating Persistence Database Passwords

In an ECE cloud native deployment, you can rotate the Oracle persistence database password without restarting the ECE pods.

When you change a persistence database password, update the corresponding **PERSISTENCEDATABASEPASSWORD** value for the affected schema in the Kubernetes secret. Apply the updated Helm configuration so that Kubernetes makes the new secret value available to the running ECE pod.

Copyright © 2021, 2026, Oracle and/or its affiliates.

The ECE wallet sidecar monitors the mounted secret data. When it detects a change to a mapped persistence database password, it updates the corresponding credential in the pod-local ECE wallet.

ECE refreshes the password after a connection failure when the affected pool next needs to establish or validate a database connection. The refresh does not occur immediately when the wallet sidecar updates the wallet.

If ECE cannot retrieve a valid password from the wallet, it leaves the existing connection-pool password unchanged. If the connection retry fails, ECE reports the connection failure through the existing error path. ECE logs the initial connection failure, the password-refresh status, and the retry result without logging password values or Kubernetes secret contents.

> **Note**
>
> Persistence upgrade jobs do not use the runtime password-refresh mechanism. These jobs run as startup or Helm job flows and use the database credentials in the Kubernetes secret for each schema. Before you run an upgrade that includes persistence jobs, ensure that each schema-specific database password matches the corresponding Kubernetes secret.

To view the ECE persistence deployment logs, run the following command:

```
kubectl logs -f EcePersistenceJobPod -n BrmNameSpace
```

where:

- *EcePersistenceJobPod* is the name of the pod in which **ece-persistence-job** runs.
- *BrmNameSpace* is the namespace that contains the BRM cloud native deployment.

Avoiding Missing Pricing Data During ECE Cold Restarts

Use this procedure in a BRM cloud native deployment when ECE persistence is enabled and you perform an ECE cold restart with **pricingLoadFromPersistence** set to **false**.

In this configuration, ECE does not load pricing data from the persistence database. Instead, ECE expects PDC to republish the pricing data. Ensure that PDC republishes the pricing data and that the data is available on all ECE nodes before Customer Updater loads customer data from BRM.

If Customer Updater loads customer data before pricing data is available on all ECE nodes, ECE might skip product data for some service types. Subsequent account, product, discount, or lifecycle updates for affected subscribers can then fail because ECE cannot find valid product or rating-context data.

If this condition occurs, you can use an incremental customer load to restore the missing customer data. To avoid the condition, publish the pricing data and verify that all ECE nodes have the data before Customer Updater begins loading customer data.

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **Note**
>
> Validate the cold restart procedure in a preproduction environment before you use it in production.

To ensure that all pricing data is available after an ECE cold restart:

1. In the **override-values.yaml** file for **oc-cn-ece-helm-chart**, verify that ECE persistence is enabled.
2. Set **pricingLoadFromPersistence** to **false**.
3. If Customer Updater must reload all customer data from BRM when ECE starts, set **customerLoadFromPersistence** to **false**.

   > **Note**
   >
   > Do not manually delete data from the ECE persistence tables as part of this procedure.

4. If you use a custom ECE startup script or another startup process, configure it so that Customer Updater loads customer data only after PDC publishes its data to ECE and pricing data is available on all ECE nodes.

   Use a readiness check when one is available. If you use a delay instead, validate the delay in a preproduction environment and set it according to the size of your pricing data and ECE cluster.
5. Start ECE and verify that Pricing Updater starts and is ready to receive pricing data from PDC.
6. Use **ImportExportPricing** to publish PDC data to ECE as separate operations in the following order:
   - **metadata**
   - **config**
   - **pricing**

   If your ECE startup process invokes the utility directly, run:

   ```
   ./ImportExportPricing -publish metadata -target ece
   ./ImportExportPricing -publish config -target ece
   ./ImportExportPricing -publish pricing -target ece
   ```

   For information about running PDC utilities in a BRM cloud native environment, see "About Running the PDC Utilities". For information about the utility options, see "ImportExportPricing" in *PDC Creating Product Offerings*.

   > **Note**
   >
   > Run separate **metadata**, **config**, and **pricing** publish operations. Do not use a single publish operation with **all**.

7. After the publish operations complete and the readiness check or validated delay confirms that pricing data is available, allow Customer Updater to load customer data.

Copyright © 2021, 2026, Oracle and/or its affiliates.

If your startup process controls Customer Updater, verify that the configured sequence enforces this order.

8. Confirm that ECE reaches the **USAGE_PROCESSING** state.
9. Verify that the ECE customer cache contains the active services and product data from BRM. Review the Pricing Updater and Customer Updater logs and verify that ECE does not skip service types or product data while loading customer data.

Re-Creating the ECE Schema After Deployment

To re-create the ECE schema, a table, or an index after you deploy the ECE Helm chart:

1. Delete the ECE Helm chart.
2. Delete the existing **ece-persistence-job** from your system:

   ```
   kubectl delete job ece-persistence-job -n BrmNameSpace
   ```

3. Install the Helm chart again:

   ```
   helm install ece --namespace BrmNameSpace oc-cn-ece-helm-chart [--no-hooks]
   ```

> **Note**
>
> Include **--no-hooks** only when the persistence database already contains everything that ECE requires.

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 the 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 the 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 the ECE cache at startup, but you can configure the pod to load customer data incrementally.

To incrementally load customer data into the ECE cache:

Copyright © 2021, 2026, Oracle and/or its affiliates.

1. Configure the customerupdater pod to load only an initial set of customers into the 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 the 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.
   - **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.
   - **migration.loader.customerGroupList**: In active-active deployments, set this to a comma-separated list of the customer group names to load, such as "customergroup1,customergroup2". The names must match the customerGroupConfigurations entries in the same file.
   - **migration.loader.incrementalCustomerLoadFilterQuery**: If this query references ACCOUNT_T, qualify POID_ID0 as account_t.poid_id0. For example, use:

     ```
     and account_t.poid_id0 NOT IN (select account_t.poid_id0 from account_t where
     account_t.poid_id0 <> 1 and ROWNUM <= 1)"
     ```
4. Perform a Helm upgrade by running this command:

   ```
   helm upgrade EceReleaseName oc-cn-ece-helm-chart --values OverrideValuesFile --
   namespace 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

31

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
- Processing CDR Generation in an Active-Active System
- Processing Usage Requests on Site Receiving Request
- Stopping ECE from Routing to a Failed Site
- Adding Fixed Site Back to ECE System
- Activating a Secondary Rated Event Formatter Instance
- About Conflict Resolution During the Journal Federation Process

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 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 the 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"
   ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
        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: "0.1.3.4"
   federatedCacheScheme:
      federationPort:
         brmfederated: 31016
         xreffederated: 31017
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
          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:
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
      primary:
         clusterName: "BRM"
         eceServiceName: ece-server
         eceServicefqdnOrExternalIP: "0.1.2.3"
      secondary:
         - clusterName: "BRM2"
           eceServiceName: ece-server
           eceServicefqdnOrExternalIp: "0.1.3.4"
   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.

   If you use incremental customer loading in this active-active configuration, set migration.loader.customerGroupList to the customer group names defined in **customerGroupConfigurations**, for example, "**customergroup1,customergroup2**". The names must match exactly across the active-active configuration and the customer loading configuration.

   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: "0.1.3.4:31500"
         name: "BRM2"
   - name: "customergroup2"
     clusterPreference:
       - priority: "2"
         routingGatewayList: "0.1.2.3:31500"
         name: "BRM"
       - priority: "1"
         routingGatewayList: "0.1.3.4:31500"
         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"
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
    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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **Note**
>
> In an active-active deployment, keep all primary and secondary Rated Event Formatter pods running. Set the **jmxEnabled** configuration entry to **true** for every primary and secondary Rated Event Formatter pod. ECE initializes site configuration only for JMX-enabled processes. A secondary Rated Event Formatter instance begins processing rated events only after it is activated when its corresponding primary instance is unavailable.
> If **jmxEnabled** is set to **false**, site configuration is not initialized and the Rated Event Formatter pod can fail to start.

The following shows example settings to add to the **override-values.yaml** file in your primary and secondary production sites:

```
ratedEventFormatter:
  ratedEventFormatterList:
    - jmxEnabled: true
      ratedEventFormatterConfiguration:
        name: "ref_site1_primary"
        clusterName: "BRM"
        primaryInstanceName: "REF-1"
        partition: "1"
        noSQLConnectionName: "noSQLConnection"
        connectionName: "oracle1"
        threadPoolSize: "2"
        retainDuration: "0"
        ripeDuration: "30"
        checkPointInterval: "20"
        siteName: "site1"
        pluginPath: "ece-ratedeventformatter.jar"
        pluginType:
"oracle.communication.brm.charging.ratedevent.formatterplugin.internal.Samp
leFormatterPlugInImpl"
        pluginName: "brmCdrPluginDC1Primary"
        noSQLBatchSize: "25"

    - jmxEnabled: true
      ratedEventFormatterConfiguration:
        name: "ref_site1_secondary"
        clusterName: "BRM2"
        partition: "1"
        noSQLConnectionName: "noSQLConnection"
        connectionName: "oracle2"
        threadPoolSize: "2"
        retainDuration: "0"
        ripeDuration: "30"
        checkPointInterval: "20"
        siteName: "site1"
        pluginPath: "ece-ratedeventformatter.jar"
        pluginType:
"oracle.communication.brm.charging.ratedevent.formatterplugin.internal.Samp
leFormatterPlugInImpl"
        pluginName: "brmCdrPluginDC1Primary"
        noSQLBatchSize: "25"
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
    - jmxEnabled: true
      ratedEventFormatterConfiguration:
        name: "ref_site2_primary"
        clusterName: "BRM2"
        primaryInstanceName: "REF-2"
        partition: "1"
        noSQLConnectionName: "noSQLConnection"
        connectionName: "oracle2"
        threadPoolSize: "2"
        retainDuration: "0"
        ripeDuration: "30"
        checkPointInterval: "20"
        siteName: "site2"
        pluginPath: "ece-ratedeventformatter.jar"
        pluginType:
"oracle.communication.brm.charging.ratedevent.formatterplugin.internal.Samp
leFormatterPlugInImpl"
        pluginName: "brmCdrPluginDC1Primary"
        noSQLBatchSize: "25"

    - jmxEnabled: true
      ratedEventFormatterConfiguration:
        name: "ref_site2_secondary"
        clusterName: "BRM"
        partition: "1"
        noSQLConnectionName: "noSQLConnection"
        connectionName: "oracle1"
        primaryInstanceName: "ref_site2_primary"
        threadPoolSize: "2"
        retainDuration: "0"
        ripeDuration: "30"
        checkPointInterval: "20"
        siteName: "site2"
        pluginPath: "ece-ratedeventformatter.jar"
        pluginType:
"oracle.communication.brm.charging.ratedevent.formatterplugin.internal.Samp
leFormatterPlugInImpl"
        pluginName: "brmCdrPluginDC1Primary"
        noSQLBatchSize: "25"
```

The **siteName** key specifies which ECE site's rated events are processed by the Rated Event Formatter instance, independent of which ECE site the Rated Event Formatter instance is running in. Primary Rated Event Formatter instances run in the same site as **siteName** and are normally active and processing the local ECE site's rated events. Secondary Rated Event Formatter instances run in a different ECE site as **siteName** and are only activated when the primary instance is unavailable.

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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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"
...
```

> **Note**
>
> If the secondary production site must receive configuration, pricing, and customer data only through federation from the primary production site, do not start ConfigLoader, Pricing Updater, or Customer Updater on the secondary production site. In the secondary production site's **override-values.yaml** file, set the replica counts for Pricing Updater and Customer Updater to **0**. Set the ConfigLoader replica count to **0** by using the supported ConfigLoader replica configuration for your version of **oc-cn-ece-helm-chart**. With these components stopped, ECE remains in the Initial state on the secondary production site until data is federated from the primary production site.

- If persistence is disabled, add the **ratedEventPublishers** and **NoSQLConnectionConfigurations** sections to your **override-values.yaml** file on 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"
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
                    name: "noSQLConnection2"
    ...
```

8. Deploy the ECE Helm chart (**oc-cn-ece-helm-chart**) 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 ecs-0 pod:

       ```
       kubectl label -n NameSpace po ecs-0 ece-jmx=ece-jmx-external
       ```

    b. Update the **/etc/hosts** file on the remote machine with the worker node of ecs-0:

       ```
       IP_OF_WORKER_NODE ecs-0.ece-server.namespace.svc.cluster.local
       ```

    c. Connect to JConsole:

       ```
       jconsole ecs-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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **Note**
>
> By default, the federation interceptor is invoked for events received at the destination site during **replicateAll** for conflict resolution. You can disable this by setting the **disableFederationInterceptor** attribute to **true** in the **charging.server** AppConfiguration MBean at the destination site before invoking **replicateAll** for Coherence services from the source site. The cache data will be replicated at the destination site without performing conflict resolution. Once **replicateAll** is complete, set the **disableFederationInterceptor** attribute to **false**.

10. Start federation from the secondary production site to the primary production site.

    To do so, connect to JConsole for an **`ecs-0`** pod in the secondary production site. For connection instructions, see 9.c.

    In the Coordinator MBean for each federated cache listed in 9.d, run **`start (PrimaryProductionSiteName)`**, where **`PrimaryProductionSiteName`** is the name of the primary production site. Run only the start operation on the secondary production site.

> **Note**
>
> Federation in the secondary-to-primary direction is not automatically initiated. You must start it manually.

11. When all pods are 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 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
    ```

12. Verify that the monitoring agent logs are collecting metrics.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Processing CDR Generation in an Active-Active System

When CDR generation is enabled in an active-active system, each production site can generate unrated CDRs for external systems. If a production site fails, the CDR Store retains in-progress CDR sessions, and subsequent 5G usage events are diverted to the CDR Gateway at another production site.

You can configure CDR Generator to mark partially processed CDRs as incomplete, identify possible duplicate usage updates, and add a custom reason when a CDR is closed. Downstream systems can use this information when processing CDRs generated across sites after a failover.

For information about these CDR Generator configurations and prerequisites, see "About CDR Generator in an Active-Active System" in *BRM System Administrator's Guide*.

Processing Usage Requests on 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 on 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 on 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 on production site 1. Similarly, if the usage request is received by production site 2, it is processed on 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 on the same site as the sharing group parent.

To configure the ECE active-active mode to process usage requests on 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 --
   namespace 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

- *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.

> **Note**
>
> Marking a site as failed affects traffic routing only. It does not prevent ECE components at the failed site from starting or initializing their configured database connections. Components configured for database access can therefore attempt authentication while the site is still marked as failed and before you run **recoverSite()**, even when the site is not processing usage requests or BRM traffic.

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
      ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

b. Connect through JConsole by running this command:

```
jconsole ece-monitoringagent-service-1:31020
```

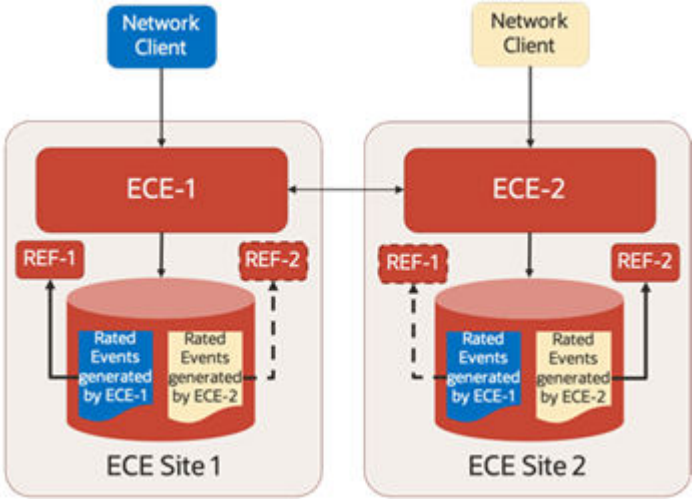
JConsole starts.

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. For example, in Figure 31-1, you could activate REF-2 in ECE Site 1 if REF-1 in ECE Site 1 goes down.

Figure 31-1 Sample Rated Event Formatter Instance in Active-Active Mode



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 the IP of the **ratedeventformatter** pod.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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.

About Conflict Resolution During the Journal Federation Process

In active-active ECE deployments, any changes to the ECE cache on one site are automatically federated to the ECE cache on other sites to keep the sites in sync. Most types of cache objects are replicated seamlessly. However, the following types can encounter conflicts during the federation process:

- Balance
- ActiveSession
- Customer
- TopUpHistory
- RecurringBundleIdHistory

Conflicts can arise for these object types when the same cache entry changes simultaneously at both sites or when the same cache entry changes at multiple sites while federation is down. For example, Site 1 processes Joe's purchase of 500 prepaid minutes, while Site 2 processes his usage of 20 prepaid minutes. ECE employs custom conflict resolution logic to detect and resolve these conflicting changes.

On rare occasions, ECE may be unable to resolve certain conflicts. When ECE cannot resolve a conflict, it:

- Keeps the change to the cache entry at the local ECE site but does not modify the cache entry at the receiving ECE site.
- Logs details about the unresolved conflict to the ECS log files, which are located in the *ECE_home*/**logs** directory, for subsequent review and possible corrective actions.

You can find details about the processing results of ECE federation change events, including cache conflict detection and resolution, in the **ece.federated.service.change.records** metric. See "ECE Federated Service Metrics" for more details.

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **Note**
>
> If an ECE site or federation between ECE sites remains down for a significant amount of time, the ECE conflict resolution logic may not be sufficient to synchronize the caches. In such cases, you must replace the ECE site's cache to achieve synchronization. See "Using startWithSync to Resynchronize an ECE's Cache Contents".

Using startWithSync to Resynchronize an ECE's Cache Contents

If an ECE site or federation between ECE sites remains down for an extended period, causing ECE cache data to become significantly out of sync, you can completely replace one ECE site's cache data with the contents of another ECE site. To do this, enable federation from the functioning origin ECE site to the recovering destination ECE site, and replicate all ECE cache data from the origin to the destination. You perform these two steps using the single Coherence **startWithSync** command.

The ECE cache data resynchronization procedure includes the following steps:

1. Truncate the ECE cache data and the corresponding ECE cache persistence database tables at the recovering destination ECE site.
2. At the recovering ECE destination site, set the **disableFederationInterceptor** ECE configuration MBean to **true**. This setting ensures that the federated cache data received from the origin ECE site always overwrites the recovering site's cache data.
   a. Create a JMX connection to ECE cloud native using JConsole. See "Creating a JMX Connection to ECE Using JConsole".
   b. In JConsole, expand the **ECE Configuration** node.
   c. Expand **charging.server**.
   d. Expand **Attributes**.
   e. Set the **disableFederationInterceptor** to **true**.
3. From the origin ECE site, run the **startWithSync()** operation with the recovering destination ECE site's name. Perform these tasks using JConsole:
   a. Expand the **Coherence** node, expand **Federation**, expand **BRMFederatedCache**, expand **Coordinator**, and then expand **Coordinator**. Click **startWithSync(***destination***)**, where *destination* is the recovering ECE site's name. Confirm that **ReplicateAllPercentComplete** in the Coherence destination MBean for the **BRMFederatedCache** service reaches 100% for all ecs nodes in the origin site before proceeding.
   b. Expand the **Coherence** node, expand **Federation**, expand **OfferProfileFederatedCache**, expand **Coordinator**, and then expand **Coordinator**. Click **startWithSync(***destination***)**. Confirm that **ReplicateAllPercentComplete** in the Coherence destination MBean for the **BRMFederatedCache** service reaches 100% for all ecs nodes in the origin site before proceeding.
   c. Expand the Coherence node, expand Federation, expand ReplicatedFederatedCache, expand Coordinator, and then expand Coordinator. Click **startWithSync(***destination***)**. Confirm that **ReplicateAllPercentComplete** in the Coherence destination MBean for the **BRMFederatedCache** service reaches 100% for all ecs nodes in the origin site before proceeding.

Copyright © 2021, 2026, Oracle and/or its affiliates.

d. Expand the Coherence node, expand Federation, expand XRefFederatedCache, expand Coordinator, and then expand Coordinator. Click **startWithSync(***destination***)**. Confirm that **ReplicateAllPercentComplete** in the Coherence destination MBean for the **BRMFederatedCache** service reaches 100% for all ecs nodes in the origin site before proceeding.

4. From the recovering destination ECE site, verify that all ECE cache data has been federated from the origin ECE site and continues to synchronize with ongoing changes, along with performing other standard ECE health checks.

5. At the recovering destination ECE site, verify that the **disableFederationInterceptor** ECE configuration MBean matches the value at the origin ECE site (the default is **true**).

Copyright © 2021, 2026, Oracle and/or its affiliates.

32
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**
>
> ECE server scaling and restart operations must be done only when partitions are balanced, and the current deployment uses the ecs StatefulSet for ECE server pods.

```
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
- 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **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** 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, the average will be the sum of CPU usage divided by four. 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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"
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

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** 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 namespace 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

> **Note**
>
> ECE server scaling and restart operations must be done only when partitions are balanced, and the current deployment uses the ecs StatefulSet for ECE server 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 or 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 32-1 lists the keys to use for restarting each ECE pod.

Table 32-1 Keys for Restarting ECE Pods

ECE Pod	Key
ecs	charging.ecs.restartCount

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 32-1 (Cont.) Keys for Restarting ECE Pods

ECE Pod	Key
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 --namespace BrmNameSpace
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

33

Monitoring ECE in a Cloud Native Environment

Learn how to monitor the system processes, such as memory and thread usage, in 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
- Setting Up Alerts with the ECE Alert Configuration Template
- Modifying Alert Rules and Thresholds
- Enabling ECE Metric Endpoints
- Best Practices and Important Notes
- Sample Prometheus Operator Configuration
- ECE Cloud Native Metrics
- ECE Cloud Native Alerts

About Monitoring ECE in a Cloud Native Environment

You can configure ECE to expose JVM, Coherence, and application metric data through a single HTTP 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.

> **Note**
>
> - ECE only exposes the metrics on an HTTP endpoint. It does not provide the Prometheus service.
> - Do not modify the **oc-cn-ece-helm-chart/templates/ece-ecs-metricsservice.yaml** file. It is used only during ECE startup and rolling upgrades. It is not used for monitoring.

You can monitor ECE operations by using external applications:

- **Prometheus**: Scrapes ECE metrics from the **/metrics** endpoint and monitors for issues that match your defined alert rules. When a condition is met, Prometheus generates an alert and sends it to Prometheus Alertmanager.
- **Prometheus Alertmanager**: Streamlines alert management by consolidating related alerts, suppressing repeats, and directing alerts to your chosen channels, such as email, Slack, or PagerDuty. For details, see the Prometheus Alertmanager documentation at: https://prometheus.io/docs/alerting/latest/overview/.
- **Grafana**: Displays ECE metrics and alerts in graphical dashboards.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Metrics exposed by ECE cover areas including system health, JVM, Coherence, Kubernetes, and gateway performance (Diameter, HTTP, EM, and so on). Oracle provides an Alert Configuration Template as a reference to help you set up and maintain your alerting policy.

> **Note**
>
> The template offers starting-point configurations and sample dashboards for alerting and visualization. You are responsible for deploying, customizing, and maintaining your monitoring solution.

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 cloud native exposes metric data through the following endpoint: **http://localhost:19612/metrics**.

2. Setting up a centralized metrics service, such as Prometheus Operator, to scrape metrics from the endpoint.

   For an example of how to configure Prometheus Operator to scrape ECE metric data, see "Sample Prometheus Operator Configuration".

3. Setting up a visualization tool, such as Grafana, to display your ECE metric data in a graphical format.

Setting Up Alerts with the ECE Alert Configuration Template

To configure and enable alerting for your ECE deployment:

1. Locate and extract the Alert Configuration Template from the ECE docker archive (**oc-cn-ece-docker-files-***version***.tgz**). The template is in the **docker_files/samples/monitoring/prometheus_rules/** directory. Alternatively, you can find the file in the *home*/**charging/temp/sample_data/metrics/prometheus_rules** in the ECS pod or **/scratch/ri-user-1/opt/OracleCommunications/ECE/ECE/oceceserver/sample_data/metrics/prometheus_rules in case of ON-PREM**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

2. Deploy Prometheus (standalone or using Prometheus Operator) in your Kubernetes cluster. Install Alertmanager for managing alert routing and notifications. Deploy Grafana for metrics visualization. See *BRM Compatibility Matrix* for version information.
3. Edit the ECE alert file, **eceAlertRules.yaml**, to set alert thresholds, durations, and logic. For more information on creating alert rules, see "Alerting Rules" in the Prometheus documentation at https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/. You can also consult the **README.md** file located in the same directory as the **eceAlertRules.yaml** file for information specific to ECE.
4. To use these rules in a standalone Prometheus environment:
   a. Add the file path to the **rule_files** section of your Prometheus configuration file (*Prometheus_home***/prometheus.yml)** from your local machine.

      ```
      rule_files:
          - "rules/eceAlertRules.yaml"
      ```

   b. Restart or reload Prometheus to apply the new rules.
5. To deploy these rules in a Prometheus Operator for Kubernetes (based on Custom Resource Definition) configuration, edit the Kubernetes **eceAlertRules.yaml** file:
   a. Add the following lines at the beginning of the file. Please ensure that the labels align with what is used in your environment. This is just a sample:

      ```
      apiVersion: monitoring.coreos.com/v1
          kind: PrometheusRule
          metadata:
              name: ece-alerts
              labels:
                  monitoring: shared
                  release: prometheus
        spec:
      ```

   b. Indent the entire **groups** section by 2 spaces to move it inside the **spec** section.
   c. Apply the file to your Kubernetes cluster by running the following command:

      ```
      kubectl apply -f eceAlertRules.yaml
      ```

6. Configure your preferred notification endpoints using Alertmanager or Grafana. For information about configuring this in Alertmanager, see "Configuration" in the Alertmanager documentation at: https://prometheus.io/docs/alerting/latest/configuration/. For information about configuring this in Grafana, see "Configure notifications" in the Grafana documentation at: https://grafana.com/docs/grafana/latest/alerting/configure-notifications/.
7. Import the sample Grafana dashboards from the *home*/**charging/temp/sample_data/metrics/grafana_dashboards** directory in the ECS pods, or, in an on-prem environment, from *scratch*/**ri-user-1/opt/OracleCommunications/ECE/ECE/oceceserver/sample_data/metrics/grafana_dashboards**, into Grafana. Connect Grafana to Prometheus as a data source. For more information, see "Prometheus data source" in the Grafana documentation at: https://grafana.com/docs/grafana/latest/datasources/prometheus/. These dashboards can help you to visualize metrics and alert statuses.
8. Validate all changes in a non-production environment before deploying to production.

Modifying Alert Rules and Thresholds

You can update alert rules, thresholds, and durations at any time by editing a copy of the **eceAlertRules.yaml** file present in the ECS pods at *sample_data*/**metrics**/*prometheus_rules* directory.

Copyright © 2021, 2026, Oracle and/or its affiliates.

After modifying any configuration, to apply your changes, run:

```
kubectl apply -f eceAlertRules.yaml
```

Review comments in YAML files for descriptions and operational advice. Always document and test changes in a non-production environment before deploying to production.

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 additional instances of ECE components, you must configure them 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 where 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 enables (**true**) or disables (**false**) the metrics service for that role.

   For example, these **override-values.yaml** entries would enable the metrics service for **ecs**.

   ```
   charging:
      labels: "ece"
      jmxport: "9999"
      …
      metrics:
         port: "19612"
      ecs:
         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 namespace 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 components expose metrics by default at their **/metrics** endpoint on an assigned port.

Copyright © 2021, 2026, Oracle and/or its affiliates.

To change which components expose metrics or to adjust port settings, edit the **override-values.yaml** file for **oc-cn-ece-helm-chart**. After updating the file, run a helm upgrade command to update Prometheus to point to the new metrics endpoint or port number.

This enables you to manage the scope and detail of data collected for alerting and visualization according to your operational requirements.

Best Practices and Important Notes

The following guidelines support effective use and configuration of alerting and monitoring tools:

- The Alert Configuration Template provides sample rules and dashboards. Always adapt these to your business requirements.
- ECE alert rules supplied by Oracle are general; customize them for your operational context.
- Test all monitoring and alerting configurations in a non-production environment first.
- Set up and validate notification channels as part of your deployment.
- Log-based alerting is not included and must be added separately if needed.
- Always validate rule syntax using Prometheus tools before deployment.
- Maintain version control for all Prometheus rule files to track configuration changes.
- Document all custom thresholds and rationale for future reference.

> **Note**
>
> Alertmanager integrations, such as email, PagerDuty, and Slack, require your own configuration. See the "Alertmanager" in the Prometheus documentation at: https://prometheus.io/docs/alerting/latest/overview/ for more information on integration specifics.

Sample Prometheus Operator Configuration

After installing Prometheus Operator, you configure it to scrape metrics from the ECE metric endpoint. The following shows sample entries you can use to create Prometheus Service and ServiceMonitor objects that scrape ECE metric data.

This sample creates a Service object that specifies to:

- Select all pods with the app label **ece**
- Scrape metrics from port 19612

```
apiVersion: v1
kind: Service
metadata:
  name: prom-ece-metrics
  labels:
    application: prom-ece-metrics
spec:
  ports:
    - name: metrics
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
      port: 19612
      protocol: TCP
      targetPort: 19612
  selector:
    app: ece
  sessionAffinity: None
  type: ClusterIP
  clusterIP: None
```

This sample creates a ServiceMonitor object that specifies to:

- Select all namespaces with **ece** in their name
- Select all Service objects with the application label **prom-ece-metrics**
- Scrape metrics from the HTTP path **/metrics** every 15 seconds

> **Note**
>
> In this sample, replace the sample value in **namespaceSelector.matchNames** with the ECE namespace.
>
> Also ensure that the labels defined for the ServiceMonitor match the labels selected by the Prometheus **serviceMonitorSelector**. For example, if your Prometheus configuration selects ServiceMonitor resources by using a **release** label, the ECE ServiceMonitor must include the matching **release** label value.
>
> If the namespace or labels do not match, Prometheus cannot discover or scrape the ECE metrics, and the ECE Grafana dashboards might not display data.

```
kind: ServiceMonitor
metadata:
  name: prom-ece-metrics
spec:
  endpoints:
    - interval: 15s
      path: /metrics
      port: metrics
      scheme: http
      scrapeTimeout: 10s
  namespaceSelector:
    matchNames:
      - ece
  selector:
    matchLabels:
      application: prom-ece-metrics
```

For more information about configuring Prometheus Operator, see https://github.com/prometheus-operator/prometheus-operator/tree/main/Documentation.

Copyright © 2021, 2026, Oracle and/or its affiliates.

ECE Cloud Native Metrics

ECE cloud native collects metrics in the following groups to produce data for monitoring your ECE components:

- BRS Metrics
- BRS Queue Metrics
- CDR Formatter Metrics
- Coherence Metrics
- Coherence Federated Service Metrics
- Diameter Gateway Metrics
- ECE Federated Service Metrics
- ECE Notification Metrics
- EM Gateway Metrics
- JVM Metrics
- Kafka JMX Metrics
- Kafka Client Metrics
- Micrometer Executor Metrics
- RADIUS Gateway Metrics
- Rated Event Formatter (REF) Metrics
- Rated Events Metrics
- Session Metrics

> **Note**
>
> Additional labels in the metrics indicate the name of the executor.

BRS Metrics

The BRS Metrics group contains the metrics for tracking the throughput and latency of the charging clients that use batch request service (BRS).

Table 33-1 lists the metrics in this group.

Table 33-1 BRS Metrics

Metric Name	Type	Description
ece.brs.message.receive	Counter	Tracks how many messages have been received.
ece.brs.message.send	Counter	Tracks how many messages have been sent.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 33-1 (Cont.) BRS Metrics

Metric Name	Type	Description
ece.brs.task.processed	Counter	Tracks the total number of requests accepted, processed, timed out, or rejected by the ECE component. You can use this to track the approximate processing rate over time, aggregated across all client applications, and so on.
ece.brs.task.pending.count	Gauge	Contains the number of requests that are pending for each ECE component.
ece.brs.current.latency.by.type	Gauge	Tracks the latency of a charging client for each service type in the current query interval. These metrics are segregated and exposed from the BRS layer per service type and include **event_type**, **product_type**, and **op_type** tags. 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 the maximum and average latency for an operation type since the last query. The maximum window size for collecting this data is 30 seconds, so the query has to be run 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**.
ece.brs.retry.queue.phase.count	Counter	Tracks the count of operations performed on the retry queue. **Additional Label: phase**
ece.brs.task.resubmit	Counter	Tracks the number of tasks that were scheduled for retry. **Additional Label: resubmitReason**
ece.brs.task.retry.count	Counter	Tracks the distributions of the number of retries performed for a retried request. **Additional Label: source, retries**
ece.brs.task.retry.distribution	Distribution Summary	Tracks the distributions of the number of retries performed for a retried request. **Additional Label: source**
ece.brs.queue.health.monitor.threshold.enabled	Gauge	Tracks the status of the threshold monitor: **0**: Stopped **1**: Started **Available Labels**: **applicationRole**, **configName**, **name**, **type**

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 33-1 (Cont.) BRS Metrics

Metric Name	Type	Description
ece.brs.queue.health.monitor.threshold.level	Gauge	Tracks the threshold-level index representing the monitor's severity stage as an integer: **0**: None **1**: LEVEL_ONE **2**: LEVEL_TWO **3**: LEVEL_THREE **Available Labels**: **applicationRole**, **configName**, **name**, **type**
ece.brs.queue.health.monitor.threshold.reading	Gauge	Tracks the monitored value against which the threshold values are evaluated. **Available Labels**: **applicationRole**, **configName**, **name**, **type**
ece.brs.queue.health.monitor.threshold.setpoint	Gauge	Specifies the threshold setpoint above which it enters a particular overload control level. **Available Labels**: **applicationRole**, **configName**, **level**, **name**, **thresholdType**, **type**
ece.brs.queue.health.monitor.threshold.state.transitions.total	Counter	Tracks the total number of transitions between the threshold states. **Available Labels**: **applicationRole**, **configName**, **name**, **type**

Reactor Netty ConnectionProvider Metrics

The Reactor Netty ConnectionProvider Metrics group contains standard metrics that provide insights into the pooled ConnectionProvider which supports built-in integration with Micrometer. Table 33-2 lists the metrics in this group.

For additional information about Reactor Netty ConnectionProvider Metrics, see the *Reactor Netty Reference Guide* in the Project Reactor documentation: https://projectreactor.io/docs/netty/1.1.17/reference/index.html.

Table 33-2 Reactor Netty ConnectionProvider Metrics

Metric Name	Type	Description
reactor.netty.connection.provider.total.connections	Gauge	Tracks the number of active or idle connections.
reactor.netty.connection.provider.active.connections	Gauge	Tracks the number of connections that have been successfully acquired and are in active use.
reactor.netty.connection.provider.max.connections	Gauge	Tracks the maximum number of active connections that are allowed.
reactor.netty.connection.provider.idle.connections	Gauge	Tracks the number of idle connections.
reactor.netty.connection.provider.pending.connections	Gauge	Tracks the number of requests that are waiting for a connection.
reactor.netty.connection.provider.pending.connections.time	Timer	Tracks the time spent waiting to acquire a connection from the connection pool.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 33-2 (Cont.) Reactor Netty ConnectionProvider Metrics

Metric Name	Type	Description
reactor.netty.connection.provider.max.pending.connections	Gauge	Tracks the maximum number of requests that are queued while waiting for a ready connection.

Reactor Netty HTTP Client Metrics

The Reactor Netty HTTP Client Metrics group contains standard metrics that provide insights into the HTTP client which supports built-in integration with Micrometer. Table 33-3 lists the metrics in this group.

For additional information about Reactor Netty HTTP Client Metrics, see "Reactor Netty Reference Guide" in the Project Reactor documentation for more information: https://projectreactor.io/docs/netty/1.1.17/reference/index.html.

Table 33-3 Reactor Netty HTTP Client Metrics

Metric Name	Type	Description
reactor.netty.http.client.data.received	DistributionSummary	Tracks the amount of data received, in bytes.
reactor.netty.http.client.data.sent	DistributionSummary	Tracks the amount of data sent, in bytes.
reactor.netty.http.client.errors	Counter	Tracks the number of errors that occurred.
reactor.netty.http.client.tls.handshake.time	Timer	Tracks the amount of time spent for TLS handshakes.
reactor.netty.http.client.connect.time	Timer	Tracks the amount of time spent connecting to the remote address.
reactor.netty.http.client.address.resolver	Timer	Tracks the amount of time spent resolving the remote address.
reactor.netty.http.client.data.received.time	Timer	Tracks the amount of time spent consuming incoming data.
reactor.netty.http.client.data.sent.time	Timer	Tracks the amount of time spent in sending outgoing data.
reactor.netty.http.client.response.time	Timer	Tracks the total time for the request or response.

BRS Queue Metrics

The BRS Queue Metrics group contains the metrics for tracking the throughput and latency of the BRS queue. Table 33-4 lists the metrics in this group.

Table 33-4 BRS Queue Metrics

Metric	Type	Description
ece.eviction.queue.size	Gauge	Tracks the number of items in the queue.
ece.eviction.queue.eviction.batch.size	Gauge	Tracks the number of queue items the eviction cycle processes in each iteration.
ece.eviction.queue.time	Timer	Tracks the amount of time items spend in the queue.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 33-4 (Cont.) BRS Queue Metrics

Metric	Type	Description
ece.eviction.queue.operation.duration	Timer	Tracks the time it takes to perform an operation on the queue.
ece.eviction.queue.scheduled.operation.duration	Timer	Tracks the time it takes to perform a scheduled operation on the queue.
ece.eviction.queue.operation.failed	Counter	Counts the number of failures for a queue operation.

CDR Formatter Metrics

The CDR Formatter Metrics group contains the metrics for tracking Charging Function (CHF) records. Table 33-5 lists the metrics in this group.

Table 33-5 CDR Formatter Metrics

Metric Name	Metric Type	Description
ece.chf.records.processed	Counter	Tracks the total number of CHF records the CDR formatter has processed.
ece.chf.records.purged	Counter	Tracks the total number of CHF records the CDR formatter purged.
ece.chf.records.loaded	Counter	Tracks the total number of CHF records the CDR formatter has loaded.

Coherence Metrics

All Coherence metrics that are available through the Coherence metrics endpoint are also accessible through the ECE metrics endpoint.

- For details of the Coherence metrics, see "Oracle Coherence MBeans Reference" in *Oracle Fusion Middleware Managing Oracle Coherence*
- For information about querying Coherence metrics, see "Querying for Coherence Metrics" in *Oracle Fusion Middleware Managing Oracle Coherence*

Coherence Federated Service Metrics

The Coherence Federated Service Metrics group contains metrics about conflicts that have occurred during the federation process in active-active disaster recovery systems. Table 33-6 lists the metrics in this group.

Table 33-6 Coherence Federated Service Metrics

Metric Name	Type	Description
coherence_federation_destination_total_committing_local_events	Counter	Tracks the total number of Coherence COMMITTING_LOCAL change event types.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 33-6 (Cont.) Coherence Federated Service Metrics

Metric Name	Type	Description
coherence_federation_destination_total_committing_local_local_only	Counter	Tracks the total number of cache entries that were set local only by a COMMITTING_LOCAL interceptor. Local only changes are not federated.
coherence_federation_destination_total_committing_local_modified	Counter	Tracks the total number of cache entries that were modified by a COMMITTING_LOCAL interceptor.
coherence_federation_destination_total_committing_local_rejected	Counter	Tracks the total number of cache entries that were rejected by a COMMITTING_LOCAL interceptor.
coherence_federation_destination_total_committing_local_unmodified	Counter	Tracks the total number of cache entries in COMMITTING_LOCAL events that were not modified by any COMMITTING_LOCAL event interceptors.
coherence_federation_destination_total_replicating_events	Counter	Tracks the total number of REPLICATING events.
coherence_federation_destination_total_replicating_modified	Counter	Tracks the total number of cache entries that were modified by a REPLICATING interceptor.
coherence_federation_destination_total_replicating_rejected	Counter	Tracks the total number of cache entries that were rejected by a REPLICATING interceptor.
coherence_federation_destination_total_replicating_unmodified	Counter	Tracks the total number of cache entries in REPLICATING events that were not modified by any REPLICATING event interceptors.
coherence_federation_origin_total_committing_remote_events	Counter	Tracks the total number of COMMITTING_REMOTE events.
coherence_federation_origin_total_committing_remote_modified	Counter	Tracks the total number of cache entries that were modified by a COMMITTING_REMOTE interceptor.
coherence_federation_origin_total_committing_remote_rejected	Counter	Tracks the total number of cache entries that were rejected by a COMMITTING_REMOTE interceptor.
coherence_federation_origin_total_committing_remote_unmodified	Counter	Tracks the total number of cache entries in COMMITTING_REMOTE events that were not modified by any COMMITTING_REMOTE event interceptors.

Diameter Gateway Metrics

The Diameter Gateway group contains metrics on events processed by the Diameter Gateway. Table 33-7 lists the metrics in this group.

Table 33-7 Diameter Gateway Metrics

Metric Name	Type	Description
ece.diameter.current.latency.by.type	Gauge	Tracks the latency of an Sy request for each operation type in the current query interval. The **SLR_INITIAL_REQUEST**, **SLR_INTERMEDIATE_REQUEST**, and **STR** operations are tracked.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 33-7 (Cont.) Diameter Gateway Metrics

Metric Name	Type	Description
ece.diameter.session.count	Gauge	Tracks the count of the currently cached diameter sessions. **Additional label: Identity**
ece.diameter.session.cache.capacity	Gauge	Tracks the maximum number of diameter session cache entries. **Additional label: Identity**
ece.diameter.session.sub.count	Gauge	Tracks the count of currently cached active ECE sessions. This is the count of sessions in the right side of the session map (Map*String,MapString,DiameterSession*).
ece.diameter.notification.requests.sent	Timer	Tracks the amount of time taken to send a diameter notification. **Additional labels: protocol, notificationType, result**
ece.requests.by.result.code	Counter	Tracks the total number of requests processed for each result code.

ECE Federated Service Metrics

The ECE Federated Service Metrics group contains metrics about conflicts that have occurred during the federation process in active-active disaster recovery systems. See "About Conflict Resolution During the Journal Federation Process" for more information.

Table 33-8 lists the metrics in this group.

Table 33-8 ECE Federated Service Metrics

Metric Name	Type	Description
ece.federated.service.change.records	Counter	Tracks the number of change records and tags them by conflict classification type: • notModified • error • alreadyConflictResolved • internallyModified • externallyModified • sameBinary • sameRevisionNumber • deleted • conflictDetected

ECE Notification Metrics

The ECE Notification Metrics group contains metrics for tracking the throughput, latency, and success or error rates for outgoing requests from the various ECE gateways grouped by application role. Table 33-9 lists the metrics in this group.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 33-9 ECE Notification Metrics

Metric Name	Type	Description
ece_notification_requests_sent_seconds	Timer	Tracks the latency of requests sent from a gateway to clients. In the metric, the **applicationRole** label can be: • HttpGateway • DiameterGateway • RadiusGateway
ece_notification_requests_sent_total	Counter	Counts the number of successfully delivered requests to ECE clients as well as the number of records delivered to the Dead Letter queue for unsuccessful requests. It keeps a separate count for the following record types: • Expired records from HTTP Gateway and Diameter Gateway. • Failed records from BRM Gateway, RADIUS Gateway, Diameter Gateway, and HTTP Gateway. • Successfully delivered records from BRM Gateway, RADIUS Gateway, Diameter Gateway, and HTTP Gateway.

EM Gateway Metrics

The EM Gateway Metrics group contains standard metrics that provide insights into the current status of your EM Gateway activity and tasks. Table 33-10 lists the metrics in this group.

Table 33-10 EM Gateway Metrics

Metric Name	Type	Description
ece.emgw.processing.latency	Timer	Tracks the overall time taken in the EM Gateway. **Additional label: handler**
ece.emgw.handler.processing.latency	Timer	Tracks the total processing time taken for each request processed by a handler. **Additional label: handler**
ece.emgw.handler.processing.latency.by.phase	Timer	Tracks the time it takes to send a request to the dispatcher or BRS. **Additional label: phase,handler**
ece.emgw.handler.error.count	Counter	Tracks the number of failed requests. **Additional label: handler, failureReason**
ece.emgw.opcode.formatter.error	Counter	Tracks the number of opcode formatter errors. **Additional label: phase**

HTTP Gateway Metrics

The HTTP Gateway group contains metrics on events processed by the HTTP Gateway. Table 33-11 lists the metrics in this group.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 33-11 HTTP Gateway Metrics

Metric Name	Type	Description
ece_http_server_api_requests_failed	Counter	Tracks the number of failed requests.
ece_http_current_latency_seconds	Timer/ Gauge	Tracks the current latency of a request for each operation type in seconds.

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 33-12 lists the metrics in this group.

Table 33-12 JVM Metrics

Metric Name	Type	Description
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.
coherence.os.system.cpu.load	Gauge	Contains the CPU load information 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.
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.
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, in bytes of Java heap and non-heap memory 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 in bytes, of Java memory space in use by the following JVM memory pools: **G1 Survivor Space**, **G1 Old Gen**, and **G1 Survivor Space**.
process.cpu.usage	Gauge	Contains the CPU percentage for each ECE component on the server. This data is collected from the corresponding MBean attributes by JVMs.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 33-12 (Cont.) JVM Metrics

Metric Name	Type	Description
process.files.open.files	Gauge	Contains the total number of file descriptors currently available for an ECE component and the descriptors in use for that ECE component.
system.load.average.1m	Gauge	Contains the system load average (the number of items waiting in the CPU run queue) 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.

Kafka JMX Metrics

The Kafka JMX Metrics group contains metrics for tracking the throughput and latency of the Kafka server and topics. Table 33-13 lists the metrics in this group.

Table 33-13 Kafka JMX Metrics

Metric Name	Type	Description
kafka.app.info.start.time.ms	Gauge	Indicates the start time in milliseconds.
kafka.producer.connection.close.rate	Gauge	Contains the number of connections closed per second.
kafka.producer.io.ratio	Gauge	Contains the fraction of time the I/O thread spent doing I/O.
kafka.producer.io.wait.time.ns.total	Counter	Contains the total time the I/O thread spent waiting.
kafka.producer.iotime.total	Counter	Contains the total time the I/O thread spent doing I/O.
kafka.producer.metadata.wait.time.ns.total	Counter	Contains the total time, in nanoseconds the producer has spent waiting on topic metadata.
kafka.producer.node.request.latency.max	Gauge	Contains the maximum latency, in milliseconds of producer node requests.
kafka.producer.record.error.total	Counter	Contains the total number of record sends that resulted in errors.
kafka.producer.txn.begin.time.ns.total	Counter	Contains the total time, in nanoseconds the producer has spent in beginTransaction.
kafka.producer.txn.commit.time.ns.total	Counter	Contains the total time, in nanoseconds the producer has spent in commitTransaction.

Kafka Client Metrics

The Kafka Client Metrics group contains metrics for tracking the throughput, latency, and performance of Kafka producer and consumer clients.

Copyright © 2021, 2026, Oracle and/or its affiliates.

> **Note**
>
> All Kafka producer metrics apply to the ECS, HTTP Gateway, Diameter Gateway, and BRM Gateway. All Kafka consumer metrics apply to the BRM Gateway, RADIUS Gateway, HTTP Gateway, and Diameter Gateway.

For more information about the available metrics, refer to the following Apache Kafka documentation:

- Producer Metrics: https://kafka.apache.org/36/generated/producer_metrics.html
- Consumer Metrics: https://kafka.apache.org/36/generated/consumer_metrics.html

Micrometer Executor Metrics

The Micrometer Executor Metrics group contains standard metrics that provide insights into the activity of your thread pool and the status of tasks. These metrics are created by Micrometer, a third party software. Table 33-14 lists the metrics in this group.

> **Note**
>
> The Micrometer API optionally allows a prefix to the name. In the table below, replace *prefix* with **ece.brs** for BRS metrics or **ece.emgw** for EM Gateway metrics.

Table 33-14 Micrometer Executor Metrics

Metric Name	Type	Description
prefix.executor.completed.tasks	FunctionCounter	Tracks the approximate total number of tasks that have completed execution. **Additional label: Identity**
prefix.executor.active.threads	Gauge	Tracks the approximate number of threads that are actively executing tasks. **Additional label: Identity**
prefix.executor.queued.tasks	Gauge	Tracks the approximate number of tasks that are queued for execution. **Additional label: Identity**
prefix.executor.queue.remaining.tasks	Gauge	Tracks the number of additional elements that this queue can ideally accept without blocking. **Additional label: Identity**
prefix.executor.pool.size.threads	Gauge	Tracks the current number of threads in the pool. **Additional label: Identity**
prefix.executor.pool.core.threads	Gauge	Tracks the core number of threads in the pool. **Additional label: Identity**
prefix.executor.pool.max.threads	Gauge	Tracks the maximum allowed number of threads in the pool. **Additional label: Identity**

Copyright © 2021, 2026, Oracle and/or its affiliates.

RADIUS Gateway Metrics

The RADIUS Gateway Metrics group contains standard metrics that track the customer usage of services. Table 33-15 lists the metrics in this group.

Table 33-15 RADIUS Gateway Metrics

Metric Name	Type	Description
ece.radius.sent.disconnect.message.counter.total	Counter	Tracks the number of unique disconnect messages sent to the Network Access Server (NAS), excluding the retried ones.
ece.radius.retried.disconnect.message.counter.total	Counter	Tracks the number of retried disconnect messages, excluding the total number of retries.
ece.radius.successful.disconnect.message.counter.total	Counter	Tracks the number of successful disconnect messages.
ece.radius.failed.disconnect.message.counter.total	Counter	Tracks the number of failed disconnect messages.
ece.radius.auth.extension.user.data.latency	Timer	Tracks the following: • The latency of converting a customer into an extension customer. • The latency of converting a balance into an extension balance map. • The latency of how long it takes to get a user data response from the user data.

Rated Event Formatter (REF) Metrics

The Rated Event Formatter (REF) Metrics group contains standard metrics that provide insights into the current status of your REF activity and tasks. Table 33-16 lists the metrics in this group.

Table 33-16 REF Metrics

Metric Name	Type	Description
ece.rated.events.checkpoint.interval	Gauge	Tracks the time, in seconds, used by the REF instance to read a set of rated events at a specific time interval.
ece.rated.events.ripe.duration	Gauge	Tracks the duration, in seconds, that rated events have existed before they can be processed.
ece.rated.events.worker.count	Gauge	Contains the number of worker threads used to process rated events.
ece.rated.events.phase.latency	Timer	Tracks the amount of time taken to complete a rated event phase. This only measures successful phases. **Additional labels: phase, siteName**
ece.rated.events.phase.failed	Counter	Tracks the number of rated event phase operations that have failed. **Additional labels: phase, siteName**
ece.rated.events.checkpoint.age	Gauge	Tracks the difference in time between the retrieved data and the current time stamp. **Additional labels: phase, siteName**

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 33-16 (Cont.) REF Metrics

Metric Name	Type	Description
ece.rated.events.batch.size	Gauge	Tracks the number of rated events retrieved on each iteration. **Additional labels: phase, siteName**

Rated Events Metrics

The Rated Events Metrics group contains metrics on rated events processed by ECE server sessions. Table 33-17 lists the metrics in this group.

Table 33-17 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.
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 database.
ece.rated.events.insert.failed	Counter	Contains the total number of rated events that failed to be inserted into the database.
ece.rated.events.purged	Counter	Contains the total number of rated events that are purged.
ece.requests.by.result.code	Counter	Tracks the total number of requests processed for each result code.

Session Metrics

The Session Metrics group contains metrics on ECE server sessions. Table 33-18 lists the metrics in this group.

Table 33-18 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.

ECE Cloud Native Alerts

ECE cloud native collects alert expressions in the following groups for monitoring your ECE components:

- CDR Formatter Alerts
- CDR Gateway Alerts
- Coherence Alerts

Copyright © 2021, 2026, Oracle and/or its affiliates.

- Diameter Gateway Alerts
- EM Gateway Alerts
- HTTP Gateway Alerts
- JVM Alerts
- Kubernetes Alerts
- Rated Event Formatter (REF) Alerts

The default alerts described in this section are located in the *sample_data*/**metrics**/*prometheus_rules*/**eceAlertRules.yaml** file. See "Setting Up Alerts with the ECE Alert Configuration Template" for more information.

CDR Formatter Alerts

The CDR Formatter Alerts group contains the alerts expressions for Charging Function (CHF) records. Table 33-19 lists the alerts in this group.

Table 33-19 CDR Formatter Alerts

Alert Name	Default Severity	Description
CdrFormatterProcessedRecordsIn15Mins	**critical**	An alert is triggered if the number of records processed by cdrformatter in last 15 minutes is zero.
CdrFormatterProcessedRecordsLow	**warning**	An alert is triggered if the number of records processed by cdrformatter in last 15 minutes is less than 5000.
CdrFormatterErrorMessageIn5Mins	**warning**	An alert is triggered if the rate of error messages increases by more than 10% in 5 minutes.

CDR Gateway Alerts

The CDR Gateway group contains alerts on events processed by the CDR Gateway. Table 33-20 lists the metrics in this group.

Table 33-20 CDR Gateway Alerts

Alert Name	Default Severity	Description
CdrGatewayErrorMessageIn5Mins	**warning**	An alert is triggered if the increase in error messages is greater than 10% in 5 minutes.

Coherence Alerts

All Coherence alert expressions used in the **eceAlertRules.yaml** file for Coherence are listed in Table 33-21.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 33-21 Coherence Alerts

Alert Name	Default Severity	Description
CoherenceCacheServiceThreadUtilizationCritical	**critical**	An alert is triggered when the 5-minute average Coherence cache service thread utilization is greater than 80%.
CoherenceCacheServiceThreadUtilizationMajor	**major**	An alert is triggered when the 5-minute average Coherence cache service thread utilization is 60% to 80%.
CoherenceCacheServiceThreadUtilizationMinor	**minor**	An alert is triggered when the 5-minute average Coherence cache service thread utilization is 40% to 60%.
CoherenceCacheWriteBehindQueueTooLarge	**major**	An alert is triggered if the write-behind queue size for the cache (excluding AggregateObjectUsage) is more than 1,000 for more than 5 minutes. This alert indicates a persistence queue overload.
CoherenceAggregateObjectUsageCacheLargeAndGrowing	**major**	An alert is triggered if the AggregateObjectUsage cache is abnormally large (over 1,000,000 entries) and still increasing (in the past hour).
EceCustomerCacheDown	**warning**	An alert is triggered if the customer cache size has dropped by more than 5% compared to the maximum amount in the last 4 hours.
EceServiceNotificationHigh	**critical**	An alert is triggered if the number of entries in the ServiceContext cache exceeds 200. This can cause the write-behind thread not to publish notifications, leading to service degradation.
CoherenceCustomerCacheSizeChangeCritical	**critical**	An alert is triggered if the size of the customer cache (in bytes) exceeds 90% change over a 24-hour period.
CoherenceCustomerCacheSizeChangeMajor	**major**	An alert is triggered if the size of the customer cache (in bytes) lies in between 80% and 90% change over a 24-hour period.
CoherenceCustomerCacheSizeChangeMinor	**minor**	An alert is triggered if the size of the customer cache (in bytes) lies in between 60% and 80% change either over a 24-hour period.
CoherenceCustomerCacheSizeChangeCritical	**critical**	An alert is triggered if the customer size (in bytes) exceeds 90% change either over a 24-hour period or for total size.
EceFederationCacheReplicationIncomplete	**major**	An alert is triggered if the federation cache replication value is below 100% over a period of time, during the replicate-all operation.
EceFederationCacheError	**major**	An alert is triggered if the rate of federation cache errors increases in last 5 minutes. The alert indicates errors in federation operations.
EceFederationCacheBacklogCritical	**critical**	An alert is triggered if the total federation journal size exceeds 60% of the maximum configured flash journal size in Coherence journal config.
EceFederationCacheBacklogMajor	**major**	An alert is triggered if the total federation journal size is 40% to 60% of the maximum configured flash journal size in Coherence journal config.
EceFederationCacheBacklogMinor	**minor**	An alert is triggered if the total federation journal size is 20% to 40% of the maximum configured flash journal size in Coherence journal config.
CoherenceHAStatusEndangered	**critical**	An alert is triggered if the Coherence partition assignment HA status is **ENDANGERED**. This prevents potential data loss in high-availability systems.
CoherenceServicePartitionsUnbalanced	**critical**	An alert is triggered if more than 10 Coherence service has unbalanced partitions for more than 5 minutes.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 33-21 (Cont.) Coherence Alerts

Alert Name	Default Severity	Description
CoherenceServiceRequestPendingHigh	**major**	An alert is triggered if the Coherence service requests are pending.
CoherenceServiceRequestPendingTooLong	**major**	An alert is triggered if the Coherence service requests are pending for longer than 10,000 milliseconds.
CoherenceServiceTaskBacklogHigh	**major**	An alert is triggered if there are more than 5 service tasks in a backlog state.
EceFederationCacheBacklogIncrease	**major**	An alert is triggered when the federation cache backlog is more than 12 in 5 minutes.
CoherenceCustomerCacheSizeCritical	**critical**	An alert is triggered when the Coherence Customer cache size is greater than 15 GB.
CoherenceCustomerCacheSizeMajor	**major**	An alert is triggered when the Coherence Customer cache size is between 10 and 15 GB.
CoherenceCustomerCacheSizeMinor	**minor**	An alert is triggered when the Coherence Customer cache size is between 5 and 10 GB.

Diameter Gateway Alerts

The Diameter Gateway group contains alert expressions on events processed by the Diameter Gateway. Table 33-22 lists the alerts in this group.

Table 33-22 Diameter Gateway Alerts

Alert Name	Default Severity	Description
DiameterGatewayPendingTaskHigh	**critical**	An alert is triggered if the pods have 100 or more pending tasks for 5 minutes.
DiameterGatewaySyRequestFailed	**warning**	An alert is triggered when the failure rate for Sy requests is between 5% and 10%.
DiameterGatewaySyRequestFailedHigh	**critical**	An alert is triggered when the failure rate for Sy requests are beyond 10%.
DiameterGatewayGyRequestFailed	**warning**	An alert is triggered when the failure rate for Gy requests is between 5% and 10%.
DiameterGatewayGyRequestFailedHigh	**critical**	An alert is triggered when the failure rate for Gy requests are beyond 10%.
DiameterGatewayCurrentLatency	**warning**	An alert is triggered if the batch request server (BRS) latency is between 100 milliseconds and 500 milliseconds.
DiameterGatewayCurrentLatencyHigh	**critical**	An alert is triggered if the BRS latency is more than 300 milliseconds.
DiameterGatewayThroughput	**warning**	An alert is triggered if the Diameter Gateway throughput is less than 50 in 30 minutes.
EceCoherenceStateIncorrect	**critical**	An alarm is triggered if the pod is not in the usage processing state (10).

Copyright © 2021, 2026, Oracle and/or its affiliates.

EM Gateway Alerts

The EM Gateway Alerts group contains standard alert expressions for your EM Gateway activity and tasks. Table 33-23 lists the alerts in this group.

Table 33-23 EM Gateway Alerts

Alert Name	Default Severity	Description
EceEmGwLatency	**warning**	An alert is triggered if the latency in the EM Gateway exceeds 0.1 seconds.
EmGwHandlerFailedRequestError	**critical**	An alert is triggered if the number of recorded errors for the EM Gateway handler is more than zero in the last 5 minutes.
EceEmgwHandlerProcessingLatencyHigh	**warning**	An alert is triggered if the EM Gateway handler latency for phases that exceeds 0.1 seconds.

HTTP Gateway Alerts

The HTTP Gateway group contains alerts on events processed by the HTTP Gateway. Table 33-24 lists the alerts in this group.

Table 33-24 HTTP Gateway Alerts

Alert Name	Default Severity	Description
HttpGatewayLatencyHigh	**warning**	An alert is triggered if the request latency at the 95th percentile is greater than 0.1 seconds.
HttpRequestFailureRateHigh	**critical**	An alert is triggered if the failure rate is greater than 10%.
HttpGatewayErrorMessageCountLast5Mins	**warning**	An alert is triggered if the error message rate increase is greater than 10% over a period of 5 minutes.

JVM Alerts

The JVM Alerts group contains standard alert expressions about the central processing unit (CPU) and memory utilization of JVMs which are members of the ECE grid. Table 33-25 lists the alerts in this group.

Table 33-25 JVM Alerts

Alert Name	Default Severity	Description
EcsJvmHeap90Percent	**critical**	An alert is triggered if the heap size used is more than 90% for ECS.
EcsJvmHeap80Percent	**major**	An alert is triggered if the heap size used is between 80 and 90% for ECS.
HttpGatewayJvmHeap90Percent	**critical**	An alert is triggered if the heap size used is more than 90% for HTTP Gateway.
HttpGatewayJvmHeap80Percent	**major**	An alert is triggered if the heap size used is between 80 and 90% for HTTP Gateway.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 33-25 (Cont.) JVM Alerts

Alert Name	Default Severity	Description
CdrGatewayJvmHeap90Percent	**critical**	An alert is triggered if the heap size used is more than 90% for CDR Gateway.
CdrGatewayJvmHeap80Percent	**major**	An alert is triggered if the heap size used is between 80 and 90% for CDR Gateway.
CdrFormatterJvmHeap90Percent	**critical**	An alert is triggered if the heap size used is more than 90% for CDR Formatter.
CdrFormatterJvmHeap80Percent	**major**	An alert is triggered if the heap size used is between 80 and 90% for CDR Formatter.
HighJvmGcPauseTotal	**warning**	An alert is triggered if the rate of garbage collection pause is more than 0.1 seconds over a period of 5 minutes.

Kubernetes Alerts

Table 33-26 lists the metrics in the Kubernetes group of metrics.

Table 33-26 Kubernetes Alerts

Alert Name	Default Severity	Description
ECSMinimumPods	**major**	An alert is triggered if not all of the replica pods are available for ECS.
EceEmGwMinimumPods	**warning**	An alert is triggered if not all of the replica pods are available for EM Gateway.
DiameterGatewayMinimumPods	**critical**	An alert is triggered if not all of the replica pods are available for Diameter Gateway.
HttpGatewayMinimumPods	**major**	An alert is triggered if not all of the replica pods are available for HTTP Gateway.
CdrGatewayMinimumPods	**major**	An alert is triggered if not all of the replica pods are available for CDR Gateway.
CdrFormatterMinimumPods	**warning**	An alert is triggered if not all of the replica pods are available for CDR Formatter.

Rated Event Formatter (REF) Alerts

The Rated Event Formatter (REF) Alerts group contains standard alert expressions that provide insights into the status of your REF activity and tasks. Table 33-27 lists the alerts in this group.

Table 33-27 REF Alerts

Alert Name	Default Severity	Description
EceRatedEventThroughput	**warning**	An alert is triggered if the number of rated events throughput is less than 3000 over a period of one hour.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table 33-27 (Cont.) REF Alerts

Alert Name	Default Severity	Description
EceRatedEventsPurgedLow	**warning**	An alert is triggered if the number of rated events purge falls below 5000 in the previous hour if at least one event was added.
EceRatedEventsInsertionRateLow	**critical**	An alert is triggered if the rated events insertion rate falls below 90% of the rate of the cached events.
EceRatedEventsInsertFailed	**critical**	An alert is triggered if the rate of rated event insertion has increased for 5 minutes.
EceRatedEventFormatterCacheHigh	**critical**	An alert is triggered if there are more than 1,000,000 cached events.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Part VI

Troubleshooting BRM Cloud Native Services

This part describes how to troubleshoot issues in your Oracle Communications Billing Revenue Management (BRM) cloud native services.

It contains the following chapters:

- Troubleshooting BRM Cloud Native
- Troubleshooting ECE Cloud Native
- Troubleshooting PDC Cloud Native

Copyright © 2021, 2026, Oracle and/or its affiliates.

34

Troubleshooting BRM Cloud Native

Learn how to troubleshoot issues in Oracle Communications Billing and Revenue Management (BRM) cloud native.

Topics in this document:

- Authorization Error When Running pin_update_balances Utility
- Order Processing Fails with Kafka DM Connection Errors
- Grafana Dashboards are Empty
- SSL/TLS Handshake Error When Connecting to CM Through an External Load Balancer

Authorization Error When Running pin_update_balances Utility

Problem Statement

When running the **pin_update_balances** utility, you may encounter an authorization error and the process fails to complete.

Sample Error Message

```
E Wed May 15 12:16:35 2026 <hostname> pin_update_balances:2579170
pin_mta.c:1076
3:<hostname>:pin_update_balances:2579170:1778034432:4:1715764595851079:8:bill_
inv_pymt_sub.0.0.0.1::::-1 normal error <location=PIN_ERRLOC_FM:5
class=PIN_ERRCLASS_SYSTEM_DETERMINATE:1 errno=PIN_ERR_AUTHORIZATION_FAIL:64>
<facility=0 msg_id=0 version=0>
```

Cause

This issue occurs because the user **bill_inv_pymt_sub.0.0.0.1** is not authorized to run the PCM_OP_CUST_UPDATE_ROLE opcode.

Solution

1. Update the following entries in the **testnap** ConfigMap file (**configmap_pin_conf_testnap.yaml**).

   > **Note**
   >
   > Do not change these entries in the CM ConfigMap file (**configmap_pin_conf_cm.yaml**).

   Original entries:

   ```
   - nap login_name root.0.0.0.1
   - - userid 0.0.0.1 /service/pcm_client 1
   ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

Update it to:

```
- nap login_name super_user.0.0.0.1
- - userid 0.0.0.1 /service/pcm_client 3
```

2. Run the **helm upgrade** command to update your Helm release.
3. Update the flist for PCM_OP_CUST_UPDATE_ROLE:

```
0 PIN_FLD_POID POID [0] 0.0.0.1 /config/role 412 0
0 PIN_FLD_PERMITTEDS ARRAY [0] allocated 20, used 3
1   PIN_FLD_BASE_OPCODES STR [0] ""
1   PIN_FLD_CLASSES STR [0] ""
1   PIN_FLD_FM_OPCODES STR [0]
"PCM_OP_ACT_REQUEST_CREATE,PCM_OP_ACT_REQUEST_RETRIEVE,PCM_OP_ACT_USAGE,PCM_OP_BILL_M
AKE_BILL"
0 PIN_FLD_PERMITTEDS ARRAY [3] allocated 20, used 3
1   PIN_FLD_BASE_OPCODES STR [0] "PCM_OP_READ_FLDS"
1   PIN_FLD_CLASSES STR [0] "/billinfo,/payinfo/*,/item/misc,/account,/config/
payment,/bill,/event/*,/data/trial_bill_sequence,/item/installment/debit,/ledger_"
1   PIN_FLD_FM_OPCODES STR [0] ""
```

4. Run the PCM_OP_CUST_UPDATE_ROLE opcode using **testnap**. For more information on how to run **testnap** in cloud native environments, see "Running Load Utilities Through Configurator Jobs".
5. Return the entries in the **configmap_pin_conf_testnap.yaml** file to their original values:

```
- nap login_name root.0.0.0.1
- - userid 0.0.0.1 /service/pcm_client 1
```

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

Order Processing Fails with Kafka DM Connection Errors

Problem Statement

Order processing fails intermittently when the Connection Manager (CM) cannot connect to the Kafka DM to publish events. The CM log contains messages similar to the following:

```
bad cm_change_dm_connection, new db=<DB>, current db=<DB>,
err=26(PIN_ERR_DM_CONNECT_FAILED), errno=9(Bad file descriptor)

Op (PCM_OP_TRANS_OPEN) send failed in pcm_op
```

The connection failure does not resolve automatically.

Cause

This problem can occur when the EAI Java Server (JS) and the Kafka DM use different thread pool sizes. Over time, this mismatch can lead to thread exhaustion and connection failures.

Solution

To fix this, do the following:

1. Verify the connection pool settings in the EAI Java Server (JS) source configuration in the **infranet.properties** file of the **eai_js** module used for your deployment.
2. Set the **infranet.connection.pool.enable** entry to **true**. Record the value of the **infranet.connection.pool.size** entry. If you change either entry, apply the updated EAI Java Server configuration before continuing.
3. Open the **override-values.yaml** file for **oc-cn-helm-chart**.

Copyright © 2021, 2026, Oracle and/or its affiliates.

4. Set the **ocbrm.dm_kafka.deployment.poolSize** key to the value of the **infranet.connection.pool.size** entry that you recorded for the EAI Java Server.

   Use a matching value from **1** through **2000** that meets your deployment requirements.

5. Save and close the file.

6. Run the following command to update the BRM Helm release:

   ```
   helm upgrade BrmReleaseName oc-cn-helm-chart \
       --values OverrideValuesFile --namespace BrmNameSpace
   ```

   where:

   - *BrmReleaseName* is the release name for **oc-cn-helm-chart**.
   - *OverrideValuesFile* is the file name and path of the **override-values.yaml** file.
   - *BrmNameSpace* is the namespace that contains the BRM Kubernetes objects.

7. Retrieve the BRM pod names:

   ```
   kubectl get pods -n BrmNameSpace
   ```

8. In the command output, identify the Kafka DM pod name.

9. Delete the Kafka DM pod to restart it:

   ```
   kubectl delete pod DmKafkaPodName -n BrmNameSpace
   ```

   Kubernetes creates a replacement Kafka DM pod.

10. Verify that the replacement Kafka DM pod has a status of Running:

    ```
    kubectl get pods -n BrmNameSpace
    ```

11. Retry the failed operation. Verify that the CM log no longer reports PIN_ERR_DM_CONNECT_FAILED.

> **Note**
>
> Restarting the Kafka DM pod without correcting the thread pool mismatch restores service only temporarily.

For more information, see "Integrating with Kafka Servers" and "Restarting BRM Pods".

Grafana Dashboards are Empty

Problem Statement

The sample Grafana dashboards included in the **oc-cn-docker-files-16.26.0.*x*.*x*.tgz** package do not display data.

Cause

This issue can occur if ServiceMonitor resources are missing, not selected by Prometheus, or configured with labels or namespaces that do not match the Prometheus configuration. It can also occur when the BRM services are exposing metrics, but the collected metrics do not include labels such as pod or namespace that are used by the dashboard filters.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Solution

To fix this issue:

1. In the Prometheus UI, confirm that the BRM targets are listed and healthy.
2. If you use Prometheus Operator, confirm that the ServiceMonitor resources exist for the CM and dm-oracle metrics and that they are selected by Prometheus.
3. Verify that the BRM services expose the **/metrics** endpoint.
4. In Grafana, confirm that dashboard query filters such as pod and namespace match the labels available in Prometheus for the collected metrics.

   If the BRM metrics are available in Prometheus but not shown in Grafana, the dashboard queries may be filtering on labels that are missing or different from the labels in the Prometheus metrics.

SSL/TLS Handshake Error When Connecting to CM Through an External Load Balancer

Problem Statement

A BRM client fails to connect to CM through an external load balancer. The connection fails during the SSL/TLS handshake.

Cause

This issue can occur when the client does not trust the certificate presented by the load balancer, or when the load balancer uses a plain TCP connection to connect to an SSL/TLS-enabled CM.

Solution

1. Verify that the Oracle wallet used by the client trusts the certificate presented by the load balancer.
2. Verify that the host name or IP address in the certificate matches the load balancer host name or IP address used by the client.

   If the load balancer connects to CM using SSL/TLS, verify that the load balancer is configured to start an SSL/TLS connection to CM. See “Configuring an External Load Balancer for Connection Manager SSL/TLS Connections” for instructions.

Copyright © 2021, 2026, Oracle and/or its affiliates.

35

Troubleshooting ECE Cloud Native

Learn how to troubleshoot issues in Oracle Communications Elastic Charging Engine (ECE) cloud native.

Topics in this document:

- Troubleshooting Incorrect Charges for Mid-Session Product Purchase
- Troubleshooting Missing Login Identifier for Existing Accounts After Enabling ECE
- Troubleshooting Long Cycle Grant Validity Splitting
- Reloading the Charging Configuration Reload Fails Due to Missing Cluster Port

Troubleshooting Incorrect Charges for Mid-Session Product Purchase

When a customer purchases a product during an active usage session, ECE may split the session at the purchase time and rate each segment separately. This split can cause ECE to rate the usage before and after the purchase as separate segments with split quantities.

For example, if ECE receives a 35-second usage request, and the customer purchases a product after 33 seconds, ECE splits the session at the purchase time and rates the first 33 seconds and the remaining 2 seconds as separate segments. For this example, assume when the customer uses a charge offer with a rate plan that:

1. Applies a zero rate when the quantity is less than or equal to two seconds.
2. Applies a non-zero rate when the quantity is greater than two seconds.

In this case, ECE rates the 33 second segment at the non-zero rate and rates the 2 second segment at zero. This outcome differs from the expected behavior, where ECE rates the full 35 seconds at the non-zero rate.

By default, ECE considers all products purchased by the customer for tariff splits. However, you can configure ECE to consider only products eligible for the current usage request by setting the **productsConsideredForTariffSplit** attribute in the *ECE_home***/config/management/charging-settings.xml** file to **CURRENT**.

The **productsConsideredForTariffSplit** attribute can be set to one of the following values:

- **ALL**: ECE uses the default behavior and splits the tariff when a customer purchases a product during a session.
- **CURRENT**: ECE considers only the products eligible for the current usage request and does not split the tariff when the customer does any of the following:
  - Purchases a product of a different product type (for example, a data product during a voice session).
  - Purchases an offering that ECE does not use to rate the current request (for example, a voice offering that does not apply to the current voice request).

Copyright © 2021, 2026, Oracle and/or its affiliates.

- Purchases an offering that grants balance elements that ECE does not use to rate the current request.

You can configure tariff split behavior in cloud native deployments during installation, before a system restart, or at runtime. For instructions, see:

- Configuring Tariff Split Behavior During Installation or Before Restart.
- Configuring Tariff Split Behavior at Runtime

Configuring Tariff Split Behavior During Installation or Before Restart

To configure tariff split behavior, do the following:

1. In the **override-values.yaml** file for the **oc-cn-ece-helm-chart**, set the **productsConsideredForTariffSplit** key to **CURRENT**.
2. Run the **helm install** command to deploy the ECE Helm chart:

   `helm install` *`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. It must be different from the one used for the BRM Helm chart.
   - *BrmNameSpace* is the namespace in which BRM Kubernetes objects reside for the BRM Helm chart.
   - *OverrideValuesFile* is the path to a YAML file that overrides the default configurations in the **oc-cn-ece-helm-chart/values.yaml** file.

Configuring Tariff Split Behavior at Runtime

To configure tariff split behavior at runtime:

1. Connect to a JMX editor, such as JConsole. See "Creating a JMX Connection to ECE Using JConsole".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. Set the **productsConsideredForTariffSplit** attribute to **CURRENT**.

Troubleshooting Missing Login Identifier for Existing Accounts After Enabling ECE

When you enable ECE in an environment where BRM and PDC are already live in production, ECE may synchronize only a subset of information for accounts that existed before ECE was enabled. As a result, **publicIdentifiers** may be populated with aliases, but the login identifier may be missing, which can prevent you from performing actions on those accounts. However, you can occur because, by default, ECE loads the login identifier only when **publicIdentifiers** is null or empty.

To control whether ECE loads the login identifier along with aliases in its PublicUserIdentity (PUI) cache by setting the **loadLogins** attribute in the *ECE_home***/config/management/**

Copyright © 2021, 2026, Oracle and/or its affiliates.

charging-settings.xml file to **true**. The **loadLogins** attribute can be set to one of the following values:

- **true**: ECE loads the login identifier along with aliases into its PUI cache even if aliases are present.
- **false**: ECE loads only aliases into its PUI cache and loads the login identifier only when the alias list is null or empty. This is the default value.

You can configure login identifier loading in cloud native deployments during installation, before a system restart, or at runtime. For instructions, see:

- Configuring Login Identifier Loading During Installation or Before Restart
- Configuring Login Identifier Loading at Runtime

Configuring Login Identifier Loading During Installation or Before Restart

To configure login identifier loading, do the following:

1. In the **override-values.yaml** file for the **oc-cn-ece-helm-chart**, set the **loadLogins** key to **true**.
2. Run the **helm install** command to deploy the ECE Helm chart:

   `helm install` *`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. It must be different from the one used for the BRM Helm chart.
   - *BrmNameSpace* is the namespace in which BRM Kubernetes objects reside for the BRM Helm chart.
   - *OverrideValuesFile* is the path to a YAML file that overrides the default configurations in the **oc-cn-ece-helm-chart/values.yaml** file.

Configuring Login Identifier Loading at Runtime

To configure login identifier loading at runtime:

1. Connect to a JMX editor, such as JConsole. See "Creating a JMX Connection to ECE Using JConsole".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. Set the **loadLogins** attribute to **true**.

Troubleshooting Long Cycle Grant Validity Splitting

When a bill unit has PIN_FLD_ACTG_LAST_T and PIN_FLD_ACTG_NEXT_T that span multiple months, your customers expect grant validity to be calculated monthly. However, creating an account with an older effective date can extend the billing cycle across multiple months. This happens because, by default, ECE grants overage buckets based on the full billing cycle period, whereas BRM splits them monthly.

Copyright © 2021, 2026, Oracle and/or its affiliates.

When you enable monthly splitting for long accounting cycles, ECE grants validity in one-month periods, as shown in the following examples. For a current long accounting cycle, ECE determines the applicable monthly validity period for the request by using PIN_FLD_ACTG_NEXT_T as the reference boundary. ECE evaluates one month intervals to identify the period that contains the request time and ensures that the validity start time does not precede PIN_FLD_ACTG_LAST_T.

- A billing unit has PIN_FLD_ACTG_LAST_T August 1 and PIN_FLD_ACTG_NEXT_T October 1, so the accounting cycle spans two months. ECE grants a request on August 29 that is valid from August 1 through September 1, and ECE grants a later request in September that is valid from September 1 through October 1, instead of granting one validity period from August 1 through October 1.
- A billing unit has PIN_FLD_ACTG_LAST_T August 15 and PIN_FLD_ACTG_NEXT_T November 15, so the accounting cycle spans three months and starts mid-month. The system receives requests on October 5 and November 10. If you enable monthly splitting, ECE handles the two requests in different validity periods (September 15 through October 15 and October 15 through November 15, respectively). If you do not enable monthly splitting, ECE applies both requests to the same validity period from August 15 through November 15.

To control whether ECE splits long-cycle grant validity monthly, set **splitLongCyclePeriod** to one of the following values:

- **true**: Split the grant validity on a monthly basis instead of using the full cycle.
- **false**: Use the full long cycle for grant validity. This is the default value.

You can control whether ECE splits long-cycle grant validity monthly in cloud native deployments during installation, before a system restart, or at runtime. For instructions, see:

- Configuring Long Cycle Validity Splitting During Installation or Before Restart
- Configuring Long Cycle Validity Splitting at Runtime

Configuring Long Cycle Validity Splitting During Installation or Before Restart

To configure long cycle validity splitting, do the following:

1. In the **override-values.yaml** file for the **oc-cn-ece-helm-chart**, set the **splitLongCyclePeriod** key to **true**.
2. Run the **helm install** command to deploy the ECE Helm chart:

   helm install *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. It must be different from the one used for the BRM Helm chart.
   - *BrmNameSpace* is the namespace in which BRM Kubernetes objects reside for the BRM Helm chart.
   - *OverrideValuesFile* is the path to a YAML file that overrides the default configurations in the **oc-cn-ece-helm-chart/values.yaml** file.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Configuring Long Cycle Validity Splitting at Runtime

To configure long cycle validity splitting at runtime:

1. Connect to a JMX editor, such as JConsole. See "Creating a JMX Connection to ECE Using JConsole".
2. Expand the **ECE Configuration** node.
3. Expand **charging.server**.
4. Expand **Attributes**.
5. Set the **splitLongCyclePeriod** attribute to **true**.

Reloading the Charging Configuration Reload Fails Due to Missing Cluster Port

When running the **chargingConfigurationReloader** Kubernetes job to update a **charging-settings.xml** entry, such as **updateResponseTimeout**, the job may fail with an error similar to this:

```
Reloading charging.emGatewayConfigurations Mbean.
Exception in thread "main" java.security.PrivilegedActionException
Caused by: org.springframework.beans.factory.BeanCreationException
Caused by: java.lang.IllegalStateException:
This ContinuousQueryCache is disconnected. Retry the operation again.
```

This error usually occurs when the Coherence cluster connection is incomplete or misconfigured.

To resolve the issue, ensure that the cluster port is included in the **charging-configuration-reloader.yaml** file. In the file, find the following line:

```
- name: COH_WKA_PORT
  value: {{ .Values.charging.coherencePort | quote }}
```

Update the line with the appropriate port. For example, this sets the Coherence port to 31016:

```
- name: COH_WKA_PORT
  value: {{ .Values.charging.coherencePort | default 31016 }}
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

36

Troubleshooting PDC Cloud Native

Learn how to troubleshoot issues in Oracle Communications Pricing Design Center (PDC) cloud native.

Topics in this document:

- Troubleshooting ImportExportPricing Errors
- Unable to Import a Large Number of XML Files
- SyncPDC Fails After Cloning BRM and PDC Databases
- Publishing Pricing Components to ECE Fails
- RRE Transformation Fails After Importing Pricing Objects

Troubleshooting ImportExportPricing Errors

When running the **ImportExportPricing** utility in a BRM cloud native environment, it may generate the following error:

```
oracle.communications.brm.pdc.server.transformation.Transformer commitChanges

SEVERE: Error while committing transaction

oracle.communications.brm.pdc.server.transformation.TransformationException:
Error in database operation for [committing connection]
```

To resolve the issue:

1. Verify the connectivity to the database for both the Cross Reference (XREF) schema and the BRM schema.
2. Redeploy PDC.

> **Note**
>
> For an XREF schema error, redeploy PDC. For a BRM schema error, redeploy BRM and PDC both.

Unable to Import a Large Number of XML Files

You may encounter issues when importing a large number of XML files into your PDC cloud native environment using the **ImportExportPricing** utility. This occurs due to insufficient memory allocation for the WebLogic Server. To resolve the issue:

Copyright © 2021, 2026, Oracle and/or its affiliates.

1. In your **override-values.yaml** file for **oc-cn-op-job-helm-chart**, update the **ocpdc.configEnv.USER_MEM_ARGS** key to a higher value. For example:

   ```
   USER_MEM_ARGS: "-Xms10g -Xmx12g -XX:CompileThreshold=8000″
   ```

2. Apply the changes by running the **helm upgrade** command for **oc-cn-op-job-helm-chart**:

   helm upgrade *OpJobReleaseName* **oc-cn-op-job-helm-chart --values** *OverrideValuesFile* **--namespace** *BrmNameSpace*

3. In your **override-values.yaml** file for **oc-cn-helm-chart**, update the **ocpdc.configEnv.transformation.MEM_ARGS** key to a higher value. For example:

   ```
   MEM_ARGS: "-Xms10g -Xmx12g -XX:CompileThreshold=8000"
   ```

4. Run the **helm upgrade** command for **oc-cn-helm-chart**:

   helm upgrade *BrmReleaseName* **oc-cn-helm-chart --values** *OverrideValuesFile* **--namespace** *BrmNameSpace*

5. Verify if the import issues have been resolved by rerunning the **ImportExportPricing** utility in your cloud native environment. See “Importing and Exporting Pricing and Setup Components” in *PDC Creating Product Offerings*.

If the issue persists, you can further increase the **Xmx** value based on your system's capacity.

SyncPDC Fails After Cloning BRM and PDC Databases

> **Note**
>
> This applies to:
>
> - PDC systems with real-time rating engine (RRE) and batch rating engine (BRE) only.
> - PDC systems upgrading from version 12.0 patch set 5 or earlier to later versions (12.0 patch set 6 and after).

When you start **SyncPDC** after cloning the BRM and PDC databases from one environment to another, you may encounter an error stating that the **/event/delayed/session/telco/gprs** event is not configured as a rating event in BRM:

```
SEVERE: Error while running SyncPDC!
oracle.communications.brm.pdc.migration.MigrationException: Unexpected error occurred in
migration operation
       at oracle.communications.brm.pdc.syncpdc.SyncPDCJob.execute(SyncPDCJob.java:234)
       at oracle.communications.brm.pdc.syncpdc.SyncPDC.run(SyncPDC.java:143)
       at oracle.communications.brm.pdc.syncpdc.SyncPDC.main(SyncPDC.java:299)
Caused by: oracle.communications.brm.pdc.migration.MigrationException: Error in
migration operation
...
Caused by: java.lang.Throwable: There are errors when parsing bcc EventAttributeSpecs.
Please fix the issues, then restart syncPDC.
The event, "/event/delayed/session/telco/gprs", was configured as a rating event in BRM
and used for creating a PDC EventAttributeSpec "EventDelayedSessionTelcoGprs", is not
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
configured as a rating event in BRM currently. It is not allowed to delete a rating
event from PDC. Please restore "/event/delayed/session/telco/gprs" as a rating event in
BRM.
```

This error occurs because the data in the **pin_event_map** file and **IFW_REF_MAP** database table are corrupted.

To fix this:

1. In the BRM database, ensure that **/event/delayed/session/telco/gprs** is set with the **Event_Type** set to **USAGE_POSTPAID** in the DD_Objects_T table. To do so, use SQL*Plus to run the following command on the BRM database:

   ```
   select obj_id0, name, event_type from dd_objects_t where name = '/event/delayed/
   session/telco/gprs;'
   ```

2. In the PDC database, ensure that **/event/delayed/session/telco/gprs** is set with the **Event_Type** set to **USAGE_POSTPAID** in the ABCATTRIBUTESPEC and BCEVENTATTRIBUTESPEC tables. To do so, use SQL*Plus to run the following command on the PDC database:

   ```
   SELECT name, eventtype, version FROM abcattributespec aas, bceventattributespec eas
   WHERE classname = '/event/delayed/session/telco/gprs' and aas.entityid =
   eas.entityid
   ORDER BY version DESC;
   ```

3. In *BRM_Integration_Pack_home***/apps/syncpdc/xml/extract/Event.xml** file, ensure **/event/delayed/session/telco/gprs** is set with **EVENT_TYPE** is equal to **USAGE_POSTPAID**.

4. In the *BRM_Integration_Pack_home***/apps/syncpdc/SyncPDCConfiguration.xml** file, ensure **skipBREMigration** is set to **false**:

   ```
   <skipBREMigration>false</skipBREMigration>
   ```

5. Ensure **/event/delayed/session/telco/gprs** is listed in *BRM_home***/sys/data/config/pin_event_map_telco_gprs** file.
   If not, do the following:

   a. Edit the **pin_event_map_telco_gprs** file and add the following:

      ```
      /service/telco/gprs : /event/session/telco/gprs : Real Time Telco GPRS Session
      : /event/delayed/session/telco/gprs : Delayed Telco GPRS Session
      ```

   b. Load the updated **pin_event_map_telco_gprs** file into the database using the **load_event_map** utility. See “load_event_map” in *BRM System Administrator's Guide*.

   c. Verify that **/event/delayed/session/telco/gprs** appears in the CONFIG_PERMITTED_EVENTS_T table in the database.

   d. Restart **SyncPDC**.

6. Ensure that **/event/delayed/session/telco/gprs** exists in the IFW_REF_MAP table. SyncPDC validates all events, including derived events, against the IFW_REF_MAP table. So, you must manually ensure that each derived event also has a valid mapping in IFW_REF_MAP.
   If not, add it by editing the **pin_event_map_telco_gprs** file and running the following SQL command:

   ```
   INSERT INTO "pre"."ifw_ref_map" (id, ref_obj, ref_param, entryby,
   entrydate, modified, modifdate, recver) VALUES ('CustomerData', '/service/
   telco', '/event/delayed/session/telco/gprs', '0', to_date('2023-11-02
   05:48:38', 'YYYY-MM-DD HH24:MI:SS'), '1', to_date('2023-11-02 05:48:38',
   'YYYY-MM-DD HH24:MI:SS'), '0');
   ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

> **Note**
>
> In BRM, each service in the IFW_REF_MAP table can only be mapped to one event.

7. Restart **SyncPDC**.

Publishing Pricing Components to ECE Fails

After you upgrade BRM from version 12.0 to 15.1 or later and set up ECE and BRM in separate cloud instances, PricingUpdater may not connect to the PDC WebLogic server. As a result, publishing pricing components from PDC to ECE can fail with the following error in the pricingupdater_log file:

```
 Unable to connect to Provider : null
.
.
.
Caused by: java.net.ConnectException: t3://IPaddress:8001: Bootstrap to IPaddress/
IPaddress:8001 failed. It is likely that the remote side declared peer gone on this JVM
.
.
.
Caused by: java.net.ConnectException: t3://IPaddress:8001 Bootstrap to IPaddress/
IPaddress:8001 failed. It is likely that the remote side declared peer gone on this JVM
```

To fix this, do the following:

1. Connect PDC to the WebLogic Server by making one of the following changes for PDC in the **override-values.yaml** file for the **oc-cn-op-job-helm-chart**:
   - Use a NodePort (t3 or t3s) that is configured in Kubernetes to allow PricingUpdater to connect to the PDC WebLogic server.

     ```
     ocpdc:
         isEnabled: true
         configEnv:
             t3ChannelPort: t3Port
             t3sChannelPort: t3sPort
          service:
             name: "pdc-service"
             type: "NodePort"
     ```

     where:
     - *t3Port* is the port for T3 traffic. Choose any available Kubernetes port. The node port range for Kubernetes is 30000 – 32767.
     - *t3sPort* is the port for T3 traffic over SSL. Choose a different available Kubernetes port.
   - If the PricingUpdater is running in a different Kubernetes namespace, use the ClusterIP port to connect to the PDC WebLogic server.

     ```
     ocpdc:
         isEnabled: true
         configEnv:
             t3ChannelPort: t3Port
     ```


Copyright © 2021, 2026, Oracle and/or its affiliates.

```
      t3sChannelPort: t3sPort
   service:
      name: " pdc-domain-adminserver"
      type: " ClusterIP"
```

2. Run the helm upgrade command to apply your changes for PDC:

   helm upgrade *OpJobReleaseName* **oc-cn-op-job-helm-chart --namespace** *BrmNameSpace* **-- values** *OverrideValuesFile*

3. After updating the connection method, access PDC in your browser using one of the following URLs to verify that you can reach it.
   - If you are not using a load balancer:

     `http://kubernetesHost:pdcPort/pdc`

     where *kubernetesHost* is the host name of the machine on which Kubernetes is deployed and *pdcPort* is the PDC service node port
   - If you are using a load balancer:

     `http://LoadBalancerHost:pdcNodePort/pdc`

     where *LoadBalancerHost* is the host name of the machine on which the load balancer is deployed and *pdcNodePort* is the number assigned to the PDC node port.
4. Enter the PDC connection URL in the ECE PricingUpdater configuration. To do this, update the **override-values.yaml** file for **oc-cn-ecehelm-chart** to set the **migration.pricingUpdater.connectionURL** key.

   ```
   migration:
     pricingUpdater:
       hostName: "pdc-domain-adminserver"
       port: "8001"
       userName: "weblogic"
       connectionURL: "t3://pdc-domain-adminserver:8001"
   ```

5. Run the helm upgrade command to apply your changes for ECE:

   helm upgrade *EceReleaseName* **oc-cn-ece-helm-chart --namespace** *EceNameSpace* **--values** *OverrideValuesFile*

RRE Transformation Fails After Importing Pricing Objects

After upgrading from PDC 15.0 to a later version, the RRE may fail after you import pricing files, and the change set moves to a Failed status. The following error appears in the RRE Transformation Engine log file:

```
SEVERE: Error in transformation
oracle.communications.brm.pdc.server.transformation.TransformationException:
Error in transformation operation
Caused by: java.lang.Throwable: Error adding transaction handler
```


Copyright © 2021, 2026, Oracle and/or its affiliates.

To fix this, do the following:

1. Redeploy PDC.
2. Validate the deployment to ensure all components are correctly configured.
3. Retry the pricing file import and confirm that the change set completes successfully.

Copyright © 2021, 2026, Oracle and/or its affiliates.

A

WebLogic-Based Application Metrics

This appendix lists the WebLogic-based application metrics 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 Runtime Metric Group
- Work Manager Runtime 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

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 servletName	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 servletName	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.
wls_servlet_execution_time_low	location app name servletName	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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table A-3 (Cont.) Servlets Metrics

Metric Name	Label	Metric Type	Description
wls_servlet_execution_time_total	location app name servletName	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 servletName	int	Displays the total number of times the servlet has been invoked since WebLogic Server started.
wls_servlet_pool_max_capacity	location app name servletName	int	Displays the maximum capacity of this servlet for single-thread model servlets.
wls_servlet_reload_total_count	location app name servletName	int	Displays the number of times the 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.
wls_jvm_process_cpu_load	name	time	Returns the amount of CPU time the Java virtual machine runs in nanoseconds.
wls_jvm_uptime	name	long	Returns the number of milliseconds the virtual machine has been running.

Execute Queue Runtime Metric Group

Use the execute queue runtime metric group to return information about the queue. Table A-5 describes the metrics in this group.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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 Runtime Metric Group

Use the work manager runtime 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 Runtime 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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.
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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table A-8 (Cont.) JDBC Service Runtime Metrics

Metric Name	Label	Metric Type	Description
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.
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 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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table A-9 (Cont.) JTA Runtime Metrics

Metric Name	Label	Metric Type	Description
wls_jta_transaction_heuristics_total_count	name	long	Returns the number of completed transactions 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.
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, direct-write, and cache-flush policies.
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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table A-11 (Cont.) Persistent Store Runtime MBean Metrics

Metric Name	Label	Metric Type	Description
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.

Copyright © 2021, 2026, Oracle and/or its affiliates.

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_virtual_time**
- **syncPDC**

Copyright © 2021, 2026, Oracle and/or its affiliates.

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**
apps/storable_class_to_xml	**storableclasstoxml**
apps/load_config	**load_config**

Copyright © 2021, 2026, Oracle and/or its affiliates.

Table C-1 (Cont.) Supported Applications

Directory	Application
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
apps/multi_db	pin_multidb.pl

Copyright © 2021, 2026, Oracle and/or its affiliates.

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**

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **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**

Copyright © 2021, 2026, Oracle and/or its affiliates.

- **load_transition_type**
- **load_usage_map**
- **pin_bus_params**
- **pin_deploy**
- **pin_load_invoice_events**
- **pin_uei_deploy**
- **testnap**

Copyright © 2021, 2026, Oracle and/or its affiliates.