Oracle® Communications Billing and Revenue Management
Cloud Native Installation and Administration Guide

Release 12.0
F21893-01
September 2019
## Contents

### Preface

| Audience | vi |
| Documentation Accessibility | vi |

### 1 Overview of the BRM Cloud Native Deployment

| About the BRM Cloud Native Deployment | 1-1 |
| BRM Cloud Native Deployment Architecture | 1-1 |
| Images and Containers | 1-2 |
| Images and Containers with Non-WebLogic Server Pattern | 1-2 |
| Images and Containers with a WebLogic Server Pattern | 1-3 |

### 2 Planning Your Installation

| Overview | 2-1 |
| Software Compatibility | 2-2 |
| Environment Setup | 2-2 |
| Moving from On-Premise BRM to BRM Cloud Native Deployment | 2-4 |
| Downloading the BRM Cloud Native Deployment Package | 2-4 |
| Merging Split TAR Files | 2-6 |
| Loading BRM Component Images | 2-6 |

### 3 Installing the BRM Cloud Native Deployment Package

| About Deploying into Kubernetes | 3-1 |
| Extracting the Helm Charts | 3-2 |
| Configuring the BRM Cloud Native Deployment Package | 3-2 |
| Configuring BRM Server | 3-2 |
| Deploying BRM with a New Schema | 3-2 |
| Deploying BRM with an Existing Schema | 3-4 |
| Configuring oc-cn-helm-chart | 3-6 |
| Creating a CronJob | 3-8 |
| Configuring Pricing Design Center | 3-8 |
Adding PDC Keys 3-9
Cleaning Up Sample BRM Data 3-11
Enabling SSL in Your PDC Domain 3-13
Configuring Billing Care 3-13
  Adding Billing Care Keys 3-13
  Enabling SSL in Your Billing Care Domain 3-15
  About Authorization Policy for Billing Care 3-15
  About Billing Care Logs 3-15
  About Billing Care Volume Mounts 3-16
Configuring Business Operations Center 3-16
  Adding Business Operations Center Keys 3-16
  Enabling SSL in Your Business Operations Center Domain 3-18
  Configuring Authorization Policy for Business Operations Center 3-19
  About Business Operations Center Logs 3-19
  About Business Operations Center Volume Mounts 3-19
Installing BRM Cloud Native Services 3-20

4 Customizing BRM Cloud Native Services

  Customizing BRM Server 4-1
  Customizing Billing Care 4-2

5 Maintaining Your BRM Cloud Native Deployment

  Managing a Helm Release 5-1
  About Application Log Files 5-2

6 Uninstalling Your BRM Cloud Native Deployment

  Uninstalling Your BRM Cloud Native Deployment 6-1

7 Building Your Own Images

  Building BRM Server Images 7-1
    Building Your BRM Server Base Image 7-2
    Building Images of BRM Server Components 7-3
    Layering BRM Server Images 7-4
  Building PDC Images 7-5
    Building the Base PDC Image 7-5
    Building the Oracle PDC Image 7-5
  Building Billing Care Images 7-6
    Building the Oracle Linux Image 7-6
8 Deploying into Oracle Cloud Infrastructure

Deploying into Oracle Cloud Infrastructure 8-1
Preface

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

Audience

This document is intended for DevOps administrators and those involved in installing and maintaining an Oracle Communications Billing and Revenue Management (BRM) Cloud Native Deployment.

Documentation Accessibility

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

Access to Oracle Support

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

This chapter provides an overview of the Oracle Communications Billing and Revenue Management (BRM) cloud native deployment.

Topics in this chapter

• About the BRM Cloud Native Deployment
• BRM Cloud Native Deployment Architecture

About the BRM Cloud Native Deployment

Oracle Communications Billing and Revenue Management, Oracle Communications Pricing Design Center (PDC), Oracle Communications Billing Care, and Oracle Communications Business Operations Center now support their deployment on a cloud native environment. This allows you to harness the benefits of cloud with the services of BRM.

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

BRM Cloud Native Deployment Architecture

Figure 1-1 shows the Pods and other components in a typical BRM cloud native deployment.
In this figure:

- Billing Care, Business Operations Center (BOC), and Pricing Design Center (PDC) are client applications. They connect to the CM, which represents the business logic layer of BRM, by using the Portal Communications Protocol (PCP).
- The CM communicates with other Pods, which represent the data management layer of BRM, by using the PCP protocol.
- All PCP protocol communication is encrypted using TLS.
- The data managers (DMs) interact with other downstream products that run the business logic.
  - The downstream products can be containers or an on-premise system.
- Rating files for the batch pipeline are fed in through a Kubernetes PersistentVolumeClaim (PVC). The batch pipeline output is also available in a PVC for consumption by the Rated Event Loader Pod.

**Images and Containers**

Each BRM cloud native deployment image has been designed and hardened to serve only the purpose it's supposed to. BRM cloud native deployment images are built by stacking multiple layers, extending an operating system image with a dependent library image, and then with an image packing the application.

**Images and Containers with Non-WebLogic Server Pattern**

If your BRM images don't need WebLogic Server, you would layer the BRM base images as shown in Figure 1-2.
WebLogic Server is a primary requirement for the Billing Care, PDC, and Business Operations Center cloud native deployments. Thus, all of these images share a similar image stack.

Figure 1-3 shows the stack for a Billing Care image.
In this figure:

- Oracle Linux is the operating system base image.
- The Server JRE image provides Java, which is a prerequisite for the Fusion Middleware Infrastructure image.
- The Billing Care image extends the Fusion Middleware Infrastructure image, which provides WebLogic Server and JRF for OPSS for authorized access to the application.

The Business Operations Center and PDC images have an additional dependency on BRM applications:

- The Business Operations Center image references the brm-apps image for BRM applications.
- The PDC image references both the brm-apps and realtime-pipeline images for copying BRM applications and the `LoadIfwConfig` utility from Pipeline Manager.

Figure 1-4 shows the process for starting a container for a Billing Care image based on a WebLogic Server pattern.
If this container is stopped and restarted, only the WebLogic Server is started because the WebLogic domain and Billing Care are already installed and configured in the container file system.

In Kubernetes, this container is deployed as part of a Pod running in a Deployment controller set. This means that if the Pod crashes, the Deployment controller set starts a new Pod with a new container, thereby installing a new domain and Billing Care.
Planning Your Installation

This chapter describes how to plan your installation of the Oracle Communications Billing and Revenue Management (BRM) cloud native deployment package.

Topics in this chapter

• Overview
• Software Compatibility
• Environment Setup
• Moving from On-Premise BRM to BRM Cloud Native Deployment
• Downloading the BRM Cloud Native Deployment Package
• Merging Split TAR Files
• Loading BRM Component Images

Overview

The BRM cloud native deployment package includes ready-to-use images and Helm charts to help you orchestrate containers in Kubernetes. The package also includes sample Dockerfiles and scripts if you want to build the images in your environment.

You can use the Docker images and Helm chart to help you deploy and manage Pods of BRM product services in Kubernetes. Communication between Pods of services of BRM products are preconfigured in the Helm charts.

To improve security, the BRM services are not exposed outside of the cluster. Only Pricing Design Center, Billing Care, and Business Operations Center are exposed.

Table 2-1 lists the Pods and images for BRM whose containers are created and services are exposed through them. For the image name, replace 12.0.0.x.0 with the patch set version number, such as 12.0.0.2.0. For example, for the 12.0.0.2.0 patch set, the dm-oracle Pod image name would be dm_oracle:12.0.0.2.0.

Table 2-1   BRM Pods and Images

<table>
<thead>
<tr>
<th>Pod</th>
<th>Replica Type</th>
<th>Image Name</th>
<th>Image Size</th>
<th>Container Port</th>
<th>Service Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>cm</td>
<td>Multiple</td>
<td>cm:12.0.0.x.0</td>
<td>2.15 GB</td>
<td>11960</td>
<td>ClusterIP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>eai_js:12.0.0.x.0</td>
<td>738 MB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dm-oracle</td>
<td>Multiple</td>
<td>dm_oracle:12.0.0.0</td>
<td>1.99 GB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>init-db</td>
<td>Single</td>
<td>init_db:12.0.0.x.0</td>
<td>2.12 GB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>dm-aq</td>
<td>Single</td>
<td>dm_aq:12.0.0.x.0</td>
<td>1.97 GB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>dm-ifw-sync</td>
<td>Single</td>
<td>dm_ifw_sync12.0.0.x.0</td>
<td>1.97 GB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>dm-vertex</td>
<td>Single</td>
<td>dm_vertex:12.0.0.x.0</td>
<td>1.98 GB</td>
<td></td>
<td>ClusterIP</td>
</tr>
</tbody>
</table>
Table 2-1 (Cont.) BRM Pods and Images

<table>
<thead>
<tr>
<th>Pod</th>
<th>Replica Type</th>
<th>Image Name</th>
<th>Image Size</th>
<th>Container Port</th>
<th>Service Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>dm-eai</td>
<td>Multiple</td>
<td>dm_eai:12.0.0.x.0</td>
<td>760 MB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>dm-invoice</td>
<td>Multiple</td>
<td>dm_invoice:12.0.0.x.0</td>
<td>1.99 GB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>dm-ldap</td>
<td>Multiple</td>
<td>dm_ldap:12.0.0.x.0</td>
<td>1.77 GB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>dm-prov-telco</td>
<td>Multiple</td>
<td>dm_prov_telco:12.0.0.x.0</td>
<td>841 MB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>realtime-pipe</td>
<td>Multiple</td>
<td>realtimepipe:12.0.0.x.0</td>
<td>2.09 GB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>batch-wireless-pipe</td>
<td>Single</td>
<td>batch_pipeline:12.0.0.x.0</td>
<td>2.23 GB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>rel-daemon</td>
<td>Multiple</td>
<td>rel_daemon:12.0.0.x.0</td>
<td>2.05 GB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>batch-controller</td>
<td>Multiple</td>
<td>batch_controller:12.0.0.x.0</td>
<td>880 MB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>formatter</td>
<td>Single</td>
<td>formatter:12.0.0.x.0</td>
<td>1.94 GB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>fusa_simulator</td>
<td>Single</td>
<td>answer:12.0.0.x.0</td>
<td>1.74 GB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>dm-fusa</td>
<td>Single</td>
<td>dm_fusa:12.0.0.x.0</td>
<td>1.74 GB</td>
<td></td>
<td>ClusterIP</td>
</tr>
<tr>
<td>N/A</td>
<td>N/A</td>
<td>brm_apps:12.0.0.x.0</td>
<td>3.21 GB</td>
<td></td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 2-2 lists the Pods and images for Pricing Design Center (PDC), Billing Care, and Business Operations Center. For the image name, replace 12.0.0.x.0 with the patch set version number, such as 12.0.0.2.0. For example, for the 12.0.0.2.0 patch set, the pdc Pod image name would be oracle/pdcapp:12.0.0.2.0.

Table 2-2 Client Pods and Images

<table>
<thead>
<tr>
<th>Pod</th>
<th>Replica Type</th>
<th>Image</th>
<th>Image Size</th>
<th>Container Port</th>
<th>Service Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdc</td>
<td>Single</td>
<td>oracle/pdcapp:12.0.0.x.0</td>
<td>11.3 GB</td>
<td>7011 (http) 7012 (https)</td>
<td>NodePort</td>
</tr>
<tr>
<td>billingcare</td>
<td>Single</td>
<td>oracle/billingcare:12.0.0.x.0</td>
<td>2.75 GB</td>
<td>7012 (https)</td>
<td>NodePort</td>
</tr>
<tr>
<td>boc</td>
<td>Single</td>
<td>oracle/boc:12.0.0.x.0</td>
<td>3.81 GB</td>
<td>7012 (https)</td>
<td>NodePort</td>
</tr>
</tbody>
</table>

Software Compatibility

For the software that is compatible with the BRM cloud native deployment package, see "Cloud Native Deployment Software Compatibility".

For a full list of software that is compatible with BRM, see "BRM Software Compatibility".

Environment Setup

The BRM cloud native deployment package works with these technologies:
• **Docker:** The Docker platform is used to containerize BRM products. Docker is needed if you want to build images by writing your own Dockerfiles using the sample Dockerfiles from the BRM cloud native deployment package. You use the Docker container runtime to create and run containers of these images in Kubernetes. You also host a repository for storing the images, which can later be accessed by Kubernetes for creating Pods.

For more information about Docker, see [Docker Reference](https://docs.docker.com/reference).

• **Kubernetes:** Typically, a production-grade Kubernetes cluster is formed by multiple systems distributing the resources. Being a container platform, it provides a container-centric management environment. The cloud native deployment package contains YAML descriptors for creating Kubernetes objects and for allowing it to orchestrate containers of images from BRM products. Set up the Kubernetes cluster and secure access to the cluster and its objects with the help of service accounts and proper authentication and authorization modules. Also, choose volumes, a networking model, and logging services to be used in your cluster.

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

---

**Note:**

Ensure that your cluster is secured according to standard DevOps practices.

---

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

**Note:**

You can choose an external incubator to create persistent volumes, but ensure that it supports the ReadWriteMany access mode.

---

– **Networking:** Kubernetes assumes that Pods can communicate with other Pods, regardless of which host they land on. Every Pod gets its own IP address, so we don’t need to explicitly create a link between Pods. We almost never need to deal with mapping container ports to host ports. While Kubernetes itself doesn’t offer a solution to support its assumption, several implementations are available that meet the fundamental requirements of Kubernetes’ networking model. Choose the networking element depending on the cluster requirement.

– **Logging:** The BRM cloud native deployment centralizes all log files into a single location within each Pod. This location is a volume mount of type...
**emptyDir.** You must choose from various logging service providers to add as a side-car application to each Pod whose logs need to be persisted.

- **Helm:** Helm is the package manager for Kubernetes, and chart is a Helm package. Charts, which are packaged as part of the BRM cloud native deployment package, help create Kubernetes objects like ConfigMap, Secrets, controller sets, and Pods with a single command. This feature of Helm makes it easy to handle all BRM configurations, deployments, and services.

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

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

- **Oracle Unified Directory:** Both the Billing Care and Business Operations Center cloud native deployments support Oracle Unified Directory (OUD) as an LDAP solution for authenticating users. You must have a standalone or containerized instance of OUD running in the same network as the Kubernetes cluster.

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

### Moving from On-Premise BRM to BRM Cloud Native Deployment

If you're an existing BRM customer on BRM 7.5.x or BRM 12.x, perform these high-level tasks to move to the BRM cloud native deployment:

- Upgrade the BRM and PDC database schemas to 12.0.0.2.0 or later. For more information, see *BRM Upgrade Guide*.

- If you customized BRM or Billing Care, layer your customizations on top of the Docker images provided with this release before deploying the images. For guidelines about customization, see "Customizing BRM Cloud Native Services".

### Downloading the BRM Cloud Native Deployment Package

The BRM cloud native deployment package contains ready-to-use Docker images, a Helm package for deployment in Kubernetes, as well as sample Dockerfiles and scripts that you can use as a reference for building your own images. If you want to build your own images, you must also download these installers:

- Oracle Communications Billing and Revenue Management
- Oracle Communications Billing Care
- Oracle Communications Business Operations Center
- Oracle Communications Pricing Design Center

*Table 2-3* lists all of the component packages that are included with the BRM cloud native deployment package.
## Table 2-3  BRM Component Packages

<table>
<thead>
<tr>
<th>Component Package Name</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Communications Cloud Native Connection Manager</td>
<td>oc-cn-brm-cm-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Enterprise Application Integration Java Server</td>
<td>oc-cn-brm-eai-js-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Oracle Database Manager</td>
<td>oc-cn-brm-dm-oracle-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Database Initializer</td>
<td>oc-cn-brm-init-db-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Advanced Queuing Data Manager</td>
<td>oc-cn-brm-dm-aq-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Synchronization Queue Data Manager</td>
<td>oc-cn-brm-dm-ifw-sync-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Vertex Data Manager</td>
<td>oc-cn-brm-dm-vertex-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Enterprise Application Integration Data Manager</td>
<td>oc-cn-brm-dm-eai-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Invoice Data Manager</td>
<td>oc-cn-brm-dm-invoice-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native LDAP Data Manager</td>
<td>oc-cn-brm-dm-ldap-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Provisioning Data Manager</td>
<td>oc-cn-brm-dm-provtelco-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Real-Time Pipeline</td>
<td>oc-cn-brm-realtimepipeline-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Batch Pipeline</td>
<td>oc-cn-brm-batchpipeline-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Rated Event Loader</td>
<td>oc-cn-brm-rel-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Batch Controller</td>
<td>oc-cn-brm-batchcontroller-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Invoice Formatter</td>
<td>oc-cn-brm-invoiceformatter-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Fusa Simulator</td>
<td>oc-cn-brm-fusa-simulator-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Fusa Data Manager</td>
<td>oc-cn-brm-dm-fusa-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native BRM Applications</td>
<td>oc-cn-brm-apps-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Pricing Design Center</td>
<td>oc-cn-pdc-12.0.0.x.0_part1of3.tar</td>
</tr>
<tr>
<td></td>
<td>oc-cn-pdc-12.0.0.x.0_part2of3.tar</td>
</tr>
<tr>
<td></td>
<td>oc-cn-pdc-12.0.0.x.0_part3of3.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Billing Care</td>
<td>oc-cn-billingcare-12.0.0.x.0.tar</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Business Operations Center</td>
<td>oc-cn-boc-12.0.0.x.0.tar</td>
</tr>
</tbody>
</table>
Table 2-3 (Cont.) BRM Component Packages

<table>
<thead>
<tr>
<th>Component Package Name</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Communications Cloud Native Docker Build Files</td>
<td>oc-cn-docker-files-12.0.0.x.0.tgz</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native DB Initializer Helm Chart</td>
<td>oc-cn-init-db-helm-chart-12.0.0.x.0.tgz</td>
</tr>
<tr>
<td>Oracle Communications Cloud Native Helm Chart</td>
<td>oc-cn-helm-chart-12.0.0.x.0.tgz</td>
</tr>
</tbody>
</table>

Merging Split TAR Files

To improve download performance, BRM images that have TAR files larger than 4 GB have been split into multiple parts. You can merge the split TAR files before tagging and pushing them to the registry by entering this command:

```
cat oc-cn-Component-12.0.0.x.0_part* > oc-cn-Component-12.0.0.x.0.tar
```

where Component is the short name of the BRM component, such as pdc for Pricing Design Center (PDC).

For example, to merge the PDC TAR files into a single TAR file, you’d enter this command:

```
cat oc-cn-pdc-12.0.0.x.0_part* > oc-cn-pdc-12.0.0.x.0.tar
```

Loading BRM Component Images

Images shipped with the BRM cloud native deployment package are in the form of TAR files. After downloading the TAR files, load them as images into the Docker system.

For example, to load the Oracle Communications Cloud Native Connection Manager image into the Docker system, you would do this:

1. Download the `oc-cn-brm-cm-12.0.0.x.0.tar` file to the system where Docker is installed.
2. Enter this command:

   ```
docker load --input oc-cn-brm-cm-12.0.0.x.0.tar
```

3. Verify that the image is loaded correctly by entering this command:

   ```
docker images cm:12.0.0.x.0
```

   The image details should be listed in one row.

If you use an internal registry to access images from different Kubernetes nodes, push the images from the local system to the registry server. For example, if the registry is identified by `RepoHost:RepoPort`, you’d push the CM image to the registry like this:
1. Tag the image with the registry server by entering this command:

   `docker tag cm:12.0.0.x.0 RepoHost:RepoPort/cm:12.0.0.x.0`

2. Push the image to the registry server by entering this command:

   `docker push RepoHost:RepoPort/cm:12.0.0.x.0`
Installing the BRM Cloud Native Deployment Package

This chapter describes how to install the BRM cloud native deployment package on a cloud native environment.

Topics in this chapter

• About Deploying into Kubernetes
• Extracting the Helm Charts
• Configuring the BRM Cloud Native Deployment Package
• Installing BRM Cloud Native Services

About Deploying into Kubernetes

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

The BRM cloud native deployment package includes these Helm charts:

• oc-cn-helm-chart deploys BRM cloud native services.
• oc-cn-init-db-helm-chart deploys the init-db container, which initializes the BRM database schema and loads the initial default data (seed data). It creates database objects such as tables, views, indexes, partitions, and so on in an empty schema.

Note:

If you want to use an existing fully initialized BRM database, don't deploy oc-cn-init-db-helm-chart.

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

Before you install BRM cloud native services, extract the Helm charts from `oc-cn-helm-chart-12.0.0.x.0.tgz` and `oc-cn-init-db-helm-chart-12.0.0.x.0.tgz`. To do so, enter these commands:

```
tar xvzf oc-cn-helm-chart-12.0.0.x.0.tgz
```

```
tar xvzf oc-cn-init-db-helm-chart-12.0.0.x.0.tgz
```

Configuring the BRM Cloud Native Deployment Package

The `oc-cn-helm-chart` Helm chart configures and deploys all of your product services. YAML descriptors in the `oc-cn-helm-chart/templates` directory use the `oc-cn-helm-chart/values.yaml` file for most of the values. The `values.yaml` file itself includes comments that describe each key. You can override the values by creating an `override-values.yaml` file.

Oracle recommends that you read the `values.yaml` file at least once to become familiar with all of the options available.

Configuring BRM Server

Configuring your BRM Server involves these high-level steps:

1. Deploying the BRM database schema:
   - Deploying BRM with a New Schema
   - Deploying BRM with an Existing Schema

2. Configuring the Helm chart for deploying BRM cloud native services. See "Configuring oc-cn-helm-chart".

3. Creating a CronJon for running BRM applications. See "Creating a CronJob".

Deploying BRM with a New Schema

To deploy BRM with a new schema, deploy the Helm chart for init-db and then deploy the Helm chart for BRM.

![Note:]

Pre-create the schema user with the required grants for both BRM Server and pipeline.

Configuring oc-cn-init-db-helm-chart

For the init-db container, create the BRM schema user and pipeline schema user with the required grants. See "Setting Your Database for BRM".
Table 3-1 lists the keys that directly impact the BRM database schema and pipeline database schema. Add these keys to your override-values.yaml file with the same path hierarchy.

⚠️ Caution:

Some keys hold sensitive data. They must be handled carefully with controlled access to the file containing its values. Encode all of these values in base64. See "Secrets" in Kubernetes Concepts.

Table 3-1  oc-cn-init-db-helm-chart Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocbrm.imagePullPolicy</td>
<td>The default value is IfNotPresent, which specifies to not pull the image if it's already present. Applicable values are IfNotPresent and Always.</td>
</tr>
<tr>
<td>ocbrm.logDir</td>
<td>The path to the dm-oracle log files.</td>
</tr>
<tr>
<td>ocbrm.LOG_LEVEL</td>
<td>The dm-oracle log level.</td>
</tr>
<tr>
<td>ocbrm.brm_root_pass</td>
<td>The BRM root password. You must generate a secret. Add this key to your override-values.yaml file with the same path hierarchy.</td>
</tr>
<tr>
<td>ocbrm.wallet.client</td>
<td>The password for the client wallet. You must generate a secret. Add this key to your override-values.yaml file with the same path hierarchy.</td>
</tr>
<tr>
<td>ocbrm.wallet.server</td>
<td>The password for the server wallet. You must generate a secret. Add this key to your override-values.yaml file with the same path hierarchy.</td>
</tr>
<tr>
<td>ocbrm.wallet.root</td>
<td>The password for the root wallet. You must generate a secret. Add this key to your override-values.yaml file with the same path hierarchy.</td>
</tr>
<tr>
<td>ocbrm.db.host</td>
<td>The IP address or the host name of the machine on which the BRM database is configured.</td>
</tr>
<tr>
<td>ocbrm.db.port</td>
<td>The port on which the BRM database is configured.</td>
</tr>
<tr>
<td>ocbrm.db.service</td>
<td>The BRM database service name.</td>
</tr>
<tr>
<td>ocbrm.db.alias</td>
<td>The name of the BRM database alias.</td>
</tr>
<tr>
<td>ocbrm.db.schemaunder</td>
<td>The BRM schema user name, which should be pre-created with all the required grants.</td>
</tr>
<tr>
<td>ocbrm.db.schemapass</td>
<td>The BRM schema password. You must generate a secret. Add this key to your override-values.yaml file with the same path hierarchy.</td>
</tr>
<tr>
<td>ocbrm.db.schematablespace</td>
<td>The name of the tablespace for the BRM schema. This field is case-sensitive.</td>
</tr>
<tr>
<td>ocbrm.db.indextablespace</td>
<td>The name of the Index tablespace for the BRM schema. This field is case-sensitive.</td>
</tr>
<tr>
<td>ocbrm.db.pipeline-schemauser</td>
<td>The BRM pipeline schema username, which should be pre-created with all the required grants.</td>
</tr>
</tbody>
</table>
Table 3-1  (Cont.) oc-cn-init-db-helm-chart Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocbrm.db.pipelineschemapass</td>
<td>The BRM pipeline schema password. You must generate a secret. Add this key to your override-values.yaml file with the same path hierarchy.</td>
</tr>
<tr>
<td>ocbrm.db.pipelineschematablesp</td>
<td>The name of the tablespace for the BRM pipeline schema. This field is case-sensitive.</td>
</tr>
<tr>
<td>ocbrm.db.pipelineindextablespace</td>
<td>The name of the Index tablespace for the BRM pipeline schema. This field is case-sensitive.</td>
</tr>
<tr>
<td>ocbrm.db.pipelinealias</td>
<td>The name of the BRM pipeline database alias.</td>
</tr>
</tbody>
</table>

To commit seed data in the database, you can modify the init-db container before deploying the Helm chart. The configmap_create_obj_2.yaml file needs to be modified to add the corresponding PCM_OP_CREATE_OBJ input flist.

For example:

```
<PCM_OP $PIN_OPNAME=$PIN_CONF_INIT_OPNAME; $PIN_OPFLAGS=0>
  0 PIN_FLD_POID POID [0] 0.0.0.1 /config/recharge_card_type 0 0
  0 PIN_FLD_NAME STR [0] "--"
  0 PIN_FLD_PROGRAM_NAME STR [0] "load_pin_recharge_card_type"
  0 PIN_FLD_HOSTNAME STR [0] "--"
  0 PIN_FLD_VERSION STR [0] "1"
  0 PIN_FLD_ACCOUNT_OBJ POID [0] 0.0.0.1 /account 1 0
</PCM_OP>
```

Deploying BRM with an Existing Schema

To deploy BRM with an existing schema and with default Oracle images:

1. If the hostPath directory doesn't exist, create the directory and give it read and write permissions.
   For a multi-node cluster, create the same hostPath directory in all nodes.
2. Copy the $PIN_HOME/wallet/client directory from the staging location to hostPath.
   For a multi-node cluster, copy the client wallet to hostPath in all nodes.
3. Add this to the oc-cn-helm-chart/templates/dm_oracle.yaml file under spec.template.spec.containers.env:

   ```
   - name: BRM_WALLET
     value: /oms/client
   ```
4. Add this to the oc-cn-helm-chart/templates/dm_oracle.yaml file under spec.template.spec.containers:

   ```
   command: ["/bin/sh"]
   args: ["-c", "sleep 10; /oms/entrypoint.sh dm_oracle"]
   ```
5. Add this to the `oc-cn-helm-chart/templates/dm_invoice.yaml` file under `spec.template.spec.containers.env`:

   ```yaml
   - name: BRM_WALLET
     value: /oms/client
   ```

6. Add this to the `oc-cn-helm-chart/templates/dm_invoice.yaml` file under `spec.template.spec.containers`:

   ```yaml
   command: ["/bin/sh"]
   args: ["-c", "sleep 10; /oms/entrypoint.sh dm_invoice"]
   ```

7. Add this to both the `oc-cn-helm-chart/templates/dm_oracle.yaml` file and the `oc-cn-helm-chart/templates/dm_invoice.yaml` file:

   ```yaml
   volumeMounts:
   - name: client-wallet
     mountPath: /oms/client
   volumes:
   - name: client-wallet
     persistentVolumeClaim:
       claimName: client-wallet-pvc
   ```

8. Create this Kubernetes PersistentVolumeClaim (PVC) and place it in `oc-cn-helm-chart/templates`:

   ```yaml
   kind: PersistentVolumeClaim
   apiVersion: v1
   metadata:
     name: client-wallet-pvc
     namespace: {{ .Release.Namespace }}
     labels:
       application: {{ .Chart.Name }}
   spec:
     storageClassName: {{ .Values.ocbrm.storage_class.name }}
     accessModes:
     - ReadWriteMany
     resources:
       requests:
         storage: 50Mi
     volumeName: client-wallet-pv
   ```

9. Create this Kubernetes PersistentVolume (PV) and place it in `oc-cn-helm-chart/templates`. Ensure that `path` is set to the appropriate directory.

   ```yaml
   kind: PersistentVolume
   apiVersion: v1
   metadata:
     name: client-wallet-pv
     namespace: {{ .Release.Namespace }}
     labels:
       application: {{ .Chart.Name }}
   spec:
     storageClassName: {{ .Values.ocbrm.storage_class.name }}
     capacity:
   ```
Alternatively, you could deploy BRM with an existing schema by doing this:

1. Create Docker images for each of the BRM Server components in the installed BRM staging area (using the same staging area that initialized the database). When doing so, ensure that you:
   - Set the `ocbrm.init_database` flag to `false`.
   - Configure the queue name in the `configmap_env_dm_ifw_sync.yaml` file to match the queue name in the existing database.

2. Update `oc-cn-helm-charts/values.yaml` with the existing schema credentials.


### Configuring oc-cn-helm-chart

Table 3-2 lists the PVCs used by the BRM server.

<table>
<thead>
<tr>
<th>PVC Name</th>
<th>Pods</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtual-time</td>
<td>All Pods</td>
</tr>
<tr>
<td>oms-uel-archive</td>
<td>batch_controller</td>
</tr>
<tr>
<td>oms-uel-reject</td>
<td>batch_controller</td>
</tr>
<tr>
<td>oms-uel-input</td>
<td>batch_controller</td>
</tr>
<tr>
<td>fusa-temp</td>
<td>dm_fusa</td>
</tr>
<tr>
<td>ctqdir</td>
<td>dm_vertex</td>
</tr>
<tr>
<td>common-semaphore</td>
<td>batch_pipeline</td>
</tr>
<tr>
<td></td>
<td>realtime_pipeline</td>
</tr>
<tr>
<td>data</td>
<td>batch_pipeline</td>
</tr>
<tr>
<td>outputcdr</td>
<td>batch_pipeline</td>
</tr>
<tr>
<td></td>
<td>rel_daemon</td>
</tr>
<tr>
<td>outputrejct</td>
<td>batch_pipeline</td>
</tr>
<tr>
<td></td>
<td>rel_daemon</td>
</tr>
<tr>
<td>oms-rel-input</td>
<td>rel_daemon</td>
</tr>
<tr>
<td>oms-rel-reject</td>
<td>rel_daemon</td>
</tr>
<tr>
<td>oms-rel-archive</td>
<td>rel_daemon</td>
</tr>
</tbody>
</table>

Table 3-3 lists the keys that directly impact BRM Server Pods. Add these keys to your `override-values.yaml` file with the same path hierarchy.
Table 3-3  BRM Server Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ocbrm.brm_crypt_key</td>
<td>Fetch this from the database (table CRYPTKEY_T) after the schema is initialized. You must generate a secret.</td>
</tr>
<tr>
<td>ocbrm.ENABLE_SSL</td>
<td>For SSL-enabled deployment required in pin.conf.</td>
</tr>
<tr>
<td>ocbrm.isSSLEnabled</td>
<td>For SSL-enabled deployment required in Infranet.properties.</td>
</tr>
<tr>
<td>ocbrm.cm.load_localized</td>
<td>Set this to 0 if oc-cn-init-db-helm-chart isn't deployed and you're using an existing database.</td>
</tr>
<tr>
<td>ocbrm.virtual_time.enabled</td>
<td>Set this to true to enable pin_virtual_time.</td>
</tr>
<tr>
<td>ocbrm.wallet.*</td>
<td>Set this to the same values in the ocbrm.wallet.* keys from oc-cn-init-db-helm-chart if deployed. If you built the Docker images, update the</td>
</tr>
<tr>
<td></td>
<td>existing wallet passwords.</td>
</tr>
<tr>
<td>ocbrm.db*</td>
<td>Set this to the same values as the ocbrm.db.* keys from oc-cn-init-db-helm-chart.</td>
</tr>
<tr>
<td>ocbrm.storage_class.provisioner</td>
<td>Set this to the NFS-based provider name.</td>
</tr>
<tr>
<td>ocbrm.storage_class.mountOptions</td>
<td>Set this to the NFS-based provider version.</td>
</tr>
</tbody>
</table>

By default, all optional managers are enabled. If you don’t want to use an optional manager, remove its corresponding YAML files from the templates directory and comment out its related CM pin.conf file entries. For example, to disable dm-fusa, you’d remove the dm_fusa_pvc.yaml, dm_fusa.yaml, configmap_env_dm_fusa.yaml, and configmap_pin_conf_fusa_simulator.yaml files from the templates directory. You’d also comment out the entry for dm-fusa from the CM pin.conf file.

The fusa simulator is added for testing purposes only. In a production environment, dm-fusa will be connected to a Paymentech server rather than a fusa simulator. Thus, in a production environment, remove the fusa simulator YAML files (configmap_pin_conf_fusa_simulator.yaml, config_env_fusa_simulator.yaml, and fusa_simulator.yaml).

Deploy the oc-cn-init-db-helm-chart and oc-cn-helm-chart charts in different namespaces.

Before you deploy BRM Server, do this:

- Deploy the nfs-provisioner image and Pod.
- Update the oc-cn-helm-chart/values.yaml file with the provisioner name and version.
- If virtual_time is enabled and init-db is used, add these commands to the dm_oracle.yaml file under spec.template.spec.containers:

```sh
command: ["/bin/sh"]
args: ["-c", "sleep 120;/oms/entrypoint.sh dm_oracle"]
```
To load any business parameters or load utilities, you can use CronJob templates or use the restart_cm script. See "Creating a CronJob" or "Layering BRM Server Images".

Creating a CronJob

You can create a CronJob that automatically runs BRM applications for you.

Note:

MTA applications that aren't supported in the Business Operations Center UI can be run by using the CronJob.

To create a CronJob:

1. Build a Docker image named "kubectl:12.0.0.x.0" for the CronJob:

   ```bash
docker build -t kubectl:12.0.0.x.0 -f DockerFileLocation/Dockerfile .
   ```

2. Provide role-based access control to the CronJob so it can run kubectl commands:

   ```bash
   kubectl apply -f _cronjob_service_account.yaml
   kubectl apply -f _cronjob_rbac.yaml
   ```

3. Deploy the CronJob by using the templates/_cronjob.yaml file:

   a. Open the oc-cn-helm-chart/values.yaml file in a text editor.

   b. Under ocbrm.cronjob.deployment, update the app_name, schedule, app_dir, app_cmd, and app_cmd_args parameters.

      For example, to run the pin_bill_day application, the cronjob section of the values.yaml file would include these parameter values:

      ```yaml
      cronjob:
        deployment:
          app_name: pin-bill-day
          imageName: kubectl
          imageTag: 12.0.0.x.0
          schedule: "*/5 * * * *"
          serviceAccountName: job-scheduler
          app_dir: /oms/apps/pin_billd
          app_cmd: pin_bill_day
          app_cmd_args: "-verbose"
      ```

   c. Enter this command:

      ```bash
      kubectl apply -f _cronjob.yaml
      ```

Configuring Pricing Design Center

Configuring PDC involves these high-level steps:
1. Overriding the PDC-specific keys in the `values.yaml` file. See "Adding PDC Keys".
2. Cleaning up sample BRM and pipeline data in the database. See "Cleaning Up Sample BRM Data".
3. After PDC is deployed, enabling SSL in the WebLogic domain. See "Enabling SSL in Your PDC Domain".

**Note:**

If you want to use an existing PDC schema, point your PDC schema to PDC 12.0.0.x.0 before you deploy PDC. If the database schema is from a previous PDC release, first upgrade the schema to PDC 12.0.0.x.0.

---

### Adding PDC Keys

Table 3-4 lists the keys that directly impact PDC deployment. Add these keys to your `override-values.yaml` file with the same path hierarchy.

**Caution:**

Keys with the path `ocpdc.secretEnv` hold sensitive data. They must be handled carefully with controlled access to the file containing its values. Encode all of these values in base64. See "Secrets" in Kubernetes Concepts.

#### Table 3-4 Pricing Design Center Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Path in <code>values.yaml</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adminUser</td>
<td>ocpdc.config Env</td>
<td>The WebLogic Server domain administrator user name, which is usually <code>weblogic</code>.</td>
</tr>
<tr>
<td>rcuJdbcURL</td>
<td>ocpdc.config Env</td>
<td>The connection string to connect to a database where schemas needed by Oracle Fusion Middleware products will be created, especially OPSS. Use the format &quot;host:port/service&quot;.</td>
</tr>
<tr>
<td>rcuSysDBAUser</td>
<td>ocpdc.config Env</td>
<td>The database user name. The default user name for Oracle databases is <code>SYS</code>.</td>
</tr>
<tr>
<td>brmDBHostName</td>
<td>ocpdc.config Env</td>
<td>The IP address or host name of the machine on which the BRM pipeline database is configured. Ensure that this value matches the database details present under <code>ocbrm.db</code>.</td>
</tr>
<tr>
<td>brmDBPort</td>
<td>ocpdc.config Env</td>
<td>The port number for the BRM pipeline database. Ensure that this value matches the database details present under <code>ocbrm.db</code>.</td>
</tr>
<tr>
<td>brmDBService</td>
<td>ocpdc.config Env</td>
<td>The name of the BRM pipeline database service. Ensure that this value matches the database details present under <code>ocbrm.db</code>.</td>
</tr>
</tbody>
</table>
Table 3-4  (Cont.) Pricing Design Center Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Path in values.yaml</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>brmDBSID</td>
<td>ocpdc.config Env</td>
<td>The name of the BRM pipeline database SID. Ensure that this value matches the database details present under ocbrm.db.</td>
</tr>
<tr>
<td>brmDBUserName</td>
<td>ocpdc.config Env</td>
<td>The BRM pipeline database schema user name. Ensure that this value matches the database details present under ocbrm.db.</td>
</tr>
<tr>
<td>crossRefDBHostName</td>
<td>ocpdc.config Env</td>
<td>The IP address or the host name of the machine on which the transformation cross-reference database is configured.</td>
</tr>
<tr>
<td>crossRefDBPort</td>
<td>ocpdc.config Env</td>
<td>The port number assigned to the transformation cross-reference database service.</td>
</tr>
<tr>
<td>crossRefDBService</td>
<td>ocpdc.config Env</td>
<td>The name of the transformation cross-reference database service.</td>
</tr>
<tr>
<td>crossRefDBUserName</td>
<td>ocpdc.config Env</td>
<td>The transformation cross-reference database user name. This user should have the following capabilities on the transformation cross-reference database: create user, grant any role, grant any privileges, select any table for Enterprise edition, and DBA for XE.</td>
</tr>
<tr>
<td>crossRefSchemaPDCTableSpace</td>
<td>ocpdc.config Env</td>
<td>The name of the PDC tablespace for the transformation cross-reference schema. This field is case-sensitive.</td>
</tr>
<tr>
<td>crossRefSchemaTempTableSpace</td>
<td>ocpdc.config Env</td>
<td>The name of the temporary tablespace for the transformation cross-reference schema. This field is case-sensitive.</td>
</tr>
<tr>
<td>crossRefSchemaUserName</td>
<td>ocpdc.config Env</td>
<td>The cross-reference database schema user name.</td>
</tr>
<tr>
<td>pdcAdminUser</td>
<td>ocpdc.config Env</td>
<td>The PDC admin user name, which includes the Pricing Design Admin role.</td>
</tr>
<tr>
<td>pdcDBHostName</td>
<td>ocpdc.config Env</td>
<td>The IP address or the hostname of the machine on which the PDC database is configured.</td>
</tr>
<tr>
<td>pdcDBPort</td>
<td>ocpdc.config Env</td>
<td>The port number assigned to the PDC database service.</td>
</tr>
<tr>
<td>pdcDBService</td>
<td>ocpdc.config Env</td>
<td>The name of the PDC database service.</td>
</tr>
<tr>
<td>pdcDBUserName</td>
<td>ocpdc.config Env</td>
<td>The PDC database user name. This should be the SYSDBA user or the user should have the following capabilities on the PDC database: create user, grant any role, grant any privileges, select any table for Enterprise edition, and DBA for XE.</td>
</tr>
<tr>
<td>pdcSchemaPDCTableSpace</td>
<td>ocpdc.config Env</td>
<td>The tablespace name of the PDC schema. This field is case sensitive.</td>
</tr>
<tr>
<td>pdcSchemaTempTableSpace</td>
<td>ocpdc.config Env</td>
<td>The tablespace name of the temporary schema. This field is case sensitive.</td>
</tr>
</tbody>
</table>
Table 3-4  (Cont.) Pricing Design Center Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Path in values.yaml</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pdcSchemaUserName</td>
<td>ocpdc.config Env</td>
<td>The PDC database schema user name.</td>
</tr>
<tr>
<td>useExistingData</td>
<td>ocpdc.config Env</td>
<td>Flag that specifies whether to use the existing database schema.</td>
</tr>
<tr>
<td>adminPassword</td>
<td>ocpdc.secret Env</td>
<td>The password of the WebLogic Domain's administrative user, which is used for accessing the WebLogic Console for administrative operations.</td>
</tr>
<tr>
<td>rcuSysDBAPassword</td>
<td>ocpdc.secret Env</td>
<td>The password of the Database Administrator for the database pointed to in rcuJdbcURL.</td>
</tr>
<tr>
<td>rcuSchemaPassword</td>
<td>ocpdc.secret Env</td>
<td>The password for the schemas of Oracle Fusion Middleware products that will be created by RCU, which is used by OPSS.</td>
</tr>
<tr>
<td>brmDBPassword</td>
<td>ocpdc.secret Env</td>
<td>The BRM pipeline database schema user password. Ensure this value matches the database details present under ocbcrm.db.</td>
</tr>
<tr>
<td>brmXrefDBAPassword</td>
<td>ocpdc.secret Env</td>
<td>The transformation cross-reference database administrator user password.</td>
</tr>
<tr>
<td>brmXrefSchemaPassword</td>
<td>ocpdc.secret Env</td>
<td>The transformation cross-reference database schema user password.</td>
</tr>
<tr>
<td>brmIntegWalletPassword</td>
<td>ocpdc.secret Env</td>
<td>The BRM Integration Pack wallet password.</td>
</tr>
<tr>
<td>pdcDBAPassword</td>
<td>ocpdc.secret Env</td>
<td>The PDC database administrator user password.</td>
</tr>
<tr>
<td>pdcSchemaPassword</td>
<td>ocpdc.secret Env</td>
<td>The PDC database schema user password.</td>
</tr>
<tr>
<td>pdcWalletPassword</td>
<td>ocpdc.secret Env</td>
<td>The PDC wallet password.</td>
</tr>
<tr>
<td>pdcAdminUserPassword</td>
<td>ocpdc.secret Env</td>
<td>The PDC admin user password, which includes the Pricing Design Admin role.</td>
</tr>
</tbody>
</table>

Cleaning Up Sample BRM Data

The BRM installer populates your system with sample data, such as setup and pricing components. When using PDC, don't initiate this data in BRM. Instead, define the data in PDC and then publish it to the BRM database.

To ensure that PDC and BRM are integrated properly, clean up the BRM sample data before you start using PDC.
Cleaning Sample BRM Data

To clean up sample BRM data:

1. Copy the `update_pin_tables.sql` script from the `oc-cn-helm-chart/SQLScripts` directory under `oc-cn-helm-chart-12.0.0.x.0.tgz` to a Linux box where the Oracle database client is accessible.

2. Start SQL*Plus by entering this:

   ```
   sqlplus user_name/password@database_name
   ```

   where:
   - `user_name` is the BRM database user name.
   - `password` is the BRM database user password.
   - `database_name` is the service name or database alias for the BRM database.

3. Clean up the BRM real-time data by running this script:

   ```
   update_pin_tables.sql
   ```

4. Exit SQL*Plus by entering this:

   ```
   Exit
   ```

Cleaning Sample Pipeline Data

To clean up sample Pipeline data:

1. Copy the `update_ifw_tables.sql` script from the `oc-cn-helm-chart/SQLScripts` directory under `oc-cn-helm-chart-12.0.0.x.0.tgz` to a Linux box where the Oracle database client is accessible.

2. Start SQL*Plus by entering this:

   ```
   sqlplus user_name/password@database_name
   ```

   where:
   - `user_name` is the Pipeline database user name.
   - `password` is the Pipeline database user password.
   - `database_name` is the service name or database alias for the Pipeline database.
3. Clean up the pipeline data by running this script:

   update_ifw_tables.sql

4. Exit SQL*Plus by entering this:

   Exit

Enabling SSL in Your PDC Domain

To enable SSL in your PDC domain:

1. Copy Java KeyStore (JKS) files with valid certificates to the `mountPath` of `volMntKeyStore` (the default path is `/u01/app/share/keystore`).

2. Enable Java Secure Socket Extension (JSSE)-based SSL in the WebLogic Server Administration Console for the server on which PDC is deployed. See the Oracle WebLogic Server Administration Console Help for more information.

Configuring Billing Care

Configuring Billing Care involves these high-level steps:

1. Overriding the Billing Care-specific keys in the `values.yaml` file. See "Adding Billing Care Keys".

2. Enabling SSL in the WebLogic domain where Billing Care is deployed. See "Enabling SSL in Your Billing Care Domain".

3. Defining the authorization rules for your users and groups. See "About Authorization Policy for Billing Care".

4. Setting up your Billing Care log files. See "About Billing Care Logs".

5. Setting up volume mounts for Billing Care. See "About Billing Care Volume Mounts".

Adding Billing Care Keys

Table 3-5 lists the keys that directly impact Billing Care. Add these keys to your `override-values.yaml` file with the same path hierarchy.

⚠️ Caution:

Keys with the path `ocbc.bc.secretEnv` hold sensitive data. They must be handled carefully with controlled access to the file containing its values. Encode all of these values in base64. See "Secrets" in Kubernetes Concepts.
### Table 3-5  Billing Care Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Path in values.yaml File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyStoreAlias</td>
<td>ocbc.bc.configEnv</td>
<td>The string alias used to store and retrieve the server's private key. This is used for setting up the SSL-enabled domain.</td>
</tr>
<tr>
<td>rcjJdbcURL</td>
<td>ocbc.bc.configEnv</td>
<td>The connection string for connecting to the database where schemas needed by Oracle Fusion Middleware products will be created, especially OPSS.</td>
</tr>
<tr>
<td>ldapHost</td>
<td>ocbc.bc.configEnv</td>
<td>The host name or IP address of the LDAP Server (for example, OUD) where users and groups will be configured for access to Billing Care.</td>
</tr>
<tr>
<td>ldapGroupBase</td>
<td>ocbc.bc.configEnv</td>
<td>The LDAP base DN that contains groups.</td>
</tr>
<tr>
<td>ldapUserBase</td>
<td>ocbc.bc.configEnv</td>
<td>The LDAP base DN that contains users.</td>
</tr>
<tr>
<td>bipUrl</td>
<td>ocbc.bc.configEnv</td>
<td>The URL for PublicReportService_v11 from your BI Publisher instance, which is used by Billing Care to show invoices.</td>
</tr>
<tr>
<td>bipUserId</td>
<td>ocbc.bc.configEnv</td>
<td>The name of the user with access to the BI Publisher instance for viewing invoices from Billing Care.</td>
</tr>
<tr>
<td>adminPassword</td>
<td>ocbc.bc.configEnv</td>
<td>The password of the WebLogic domain’s administrative user, which is used for accessing the WebLogic Console for administrative operations.</td>
</tr>
<tr>
<td>ldapPassword</td>
<td>ocbc.bc.secretEnv</td>
<td>The password of the LDAP Server admin user.</td>
</tr>
<tr>
<td>walletPassword</td>
<td>ocbc.bc.secretEnv</td>
<td>The password for storing wallet sensitive data for BRM connections.</td>
</tr>
<tr>
<td>bipPassword</td>
<td>ocbc.bc.secretEnv</td>
<td>The password of the BI Publisher instance.</td>
</tr>
<tr>
<td>rcuSysDBAPassword</td>
<td>ocbc.bc.secretEnv</td>
<td>The password for the rcuJdbcURL database administrator.</td>
</tr>
<tr>
<td>rcuSchemaPassword</td>
<td>ocbc.bc.secretEnv</td>
<td>The passwords for the schemas of Oracle Fusion Middleware products that will be created by RCU, which is used by OPSS.</td>
</tr>
<tr>
<td>keyStoreIdentityKeyPass</td>
<td>ocbc.bc.secretEnv</td>
<td>The KeyPass of Identity Keystore, which is used for setting up the SSL-enabled domain.</td>
</tr>
<tr>
<td>keyStoreIdentityStorePass</td>
<td>ocbc.bc.secretEnv</td>
<td>The StorePass of Identity Keystore, which is used for setting up the SSL-enabled domain.</td>
</tr>
<tr>
<td>keyStoreTrustStorePass</td>
<td>ocbc.bc.secretEnv</td>
<td>The StorePass of Trust Keystore, which is used for setting up the SSL-enabled domain.</td>
</tr>
<tr>
<td>hostPath</td>
<td>ocbc.bc.pvPolicy</td>
<td>The location on the host system of the system-jazn-data.xml file. This applies only if you are using hostPath as the volume mount.</td>
</tr>
</tbody>
</table>
Enabling SSL in Your Billing Care Domain

To access Billing Care over the HTTPS port, SSL must be enabled in the WebLogic domain where Billing Care is deployed. The BRM cloud native deployment package takes care of the configuration necessary to equip the WebLogic domain with SSL access.

To complete the configuration for SSL setup:

2. Copy these Java KeyStore (JKS) files with valid certificates to the `oc-cn-helm-chart/keystore_billingcare` directory.
   - `identity.jks`: Provides the certificate to identify the server.
   - `trust.jks`: Establishes trust for the certificate.

   If your KeyStore files have different file names, override the `keyStoreIdentityFileName` and `keyStoreTrustFileName` keys in the `override-values.yaml` file.

During installation, Helm uses these KeyStore files to create a secret, which will be mounted as a volume inside the Billing Care Pod.

About Authorization Policy for Billing Care

Billing Care uses Oracle Platform Security Services (OPSS) for fine-grain access control to its resources. This is achieved by configuring the roles of users and groups for each resource with relevant conditions. Configuration is captured in an XML file, typically named `system-jazn-data.xml`, and loaded by running the WebLogic Scripting Tool `migrateSecurityStore`. After `migrateSecurityStore` loads the XML file, it's moved to the container's `/u01/logs/policy_loader/archive` or `/u01/logs/policy_loader/error` directory, depending on the `migrateSecurityStore` command.

The Billing Care container takes care of all configuration. You provide the `system-jazn-data.xml` file, which dictates the authorization rules for your users and groups, through a volume mount. A sample `system-jazn-data.xml` file is included with Billing Care SDK. You can access it by installing Billing Care SDK using the Billing Care installer. See Billing Care Installation Guide.

About Billing Care Logs

Log files for Billing Care are stored in the container's `/u01/logs` directory. This directory also contains domain logs, server logs, and RCU logs.
Because the container's `/u01/logs` directory has an emptyDir volume mount, the log files will be deleted when a Pod is removed from a node. To retain your log files, you can either:

- Attach a side-car application logging service, which copies the log files outside the container for analysis.
- Change the type of volume mount and directly persist the log files on permanent storage disk.

About Billing Care Volume Mounts

The Billing Care container requires Kubernetes volume mounts for batch payments and authorization policy.

- The batch payments volume mount stores the templates and payment records.
- The authorization policy volume mount shares the `system-jazn-data.xml` file from the host system to the container file system.

The default configuration comes with a hostPath PersistentVolume. For more information, see "Configure a Pod to Use a PersistentVolume for Storage" in Kubernetes Tasks.

To use a different type of PersistentVolume, modify the `oc-cn-helm-chart/templates/volume_policy_billingcare.yaml` file for authorization policy and the `oc-cn-helm-chart/templates/volume_batchpymt_billingcare.yaml` file for batch payments.

Configuring Business Operations Center

Configuring Business Operations Center involves these high-level steps:

1. Overriding the Business Operations-specific keys in the `values.yaml` file. See "Adding Business Operations Center Keys".
2. Enabling SSL in the WebLogic domain where Business Operations Center is deployed. See "Enabling SSL in Your Business Operations Center Domain".
3. Defining the authorization rules for your users and groups. See "Configuring Authorization Policy for Business Operations Center".
4. Setting up your Business Operations Center log files. See "About Business Operations Center Logs".
5. Setting up volume mounts for Business Operations Center. See "About Business Operations Center Volume Mounts".

Adding Business Operations Center Keys

Table 3-6 lists the keys that directly impact Business Operations Center. Add these keys to your `override-values.yaml` file with the same path hierarchy.
Caution:

Keys with the path `ocboc.boc.secretEnv` hold sensitive data. They must be handled carefully with controlled access to the file containing its values. Encode all of these values in base64. See Secrets in Kubernetes Concepts.

<table>
<thead>
<tr>
<th>Key</th>
<th>Path in values.yaml</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyStoreAlias</td>
<td>ocboc.boc.configEnv</td>
<td>The keystore attribute that defines the string alias used to store and retrieve the server's private key. This is used for setting up the SSL-enabled domain.</td>
</tr>
<tr>
<td>rcuJdbcURL</td>
<td>ocboc.boc.configEnv</td>
<td>The connection string for connecting to the database where schemas needed by Oracle Fusion Middleware products are created, especially OPSS.</td>
</tr>
<tr>
<td>ldapHost</td>
<td>ocboc.boc.configEnv</td>
<td>The hostname or IP address of the LDAP Server (for example, OUD) where users and groups are configured for access to Business Operations Center.</td>
</tr>
<tr>
<td>ldapGroupBase</td>
<td>ocboc.boc.configEnv</td>
<td>The LDAP base DN that contains groups.</td>
</tr>
<tr>
<td>ldapUserBase</td>
<td>ocboc.boc.configEnv</td>
<td>The LDAP base DN that contains users.</td>
</tr>
<tr>
<td>dbHostName</td>
<td>ocboc.boc.configEnv</td>
<td>The BRM database hostname.</td>
</tr>
<tr>
<td>sysDBAlias</td>
<td>ocboc.boc.configEnv</td>
<td>The database service name.</td>
</tr>
<tr>
<td>bocSchemaUserName</td>
<td>ocboc.boc.configEnv</td>
<td>The Business Operations Center database schema user name.</td>
</tr>
<tr>
<td>bocSchemaBocTablespace</td>
<td>ocboc.boc.configEnv</td>
<td>The default tablespace for the Business Operations Center database administrator.</td>
</tr>
<tr>
<td>bocSchemaTempTablespace</td>
<td>ocboc.boc.configEnv</td>
<td>The temp tablespace for the Business Operations Center database administrator.</td>
</tr>
<tr>
<td>billingCareUrl</td>
<td>ocboc.boc.configEnv</td>
<td>The URL of the Billing Care instance that is used with your BRM Server. Leave this blank if Billing Care isn't installed in your environment.</td>
</tr>
<tr>
<td>brmServerTimeZone</td>
<td>ocboc.boc.configEnv</td>
<td>The Time Zone (TZ) value of the server operating system on which BRM is running.</td>
</tr>
<tr>
<td>bocdbURL</td>
<td>ocboc.boc.configEnv</td>
<td>The connection string for connecting to the Business Operations Center database. Use one of these formats: &quot;DatabaseHost:DatabasePort/ServiceName&quot; or &quot;DatabaseHost:DatabasePort:ServiceID&quot;. This is used to create the WebLogic data source for connecting to the Business Operations Center schema.</td>
</tr>
<tr>
<td>adminPassword</td>
<td>ocboc.boc.secretEnv</td>
<td>The password for the WebLogic domain's administrative user. This is used for accessing the WebLogic Console for administrative operations.</td>
</tr>
</tbody>
</table>
### Enabling SSL in Your Business Operations Center Domain

To access Business Operations Center over the HTTPS port, SSL must be enabled in the WebLogic domain where Business Operations Center is deployed. The BRM cloud native deployment package takes care of the configuration necessary to equip the WebLogic domain with SSL access.

To complete the configuration for SSL setup:

1. Create a `keystore_boc` directory under `oc-cn-helm-chart`.
2. Copy these Java KeyStore (JKS) files with valid certificates to the `oc-cn-helm-chart/keystore_boc` directory.
   - `identity.jks`: Provides the certificate to identify the server.
   - `trust.jks`: Establishes trust for the certificate.
If your KeyStore files have different file names, override the `keyStoreIdentityFileName` and `keyStoreTrustFileName` keys in the `override-values.yaml` file.

During installation, Helm uses these KeyStore files to create a secret, which will be mounted as a volume inside the Business Operations Center Pod.

**Configuring Authorization Policy for Business Operations Center**

Business Operations Center uses Oracle Platform Security Services (OPSS) for fine-grain access control to its resources. This is achieved by configuring the roles of users and groups for each resource with relevant conditions. Configuration is captured in an XML file, typically named `system-jazn-data.xml`, and loaded by running the WebLogic Scripting Tool `migrateSecurityStore`. After `migrateSecurityStore` loads the XML file, it's moved to the container's `/u01/logs/policy_loader/archive` or `/u01/logs/policy_loader/error` directory, depending on the `migrateSecurityStore` command.

A basic authorization policy is loaded when the Business Operations Center container is started for the first time. You can override the configurations by passing `system-jazn-data.xml` in the volume location specified in the `cboc.boc.persistentVolOpssPolicy` key.

**About Business Operations Center Logs**

Log files for Business Operations Center are stored in the container's `/u01/logs` directory. This directory also contains domain logs, server logs, and RCU logs.

Because the container's `/u01/logs` directory has an `emptyDir` volume mount, the log files will be deleted when a Pod is removed from a node. To retain your log files, you can either:

- Attach a side-car application logging service, which copies the log files outside the container for analysis.
- Change the type of volume mount and directly persist the log files on permanent storage disk.

**About Business Operations Center Volume Mounts**

The Business Operations Center container requires Kubernetes volume mounts for third-party libraries, the JDBC JAR, and the authorization policy file.

- The third-party volume mount shares the third-party libraries required by Business Operations Center from the host system to the container file system.

  For the list of third-party libraries to download, see "Additional Required Third-Party Software" in Business Operations Center Installation Guide. Place library files under the third-party volume mount.

- The JDBC JAR volume mount shares the JDBC JAR required for Business Operations Center installation from the host system to the container file system.

- The authorization policy volume mount shares the `system-jazn-data.xml` file from the host system to the container file system.

The default configuration comes with a hostPath PersistentVolume. For more information, see "Configure a Pod to Use a PersistentVolume for Storage" in Kubernetes Tasks.
To use a different type of PersistentVolume, modify the `oc-cn-helm-chart/templates/volume_boc.yaml` file.

## Installing BRM Cloud Native Services

To install BRM cloud native services and optionally initialize the BRM database schema, do this:

1. Validate your chart content by entering this command from the `helmcharts` directory:

   ```bash
   helm lint --strict oc-cn-helm-chart
   ```

   You'll see this if the command completes successfully:

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

2. Install BRM cloud native services by entering this command from the `helmcharts` directory:

   ```bash
   helm install oc-cn-helm-chart --name ReleaseName --namespace NameSpace --values OverrideValuesFile
   ```

   where:
   - `ReleaseName` is the release name, which is used to track this installation instance.
   - `NameSpace` is the name space in which to create BRM Kubernetes objects.
   - `OverrideValuesFile` is the path to a YAML file that overrides the default configurations in the chart's `values.yaml` file.

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

   ```bash
   helm install oc-cn-helm-chart --name occn-ps2 --namespace ocgbu --values override-values.yaml
   ```

3. To initialize the BRM database schema, enter this command from the `helmcharts` directory:

   ```bash
   helm install oc-cn-init-db-helm-chart --name ReleaseName --namespace NameSpace --values OverrideValuesFile
   ```
Customizing BRM Cloud Native Services

This chapter describes how to customize your Oracle Communications Billing and Revenue Management (BRM) server and clients in a cloud native environment.

The Docker build commands in this chapter reference Dockerfile and related scripts as is from the `oc-cn-docker-files-12.0.0.2.0.tgz` package. Ensure that you use your own version of Dockerfile and related scripts before running the build command.

Topics in this chapter

• Customizing BRM Server
• Customizing Billing Care

⚠️ WARNING:

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

Customizing BRM Server

You can customize the BRM Server by doing one of these:

• Layering the BRM cloud native Docker image with a customized library file.
• Creating the BRM cloud native Docker image with the default Dockerfile and then layering a customized library.

For example, you could extend `fm_subscription_pol_custom.so` by doing this:

1. Create the Docker image by going to the `$PIN_HOME` directory and entering this:

   ```bash
   cat Dockerfile_cm
   FROM cm:12.0.0.x.0
   USER root
   COPY lib/fm_subscription_pol_custom.so $PIN_HOME/lib/
   fm_subscription_pol_custom.so
   RUN chown -R omsuser:oms $PIN_HOME/lib/fm_subscription_pol_custom.so
   RUN chmod 755 ${PIN_HOME}/lib/fm_subscription_pol_custom.so
   USER omsuser
   ```

2. Customize the `lib/fm_subscription_pol_custom.so` library file.

3. Copy the customized library file to `$PIN_HOME/lib`. 

---

ORACLE

4-1
4. Build the BRM Server image by entering this command:

   
   docker build -t cm_custom:12.0.0.x.0 -f Dockerfile_cm

Customizing Billing Care

Extensibility is one of the biggest features of on-premises Billing Care, and this same extensibility is available in the Billing Care cloud native deployment. You can override the existing Billing Care behavior, such as change labels and icons, add new flows and screens, and so on, by using Billing Care SDK.

To use the Billing Care SDK in a cloud native environment, do this:

1. Build the Billing Care SDK WAR the same way as described in Oracle Communications Billing Care SDK Guide.
2. Create a Billing Care SDK image by using the Billing Care image as a base.
3. Deploy the Billing Care SDK WAR to the same WebLogic domain where Billing Care is deployed.
4. Redeploy Billing Care with an updated deployment plan that references the SDK WAR to link it as a SharedLibrary.

The cloud native package includes all of the scripts necessary to prepare and run an SDK image. For example, if your SDK WAR is named BillingCareCustomizations.war, you would build the Billing Care SDK image like this:

2. Copy the BillingCareCustomizations.war file to your current working directory (oc-cn-docker-files/ocbc/billing_care_sdk).
3. Build the SDK image by entering this command:

   
   docker build --build-arg SDK_WAR=BillingCareCustomizations.war --build-arg BASE_TAG=12.0.0.x.0 -t oracle/billingcare_sdk:12.0.0.x.0

4. Open the override-values.yaml file and edit the keys shown in Table 4-1. This directs Helm to deploy the Billing Care SDK image rather than the Billing Care image.

Table 4-1    Billing Care SDK Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isSDK</td>
<td>ocbc.bc.deployment</td>
<td>Set this to True. This creates links to additional Kubernetes objects, such as ConfigMap and volume mounts, that are required for SDK in the Billing Care Deployment Controller. The default is false.</td>
</tr>
</tbody>
</table>
### Table 4-1  (Cont.) Billing Care SDK Keys

<table>
<thead>
<tr>
<th>Key</th>
<th>Path</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>imageName</td>
<td>ocbc.bc.deployment</td>
<td>Set this to <code>oracle/billingcare_sdk</code>. This is the name of the image, which must be used for the Billing Care Pod.</td>
</tr>
<tr>
<td>imageTag</td>
<td>ocbc.bc.deployment</td>
<td>Set this to <code>12.0.0.x.0</code>. This tags the image used for the Billing Care Pod.</td>
</tr>
<tr>
<td>deployName</td>
<td>ocbc.bc.configEnvSDK</td>
<td>The name of the SDK Library in the <code>Manifest.MF</code> file. The default is <code>BillingCareCustomizations</code>.</td>
</tr>
</tbody>
</table>

5. Install a new Helm release or upgrade the running release to use Billing Care with SDK.

The deployment plan for SDK is mounted inside the container through a ConfigMap formed by the `oc-cn-helm-chart/templates/configmap_deploy_plan_billingcare_sdk.yaml` template file.
This chapter describes how to maintain your Oracle Communications Billing and Revenue Management (BRM) cloud native deployment.

Topics in this chapter

• Managing a Helm Release
• About Application Log Files

Managing a Helm Release

After you install a Helm chart, Kubernetes manages all of its objects and deployments. All Pods created through `oc-cn-helm-chart` are wrapped in a Kubernetes controller that includes relevant health checking mechanisms. Controllers create and manage Pods. For example, if a node fails, controllers can automatically replace a Pod by scheduling an identical replacement on a different node.

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

• Tracking A Release's Status
• Updating A Release
• Checking A Release's Revision
• Rolling Back A Release To A Previous Revision

Tracking A Release's Status

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

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

```
helm status ReleaseName
```

where `ReleaseName` is the name you assigned to this installation instance.
Updating A Release

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

`helm upgrade --values OverridingValueFile --values NewOverridingValueFile ReleaseName Chart`

where:

- `Chart` is the location of the chart, which is `oc-cn-helm-chart` for the complete BRM cloud native deployment package or `oc-cn-init-db-helm-chart` for initializing the BRM database schema.
- `OverridingValueFile` is the path to the YAML file that overrides the default configurations in the `oc-cn-helm-chart/values.yaml` file.
- `NewOverridingValueFile` is the path to the YAML file that has updated values. The values in this file are newer than those defined in `values.yaml` and `OverridingValueFile`.

Checking A Release’s Revision

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

`helm history ReleaseName`

Rolling Back A Release To A Previous Revision

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

`helm rollback ReleaseName RevisionNumber`

where `RevisionNumber` is the value from the Helm history command.

About Application Log Files

Each Pod creates log files under a single location within its file system. Ensure that the sidecar application you use to retrieve the log files from the Pod:

- Stores the logs safely
- Cleans up space on the container’s file system

You can use log rotation to split the log files into smaller file sizes, so they can be easily removed from the container after the configured period.
Uninstalling Your BRM Cloud Native Deployment

This chapter describes how to uninstall your Oracle Communications Billing and Revenue Management (BRM) cloud native deployment.

Topics in this chapter

- Uninstalling Your BRM Cloud Native Deployment

Uninstalling Your BRM Cloud Native Deployment

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

To uninstall, enter this command:

```
helm delete ReleaseName
```

where ReleaseName is the name you assigned to this installation instance.

Helm keeps a record of its releases, so you can still re-activate the release after you uninstall it.

To completely remove the release from the cluster, add the --purge option to the command. For example, to completely remove a release named "occn-ps2", you'd enter this command:

```
helm delete --purge occn-ps2
```
This chapter provides guidance on how you can build your own Docker images from the Dockerfiles and related scripts that you create. This covers Oracle Communications Billing and Revenue Management (BRM), Pricing Design Center (PDC), Billing Care, and Business Operations Center applications.

The Docker build commands in this chapter reference Dockerfile and related scripts as is from the `oc-cn-docker-files-12.0.0.2.0.tgz` package. Ensure that you use your own version of Dockerfile and related scripts before running the build command.

**Topics in this chapter**

- Building BRM Server Images
- Building PDC Images
- Building Billing Care Images
- Building Business Operations Center Images

Sample Dockerfiles included in the BRM cloud native deployment package (`oc-cn-docker-files-12.0.0.2.0.tgz`) are examples that depict how default images are built for BRM. If you want to build your own images, refer to the sample Dockerfiles shipped with the product as a reference. Create your own Dockerfiles and then build your images.

⚠️ **WARNING:**

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

### Building BRM Server Images

To build images for BRM Server, your staging location (`$PIN_HOME`) must be available from where the Docker images are built. After you unpack `oc-cn-docker-files-12.0.0.2.0.tgz`, the BRM Server directory structure will be `oc-cn-docker-files/ocbrm`.

Building your own BRM Server images involves these high-level steps:

1. You build the BRM Server base image. See "Building Your BRM Server Base Image".

2. You build images for each BRM Server component. See "Building Images of BRM Server Components".

3. You layer the BRM Server images. See "Layering BRM Server Images".
Building Your BRM Server Base Image

To make your directory structure ready for building base images:

1. Edit the \$PIN_HOME/bin/orapki binary to replace the staging Java path with \$\{JAVA_HOME\}.

2. Create the \$PIN_HOME/installer directory.

3. If you're behind a proxy server, set the \$PROXY variable:

   ```
   export PROXY=ProxyHost:Port
   ```

4. Download the Java 1.8 update-221 binary (server-jre-8u221-linux-x64.tar.gz) and then copy it to \$PIN_HOME.

5. Download the Perl 5.28.1 binary (perl-5.28.1.tar.gz) and then copy it to \$PIN_HOME.

6. For your database client:

   a. Copy `client_install.rsp` (32 bit), `oracle_client_response_file.rsp` (64 bit), `downloadOracleClient.sh`, and `waitForOracleClientInst.sh` from `oc-cn-docker-files/ocbrm/base_images` to \$PIN_HOME.

   b. Modify these parameters in the `downloadOracleClient.sh` file:

      - ORACLE_CLIENT_ZIP: Enter the binary name.
      - REPOSITORY_URL: Enter the location to fetch the database client binary.

   c. If the db_client binary is already downloaded, copy the binary to the \$PIN_HOME/installer directory.

After preparing your directory structure, build your BRM Server base image:

- For database client 12CR2 (64 Bit) + Java 1.8 + Perl 5.28.1, enter this command:

  ```
  docker build --build-arg PROXY=$PROXY -t db_client_and_java_perl:12.0.0.x.0 -f DockerFileLocation/Dockerfile_db_client_and_java_perl .
  ```

- For database client 12CR2 (64 Bit) + Java 1.8, enter this command:

  ```
  docker build --build-arg PROXY=$PROXY -t db_client_and_java:12.0.0.x.0 -f DockerFileLocation/Dockerfile_db_client_and_java .
  ```

- For Java 1.8, enter this command:

  ```
  docker build --build-arg PROXY=$PROXY -t java:12.0.0.x.0 -f DockerFileLocation/Dockerfile_java .
  ```

- For Java 1.8 + Perl 5.28.1, enter this command:

  ```
  docker build --build-arg PROXY=$PROXY -t java_perl:12.0.0.x.0 -f DockerFileLocation/Dockerfile_java_perl .
  ```
• For database client 12CR2 (32 Bit) + Java 1.8, enter this command:

```
docker build --build-arg PROXY=$PROXY -t db_client_32_and_java:12.0.0.x.0 -f DockerFileLocation/Dockerfile_db_client_32_and_java .
```

**Note:**

When the existing database is used with custom build Docker images, override the `ocbrm.init_database` key in the Helm chart with a value of `false`.

### Building Images of BRM Server Components

The [oc-cn-docker-files-12.0.0.2.0.tgz](#) package includes references to all of the Dockerfiles and scripts needed to build images of BRM Server components (except for `oraclelinux:7-slim`).

To build an image of a BRM Server component:

1. Copy these scripts from the `oc-cn-docker-files/ocbrm` directory to `$PIN_HOME`:
   - `entrypoint.sh`
   - `createWallet.sh`
   - `pvt-utils.jar`
   - `cm/preStopHook.sh_cm`
   - `cm/postStartHook.sh`
   - `eai_js/preStopHook.sh_eai`

2. Do one of these:
   - For batch-pipeline and real-time pipeline, copy `entrypoint.sh` and `createWallet.sh` to `$PIN_HOME/..`, and copy `$PIN_HOME/..setup/BRMActions.jar` to the `$PIN_HOME/jars` directory for building the images.
   - For all other components, copy the `$PIN_HOME/..setup/BRMActions.jar` file to `$PIN_HOME`.

3. Set these environment variables:
   - `$PERL_HOME`: Set this to the path of Perl 5.28.1.
   - `$JAVA_HOME`: Set this to the path of Java 1.8 update 221.

4. Build the image for your BRM component.
   For example, to build a CM image, you'd enter this:

```
docker build -t cm:12.0.0.x.0 --build-arg STAGE_PIN_HOME=$PIN_HOME --build-arg STAGE_JAVA_HOME=$JAVA_HOME --build-arg STAGE_PERL_HOME=$PERL_HOME -f DockerFileLocation/Dockerfile .
```
To build a dm-oracle image, you'd enter this:

docker build --force-rm=true --no-cache=true -t dm_oracle:12.0.0.x.0 -f DockerfileLocation/Dockerfile.

where DockerfileLocation is the path to the Dockerfiles for your BRM component.

Note: Build batch and realtime pipeline images from $PIN_HOME/..

Layering BRM Server Images

To build custom docker images, you can layer the default images provided by Oracle.

For example, to layer dm-oracle, you could add these commands to its Dockerfile:

FROM dm_oracle:12.0.0.x.0
ENV PIN_HOME /oms
USER root
COPY wallet/* ${PIN_HOME}/wallet/
RUN chown -R omsuser:oms ${PIN_HOME}/wallet
RUN chown -R omsuser:oms ${PIN_HOME}
USER omsuser
WORKDIR $PIN_HOME/sys/dm_oracle

If you need to load a utility as part of the cm container, do this:

1. Copy restart_cm from the Dockerfile TAR to $PIN_HOME.
2. Add the utility you want to load to the restart_cm file.
3. Add the complete utility command from line number 42. For example:

   $PIN_HOME/bin/load_config -v $PIN_HOME/sys/data/config/ config_note_type.xml

4. Create the Dockerfile_utility file by entering this:

   cat Dockerfile_utility

      FROM cm:12.0.0.2.0
      USER root
      COPY restart_cm /oms/bin/restart_cm
      RUN chown -R omsuser:oms ${PIN_HOME}
      USER omsuser
      WORKDIR $PIN_HOME/sys/cm

5. Build the cm image by entering this command:

   docker build -t cm_custom:12.0.0.x.0 -f Dockerfile_utility.
6. Update the image name and tag in the `override-values-brm.yaml` file under `ocbrm.cm.deployment.imageName`.

### Building PDC Images

Building your own PDC images involves these high-level steps:

1. You build the base image for PDC. See "Building the Base PDC Image".
2. You build the Oracle PDC image. See "Building the Oracle PDC Image".

### Building the Base PDC Image

All images from the BRM cloud native deployment package use Oracle Linux as the base image. Oracle Linux is available from both Docker Hub ([https://hub.docker.com/](https://hub.docker.com/)) and Oracle Container Registry ([http://container-registry.oracle.com](http://container-registry.oracle.com)). You can pull the image from either of these repositories.

To build the base PDC image, do this:

1. Ensure that the Oracle Linux image is present in your local system, where you will build other images, with the name `oraclelinux:7-slim`.
2. Download these packages to the `ParentFolder/Docker_files/BasePDCImage/container-scripts` directory:
   - fmw_12.2.1.3.0_infrastructure_Disk1_1of1.zip
   - p28186730_139400_Generic.zip
   - p29921455_12213190416_Generic.zip
   - p29016089_122130_Generic.zip
   - perl-5.28.1.tar.gz
   - server-jre-8u221-linux-x64.tar.gz
   - ojdbc8.jar
   - orai18n.jar
3. Build the base PDC image by going to `ParentFolder/Docker_files/BasePDCImage` and entering this command:

   ```
   docker build --force-rm=true --no-cache=true -t oracle/pvt_cloud_pdc_base_image:12.0.0.x.0 -f Dockerfile
   ```

### Building the Oracle PDC Image

To build the Oracle PDC image:

1. Download the brm-apps and realtimepipe Docker images from the repository by entering this command:

   ```
   docker pull RepoHost:RepoPort/ImageName
   ```

   where:

   ```
   ```
2. Tag the Docker images by entering these commands:

```sh
docker tag RepoHost:RepoPort/brm_apps:12.0.0.x.0 brm_apps:12.0.0.x.0
docker tag RepoHost:RepoPort/realt imepipe:12.0.0.x.0 realtimepipe:12.0.0.x.0
```

3. Download `pdcserver-12.0.0.x.0_generic_full.jar` to the `ParentFolder1/Docker_files/PDCImage/container-scripts` directory.

4. Build your Oracle PDC image by entering this command from the `ParentFolder1/Docker_files/PDCImage` directory:

```sh
docker build --force-rm=true --no-cache=true -t oracle/pdcapp:12.0.0.x.0 -f Dockerfile
```

### Building Billing Care Images

The Billing Care image includes Linux OS, Java, and Fusion Middleware Infrastructure along with its own installer `BillingCare_generic.jar` file. To build your own image of Billing Care, you must have these base images ready.

The `oc-cn-docker-files-12.0.0.x.0.tgz` package includes references to all of the Dockerfiles and scripts needed to build images of Billing Care, except for `oraclelinux:7-slim`. You can refer to them when building a Billing Care image in your own environment.

### Building the Oracle Linux Image

All images from the BRM cloud native deployment package use Oracle Linux as the base image. Oracle Linux is available from both Docker Hub (https://hub.docker.com/) and Oracle Container Registry (http://container-registry.oracle.com). You can pull the image from either of these repositories. After you pull the Oracle Linux image, ensure that it's present in your local system, where you will build other images, with the name `oraclelinux:7-slim`.

You can check if the Oracle Linux image was pulled successfully by entering this command:

```sh
docker image ls
```

The command returns this if you pulled the image from the Oracle Container Registry:

`container-registry.oracle.com/os/oraclelinux:7-slim`

You can then retag the image by entering this command:

```sh
docker tag container-registry.oracle.com/os/oraclelinux:7-slim oraclelinux:7-slim
```
Building the Oracle Server JRE Image

The Oracle Server JRE image is designed for server-side applications and includes the commonly required features from the JRE and JDK. You can either:

• Pull the image from Docker Hub (https://hub.docker.com/) or Oracle Container Registry (http://container-registry.oracle.com)
• Build the image by using the Dockerfiles in oc-cn-docker-files/ocbc/serverjre.

To build the Oracle Server JRE image, do this:

1. Download the server-jre-8u221-linux-x64.tar.gz file and save it to the oc-cn-docker-files/ocbc/serverjre directory.
2. Build the JRE image by entering this command from the oc-cn-docker-files/ocbc/serverjre directory:

   docker build -t oracle/serverjre:8 .

Building the Oracle Fusion Middleware Infrastructure Image

To build the Oracle Fusion Middleware Infrastructure image, do this:

2. Download the Oracle Fusion Middleware Infrastructure 12.2.1.3 Zip installation file.
3. Download the security patches and prerequisites listed in Table 7-1. These patches address common security vulnerabilities.

   Table 7-1 Security Patches

<table>
<thead>
<tr>
<th>Patch Number</th>
<th>Product Patched</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>28186730</td>
<td>Oracle Global Lifecycle Management OPatch</td>
<td>p28186730_139400_Generic.zip</td>
</tr>
<tr>
<td>26045997</td>
<td>Oracle JDBC for Fusion Middleware 12.2.1.3.0</td>
<td>p26045997_122130_Generic.zip</td>
</tr>
<tr>
<td>29016089</td>
<td>Oracle WebLogic Server</td>
<td>p29016089_122130_Generic.zip</td>
</tr>
<tr>
<td>29921455</td>
<td>Oracle WebLogic Server</td>
<td>p29921455_12213190416_Generic.zip</td>
</tr>
</tbody>
</table>

4. Copy these Zip files to the current working directory (oc-cn-docker-files/ocbc/fmw).

   • fmw_12.2.1.3.0_infrastructure_Disk1_1of1.zip
   • p26045997_122130_Generic.zip
   • p28186730_139400_Generic.zip
   • p29016089_122130_Generic.zip
   • p29921455_12213190416_Generic.zip
5. Build the Oracle Fusion Middleware Infrastructure image by entering this command:

   `docker build -t oracle/fmw-infrastructure:12.2.1.3 .`

The Fusion Middleware Infrastructure requires the *libaio* library, which isn't included in *oraclelinux:7-slim*. To download it from the Yum repository during the image build process, Docker must have internet access. If you're behind a proxy server, your build command will look like this:

   `docker build --build-arg http_proxy=ProxyHost:Port --build-arg https_proxy=ProxyHost:Port -t oracle/fmw-infrastructure:12.2.1.3 .`

### Building the Billing Care Image

To build the Billing Care image, do this:

2. Download the Oracle Communications Billing Care installation JAR file.
4. Build the Billing Care image by entering this command:

   `docker build -t oracle/billingcare:12.0.0.x.0 .`

### Building Business Operations Center Images

As shown in Figure 7-1, the Business Operations Center image includes the Linux operating system, Java, and Fusion Middleware Infrastructure along with its own *BusinessOperationsCenter_generic.jar* installation file.
The Business Operations Center image also uses the brm-apps image and copies its files to support the BRM applications in the Business Operations Center container. To build your image of Business Operations Center, you must have these base images ready. The \texttt{oc-cn-docker-files-12.0.0.x.0.tgz} package includes references to Dockerfiles and scripts you need to build images of BRM components, except for \texttt{oraclelinux:7-slim}. You can refer to them for building the Business Operations Center image in your own environment.

### Building the Oracle Linux Image

All images from the BRM cloud native deployment package use Oracle Linux as the base image. Oracle Linux is available from both Docker Hub (https://hub.docker.com/) and Oracle Container Registry (http://container-registry.oracle.com). You can pull the image from either of these repositories. After you pull the Oracle Linux image, ensure that it's present in your local system, where you will build other images, with the name \texttt{oraclelinux:7-slim}.

You can check if the Oracle Linux image was pulled successfully by entering this command:

\begin{verbatim}
docker image ls
\end{verbatim}

The command returns this if you pulled the image from the Oracle Container Registry:

\begin{verbatim}
container-registry.oracle.com/os/oraclelinux:7-slim
\end{verbatim}
You can then retag the image by entering this command:

```
docker tag container-registry.oracle.com/os/oraclelinux:7-slim oraclelinux:7-slim
```

Building the Oracle Server JRE Image

The Oracle Server JRE image is designed for server-side applications and includes the commonly required features from the JRE and JDK. You can either:

- Pull the image from Docker Hub (https://hub.docker.com/) or Oracle Container Registry (http://container-registry.oracle.com)
- Build the image by using the Dockerfiles in `oc-cn-docker-files/ocboc/serverjre`.

To build the Oracle Server JRE image, do this:

1. Download the `server-jre-8u221-linux-x64.tar.gz` file and save it to the `oc-cn-docker-files/ocboc/serverjre` directory.
2. Build the JRE image by entering this command from the `oc-cn-docker-files/ocboc/serverjre` directory:

```
docker build -t oracle/serverjre:8 .
```

Building the Oracle Fusion Middleware Infrastructure Image

To build the Oracle Fusion Middleware Infrastructure image, do this:

1. Go to the `oc-cn-dockerfiles/ocboc/fmw` directory.
2. Download the Oracle Fusion Middleware Infrastructure 12.2.1.3 Zip installation file.
3. Download the security patches and prerequisites listed in Table 7-2. These patches address common security vulnerabilities.

<table>
<thead>
<tr>
<th>Patch Number</th>
<th>Product Patched</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>28186730</td>
<td>Oracle Global Lifecycle Management OPatch</td>
<td>p28186730_139400_Generic.zip</td>
</tr>
<tr>
<td>26045997</td>
<td>Oracle JDBC for Fusion Middleware 12.2.1.3.0</td>
<td>p26045997_122130_Generic.zip</td>
</tr>
<tr>
<td>29016089</td>
<td>Oracle WebLogic Server</td>
<td>p29016089_122130_Generic.zip</td>
</tr>
<tr>
<td>29921455</td>
<td>Oracle WebLogic Server</td>
<td>p29921455_12213190416_Generic.zip</td>
</tr>
</tbody>
</table>

4. Copy these Zip files to the current working directory (`oc-cn-dockerfiles/ocboc/fmw`).
   - `fmw_12.2.1.3.0_infrastructure_Disk1_1of1.zip`
   - `p26045997_122130_Generic.zip`
5. Build the Oracle Fusion Middleware Infrastructure image by entering this command:

```
docker build -t oracle/fmw-infrastructure:12.2.1.3 .
```

The Fusion Middleware Infrastructure requires the `libaio` library, which isn't included in `oraclelinux:7-slim`. To download it from the Yum repository during the image build process, Docker must have internet access. If you’re behind a proxy server, your build command will look like this:

```
docker build --build-arg http_proxy=ProxyHost:Port --build-arg https_proxy=ProxyHost:Port -t oracle/fmw-infrastructure:12.2.1.3 .
```

### Building the Oracle BRM Apps Image

You can build the brm-apps image by using the Dockerfiles in `oc-cn-dockerfiles/brm/brm_apps`. To build the brm-apps image, see "Building Images of BRM Server Components".

You can check if the brm-apps image was built successfully by entering this command:

```
docker image ls
```

You can then retag the image by entering this command:

```
docker tag brm_apps:Tag brm_apps:12.0.0.x.0
```

### Building the Oracle Business Operations Center Image

To build the Oracle Communications Business Operations Center image:

1. Go to the `oc-cn-dockerfiles/ocboc/boc` directory.
2. Download the Oracle Business Operations Center JAR installer.
3. Copy the `BusinessOperationsCenter_generic.jar` file to your current working directory (`oc-cn-dockerfiles/ocboc/boc`).
4. Build the Business Operations Center image by entering this command:

```
docker build -t oracle/boc:12.0.0.x.0 .
```

Business Opearations Center requires `openssh-clients`, `openssh-server`, and other libraries, which are not part of `oraclelinux:7-slim`. To download it from the
Yum repository during the image build process, Docker must have internet access. If you're behind a proxy server, your build command will look like this:

```
docker build --build-arg http_proxy=ProxyHost:Port --build-arg https_proxy=ProxyHost:Port -t oracle/boc:12.0.0.x.0.
```
Deploying into Oracle Cloud Infrastructure

This chapter describes how to deploy Oracle Communications Billing and Revenue Management (BRM) cloud native services into Oracle Cloud Infrastructure.

Topics in this chapter

• Deploying into Oracle Cloud Infrastructure

Deploying into Oracle Cloud Infrastructure

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

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

1. Sign up for Oracle Cloud Infrastructure.
2. Create a database system on a bare metal or virtual machine instance.
   Select a database version that is compatible with the BRM cloud native software requirements. See “Software Compatibility”.
3. Create a Kubernetes cluster and deselect the Tiller (Helm) Enabled option. The version of Helm used by Oracle Cloud Infrastructure isn't compatible with the BRM cloud native software requirements.
4. Install and configure the Oracle Cloud Infrastructure Command Line Interface (CLI).
   CLI is a small footprint tool that you can use on its own or with the Console to complete OCI tasks. It's needed here to download the `kubeconfig` file.
5. Install and configure `kubectl` on your system to perform operations on your cluster in Oracle Cloud Infrastructure.
6. The `kubeconfig` file (by default named `config` and stored in the `$HOME/.kube` directory) provides the necessary details to access the cluster using `kubectl` and the Kubernetes Dashboard.
Download **kubeconfig** to access your cluster on Oracle Cloud Infrastructure by entering this command:

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

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

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

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

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

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

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

```
helm init
```

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

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

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

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

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

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