Oracle® Banking Microservices Architecture

Containerization Guide

Release 14.6.1.0.0
F61112-01
August 2022

Oracle Banking Microservices Architecture Containerization Guide, Release 14.6.1.0.0

F61112-01

Contents

1 Technologies

2 Containerization

3 Oracle Banking Microservices Architecture Products Deployment Approaches

3.1 Containerization of the Services Using Tomcat 3-1
3.1.1 Using Jib Plugin and Tomcat Image 3-1
3.1.2 Pipeline Integration in Jenkins 3-2
3.1.3 Using War Artifacts Delivered in OSDC 3-4
3.2 Containerization of the Services Using WebLogic 3-6
3.2.1 Using Pre-Built WebLogic Images 3-6
3.2.2 Run WebLogic Containers Using WebLogic Kubernetes Operator 3-7
3.3 Deploying Services without Docker Images 3-7
3.3.1 Deploying Applications to Tomcat without Docker Images 3-8
3.3.2 Deploying Applications to WebLogic without Docker Images 3-8
3.4 Deploying Services on Private Cloud using Docker Images 3-9

Index

Preface

- Purpose
- Audience
- Acronyms and Abbreviations
- List of Topics
- Related Documents

Purpose

This guide provides the information on how to deploy Oracle Banking Microservices Architecture products by creating a Docker image and deploying it in a Docker container or inside a Kubernetes (K8) cluster.

Audience

This guide is intended for WebLogic admin or ops-web team who are responsible for installing OFSS Banking Products. The user of the guide should have pre-acquired skills in the below technologies to perform the steps mentioned in this guide:

- Docker
- Kubernetes
- Jenkins

Acronyms and Abbreviations

The list of the acronyms and abbreviations that are used in this guide are as follows:

Table Acronyms and Abbreviations

Abbreviation	Description
OSDC	Oracle Software Delivery Cloud
DDL	Data Definition Language
DML	Data Manipulation Language

List of Topics

This guide is organized as follows:

Table List of Topics

Topics	Description
Technologies	This topic provides information about the Technologies.
Containerization	This topic provides information about the Containerization.

Table (Cont.) List of Topics

Topics	Description
Oracle Banking Microservices Architecture Products Deployment Approaches	This topic provides information about Oracle Banking Microservices Architecture Products deployment approaches.

Related Documents

For more information, refer to the following documents:

- Product Installation Guide

1 Technologies

This topic describes about the various technologies used in Oracle Banking Microservices Architecture.

Docker

Images and Containers
An image is a read-only template with instructions for creating a Docker container and an image is based on another image.

A container is a standard unit of software that packages up code and all its dependencies. Hence, the application runs quickly and reliably from one environment to another.

A Docker Container Image is a lightweight, standalone, executable package of software that includes everything needed to run an application such as code, runtime, system tools, system libraries, and settings.

Container images become containers at runtime and for Docker containers, the images become containers when they run on the engine. Containers are available for both Linux and Windows-based applications. The containerized software always runs the same code, regardless of the infrastructure. The container isolates software from its environment and ensures that it works uniformly despite differences for instance between Development, Staging, and Production.

Kubernetes (K8)

Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications. It groups the containers that makes an application into logical units for easy management and discovery.

2
Containerization

This topic describes about the various containerization process.

Docker Registry

A Docker registry is a service for storing and retrieving the Docker images. A registry contains a collection of one or more Docker image repositories. Each image repository contains one or more tagged images. The Docker provides its own registry and the Docker Hub, but users can also use private or third-party registries.

> **Note:**
>
> The user must self register in Oracle Container Registry to access the images located in this registry.

Database

The Database is not included inside the Docker and the database feature should be used for High availability.

Building Image

The process of building an image is below:

Figure 2-1 Build Image

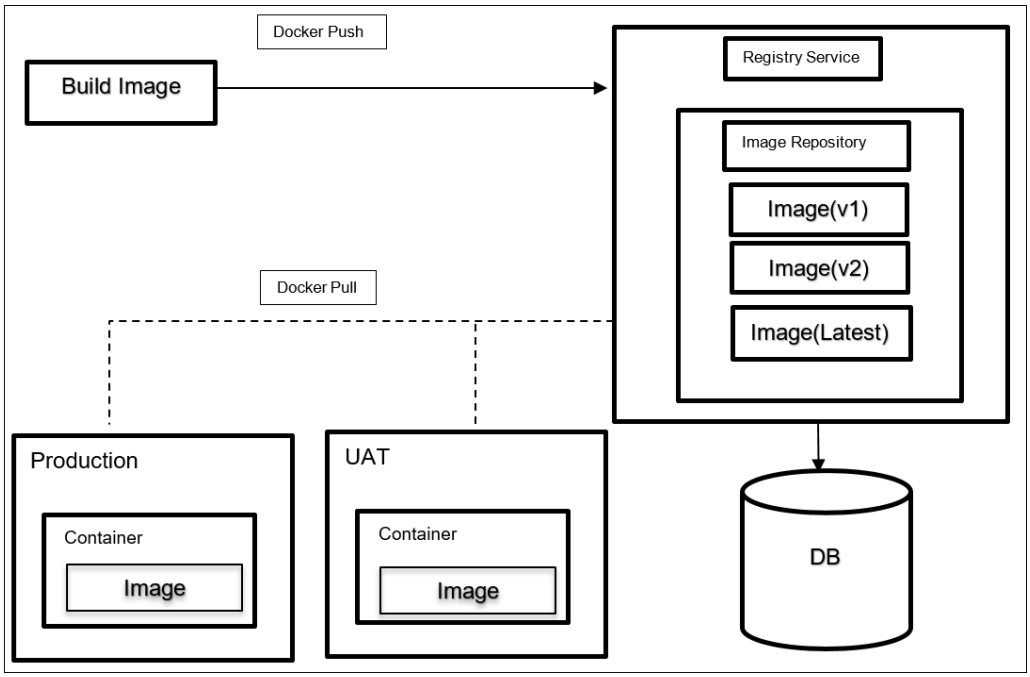
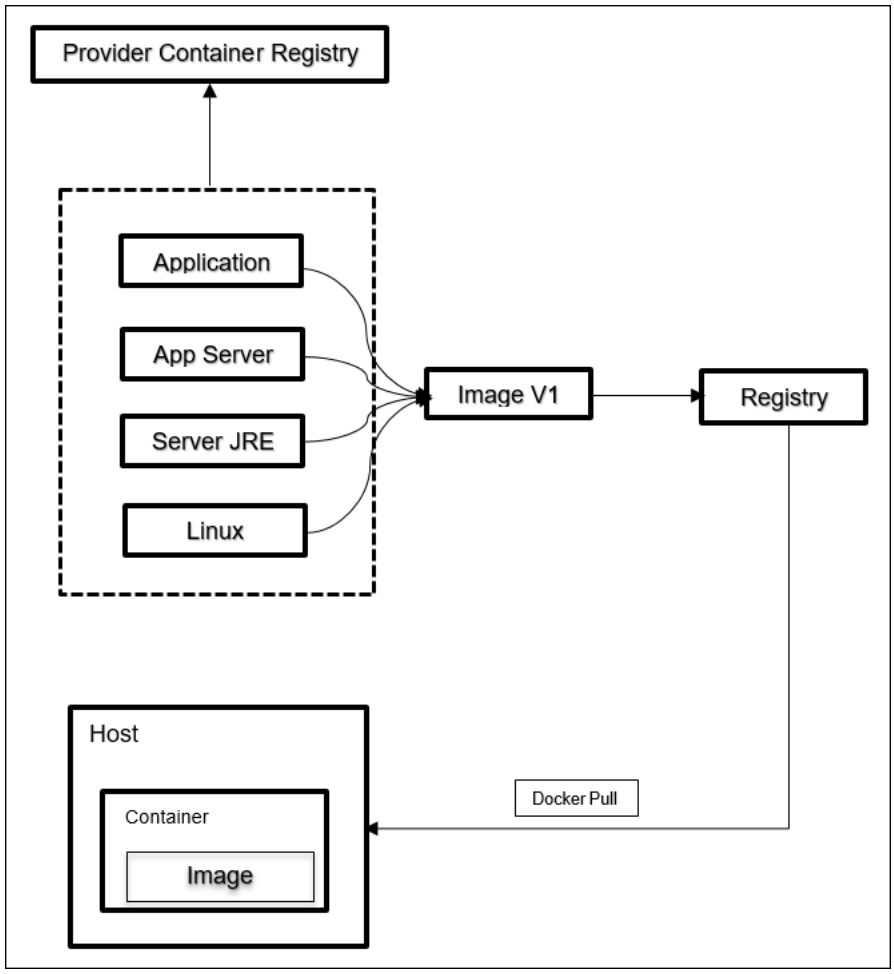


Image Creation

Image layered consists of the following components:

- Operating System Linux from provider Container Registry
- Java Runtime from provider Container Registry
- Application Server from provider Container Registry
- Applications from Oracle Banking Microservice Architecture product installer

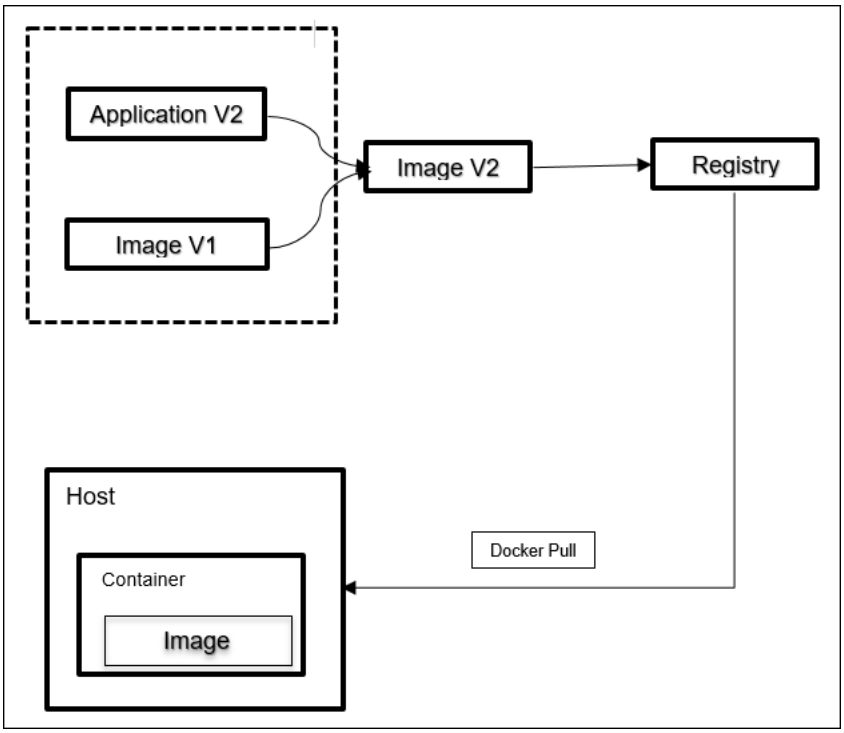
Figure 2-2 Image Components



Update Image for Patch/Customization
The modified image layered consists of the following components.

> **Note:**
>
> The image needs to be updated from local Container Registry.

Figure 2-3 Modified Image Components



3

Oracle Banking Microservices Architecture Products Deployment Approaches

This topic describes about the various approaches to deploy the Oracle Banking Microservices Architecture products.

- Containerization of the Services Using Tomcat
  This topic describes about the various methods that can be used to containerize a service in the Oracle Banking Microservices Architecture Product stack.
- Containerization of the Services Using WebLogic
  This topic describes about the various options to build the service containers using Oracle Weblogic images.
- Deploying Services without Docker Images
  This topic describes about the process to deploy product services without Docker images.
- Deploying Services on Private Cloud using Docker Images
  This topic describes about the process to deploy the services on private cloud using docker images.

3.1 Containerization of the Services Using Tomcat

This topic describes about the various methods that can be used to containerize a service in the Oracle Banking Microservices Architecture Product stack.

- Using Jib Plugin and Tomcat Image
  This topic provides the systematic instructions to create the Docker image using Jib Plugin and Tomcat image.
- Pipeline Integration in Jenkins
  This topic describes about the pipeline integration in Jenkins.
- Using War Artifacts Delivered in OSDC
  This topic describes about the usage of War Artifacts delivered in OSDC.

3.1.1 Using Jib Plugin and Tomcat Image

This topic provides the systematic instructions to create the Docker image using Jib Plugin and Tomcat image.

This approach uses the google Jib plugin to integrate the creation of Docker images along with gradle build. This approach can be used to create Docker images of components where the consulting or customer teams have access to code generated using the Oracle Banking Microservices Architecture Extensibility framework.

1. Update to `build.gradle` to include Jib plugin.

```
id 'com.google.cloud.tools.jib' version '2.6.0'
```

2. Add the Jib task in `build.gradle`.

```
jib {
   from {
     image = 'tomcat:<tag>'
   }
   to {
     auth{
       //it is ideal to use credHelper value here instead of
username/passwd if it is configured using the below line //
credHelper = '<credHelper_value>', else username/password to the //
registry can be used
      username = 'docker_registry_username'
      password = 'docker_registry_passwork'
     }
     image = <docker_registry_name>/<image_name:image_version>'
   }
   container {
      appRoot = '/usr/local/tomcat/webapps/ROOT'
   }
}

tasks {
   build {
      dependsOn(tasks.jib)
   }
}
```

3. Run the Gradle build using the command `$ gradlew clean build`.
4. Test the Docker image as follows:
   a. Login to Registry using Docker `login 'registry_name'`.
   b. Pull the image from repository using `docker pull <docker_registry_name>/<image_name:image_version>`.
   c. Run the docker image using `docker run -d -p 80:8080:<docker_registry_name>/<image_name:image_version>`.
   d. To pass `env` variable to your service to start, use the below options.
      - docker run -d -p <port> -v <Host Path>:/opt/logs/ <image> -e ds_jndi_1=jdbc/PLATO ds_db_host_1=<DBHOST> ds_db_port_1=<DB_HOST_PORT> ds_db_serviceid_1=<SID> ds_db_username_1=<USERNAME> ds_db_password_1=<PASSWORD>
      - docker run -d -v <Host Path>:/opt/logs/ --env-file <file> <image>

3.1.2 Pipeline Integration in Jenkins

This topic describes about the pipeline integration in Jenkins.

The Docker image creation using Jib plugin can be automated in the Continuous Integration pipeline. The Continuous Integration pipeline is used to run automated tasks that build a source code at preconfigure intervals to enable build automation for an application. The Jenkins Pipeline can be enhanced to support the automated deployment of Docker images in a Continuous Deployment pipeline.

Prerequisites

Before proceeding, make sure that the below installations and configurations are done.

- Jenkins installation
- Docker engine installation on Jenkins VM
- Network connectivity between docker registry and Jenkins VM
- Gradle plugin for Jenkins installation
- CredHelper Configuration

Automated Build – Continuous Integration

The gradel build step should be added as a stage in Jenkins to trigger the automated build for a service. This results in the image being created and pushed to the mentioned Docker registry.

```
stage('Build and Publish Docker Image') {
  steps {
    script {
      /* provide the Dockerfile and the context of the build which is the
directory which contains the Dockerfile */
      def image = docker.build( docker_image_name, "-f " + dockerfile_path +
"/Dockerfile " + dockerfile_path)
      /* once the image is complete, this runs the image and you can verify
if the image is correct by adding tests */
      image.inside {
        sh 'echo "Put Tests for your new image here"'
      }
      /* Replace the docker repo with your repo and the login with a cre-
dential in
      your Jenkins that has permission to push to your docker repo */
      docker.withRegistry('https://' + docker_registry + '/v2/',
docker_registry_login)
      {
      image.push(docker_image_version)
      }
    }
  }
}
```

Automated Deployment – Continuous Deployment

This topic describes about automated deployment for Continuous Deployment.

The codes used to auto-deploy an image on a VM with a vanilla Docker installation is detailed in the below Jenkins file.

```
  def docker_image_name = "<image_name:image_version>"
   def remote = [:]
   remote.name = "dkx"
   remote.host = "<docker_hostname>"
```

```
remote.allowAnyHosts = true
remote.user = "<username>"
remote.password = "<passwd>"

stage('Login to remote box') {
 steps {
   withCredentials([usernamePassword(credentialsId: 'sshUserAcct',
passwordVariable: 'password', usernameVariable: 'userName')]) {
     sshCommand remote: remote, command: 'docker pull
<docker_registry_name>/' + docker_image_name
     sshCommand remote: remote, command: 'docker run -d -p 80:8080
<docker_registry_name>/' + docker_image_name
      }
   }
}
```

3.1.3 Using War Artifacts Delivered in OSDC

This topic describes about the usage of War Artifacts delivered in OSDC.

This approach used if a consulting or partner installation team does not have access to the source code service and need to containerize product applications. This approach uses war files shipped under the Product Installer in the OSDC portal. This topic describes the individual steps to create docker images for each service.

Prerequisites

Before proceeding, make sure that the below steps are done.

- Make sure that the Docker Engine is up and running on the VM to perform the following operations.
- Make sure that the Proxy setting in `/etc/environment` file is updated using root permissions.

Create Docker file in Tomcat

Create a sample Docker file as follows:

1. Create a separate directory structure for each service.
   `$mkdir <service_name>/docker`

   `$cd <service_name>/docker`
2. Copy the service's war file from the installer to the path `<service_name>/docker`.
   `$cp <service_name>.war <service_name>/docker/`
3. Create a Docker file in the docker directory for the service.
   `$vi Dockerfile`

> **Note:**
>
> Services Dockerfile should have "tomcat:<tag_name>" as a base image.
> *FROM tomcat:<tag>*

4. Pass the appropriate build arguments to docker.

```
ARG application_context=<application context name>
ARG war_file_name=<microservices war file name>
ARG shutdown_port=<tomcat server shutdown port value>
ARG http_port=<tomcat server http port value>
ARG redirect_port=<tomcat server redirect port value>
ARG ajp_port=<tomcat server redirect port value>
```

application_context - This value represents the context root of the Plato application. This value must be passed as an argument in the docker file.
war_file_name - This value represents the name of the war file of the application that is present in the local system where the docker image is being built

shutdown_port - This value represents the shutdown port in the Tomcat server.

http_port - This value represents the HTTP port on which the application will be available for accessing via REST API.

redirect_port - This value represents the redirect port in the Tomcat server.

ajp_port - This value represents the AJP port in the Tomcat server.

> **Note:**
>
> The port values are not mandatory to give in case the docker image is getting built for deployment in Kubernetes but it is mandatory in case of docker-compose because the container port values should unique for the same. The port values are not passed/mentioned in the Docker file then the default will be used for while building the image.

5. Expose the container `http_port`
   `EXPOSE <http_port>`
6. The completed services's Docker file is shown below.

```
ARG application_context=plato-discovery-service
ARG war_file_name=plato-discovery-service-1.0.3.war
ARG shutdown_port=5008
ARG http_port=5005
ARG redirect_port=5007
ARG ajp_port=5006


COPY plato-discovery-service-1.0.3.war /usr/local/tomcat/webapps/

EXPOSE 5005

CMD["catalina.sh", "run"]
```

Test the Docker Image

Perform the following steps to test the Docker Image:

1. Run the Docker image using the below option:
   - `docker run -d -p 80:8080 <docker_registry_name>/<image_name:image_version>`

2. To pass *env* variable to your service use the below options:

   - `docker run -d -p <port> -v <Host Path>:/opt/logs/ <image> -e ds_jndi_1=jdbc/PLATO ds_db_host_1=<DBHOST> ds_db_port_1=<DB_HOST_PORT> ds_db_serviceid_1=<SID>`

     `ds_db_username_1=<USERNAME> ds_db_password_1=<PASSWORD>`

   - `docker run -d -v <Host Path>:/opt/logs/ --env-file <file> <image>`

3.2 Containerization of the Services Using WebLogic

This topic describes about the various options to build the service containers using Oracle Weblogic images.

- Using Pre-Built WebLogic Images
  This topic describes about the steps to deploy the services on a WebLogic Server running in a Docker container.
- Run WebLogic Containers Using WebLogic Kubernetes Operator
  This topic describes the process to run WebLogic containers using WebLogic Kubernetes operator.

3.2.1 Using Pre-Built WebLogic Images

This topic describes about the steps to deploy the services on a WebLogic Server running in a Docker container.

Prerequisites

Before proceeding, make sure that the below steps are completed.

- Make sure that the proxy settings is verified on the VM where Weblogic image need to run.
- Make sure that the user is logged in to the Oracle Container Registry portal, and accept the license agreements before pulling the Docker images.
- Sudo access to the VM to run commands as root.
  Create a file domain.properties with `username=""` and `password=""`.

Pull the WebLogic Docker Image

Run the following command:

```
docker pull container-registry.oracle.com/middleware/weblogic:12.2.1.4
```

Run the WebLogic Image

Run the following command:

```
docker run -d -p 7002:7001 -p 9004:9002 -v $PWD:/u01/oracle/properties
container-registry.oracle.com/middleware/weblogic:12.2.1.3
```

Deploy the Application

Access the console at `<hostname>:9004/console` with admin credentials and deploy the service.

Create Domains in WebLogic and Deploy the Applications

Deploy the application in custom domains. For information on deploying applications, refer to the below documentation:

https://github.com/oracle/docker-images/tree/main/OracleWebLogic/samples/12213-domain-home-in-image

3.2.2 Run WebLogic Containers Using WebLogic Kubernetes Operator

This topic describes the process to run WebLogic containers using WebLogic Kubernetes operator.

Prerequisites

Before proceeding, ensure that the below installation is done.

- Docker engine installation
- Kubernetes cluster
- Access to Kubernetes operator

Install and Manage WebLogic Domains using Kubernetes Operator

An operator is an application-specific controller that extends Kubernetes to create, configure, and manage instances of complex applications. The Oracle WebLogic Server Kubernetes Operator simplifies the management and operation of WebLogic domains and deployments.

For information on installation and management of weblogic domains, refer to https://oracle.github.io/weblogic-kubernetes-operator/.

3.3 Deploying Services without Docker Images

This topic describes about the process to deploy product services without Docker images.

- Deploying Applications to Tomcat without Docker Images
  This topic describes about the process to deploy the applications to Tomcat without docker images.
- Deploying Applications to WebLogic without Docker Images
  This topic describes about the process to deploy the applications to WebLogic without docker images.

3.3.1 Deploying Applications to Tomcat without Docker Images

This topic describes about the process to deploy the applications to Tomcat without docker images.

Prerequisites

Make sure that the below installation is done.

- Tomcat installation
- Jenkins installation

Manual deployment

1. Download and place the individual war files for services in a common directory.
2. Follow the steps in the below link to deploy the individual service wars.

   https://tomcat.apache.org/tomcat-10.0-doc/deployer-howto.html

Deployment using scripts

Alternatively, the war files can be configured to be deployed using a Jenkins pipeline. The deploy to container plugin should be used for configuration.

3.3.2 Deploying Applications to WebLogic without Docker Images

This topic describes about the process to deploy the applications to WebLogic without docker images.

Prerequisites

Make sure that the below installation is done.

- WebLogic installation
- Jenkins installation

Manual Deployment

Perform the following steps:

1. Download and place the individual war files for services in a common directory.
2. Follow the steps in the below link to deploy the individual service wars.

   https://docs.oracle.com/cd/E19424-01/820-4807/war-weblogic/index.html

Deployment using Jenkins

Alternatively, the war files can be configured to be deployed using a Jenkins pipeline. The Deploy WebLogic should be used for configuration. It is recommended to see if the version of the plugin has any vulnerabilities.

3.4 Deploying Services on Private Cloud using Docker Images

This topic describes about the process to deploy the services on private cloud using docker images.

When deploying the services on Docker image in the private cloud, it is important to build custom images of WebLogic and Tomcat using `openjdk 8` unless the appropriate license requirements are met with the built jdk versions in WebLogic.

For WebLogic, refer to https://github.com/oracle/docker-images/tree/main/OracleWebLogic and follow the steps to build the base WebLogic images.

For Tomcat, refer to https://hub.docker.com/_/tomcat and follow the steps to build the base tomcat images.

The pre-built OpenJDK 8 image is available in this link.

Index

A

Automated Build – Continuous Integration, *3-3*
Automated Deployment – Continuous Deployment, *3-3*

B

Building Image, *2-1*

C

Containerization, *2-1*
Containerization of the Services Using Tomcat, *3-1*
Containerization of the Services Using WebLogic, *3-6*

D

Database, *2-1*
Deploying Applications to Tomcat without Docker Images, *3-8*
Deploying Applications to WebLogic without Docker Images, *3-8*
Deploying Services on Private Cloud using Docker Images, *3-9*
Deploying Services without Docker Images, *3-7*
Docker, *1-1*
Docker Registry, *2-1*

I

Images and Containers, *1-1*
Install and Manage WebLogic Domains using Kubernetes Operator, *3-7*

K

Kubernetes (K8), *1-1*

O

Oracle Banking Microservices Architecture Products Deployment Approaches, *3-1*

P

Pipeline Integration in Jenkins, *3-2*
Pull the WebLogic Docker Image, *3-6*

R

Run WebLogic Containers Using WebLogic Kubernetes Operator, *3-7*

T

Technologies, *1-1*

U

Using Jib Plugin and Tomcat Image, *3-1*
Using Pre-Built WebLogic Images, *3-6*
Using War Artifacts Delivered in OSDC, *3-4*