Oracle® Cloud

Developing for Oracle Application Container Cloud Service

E64989-32
October 2018

Oracle Cloud Developing for Oracle Application Container Cloud Service,

E64989-32

Primary Authors: Rebecca Parks, Marilyn Beck, Rob Gray, Michael W. Williams

Contents

Preface

Topics:

- Audience
- Documentation Accessibility
- Related Resources
- Conventions

Audience

This guide is for developers who want to create new or modify existing applications so that they can be deployed on Oracle Application Container Cloud Service.

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.

Related Resources

See these Oracle resources:

- Oracle Public Cloud
- Java SE

  Java Standard Edition (SE) lets you develop and deploy Java applications to desktop and server environments. See Java Platform, Standard Edition (Java SE).
- Oracle Developer Cloud Service

  Oracle Developer Cloud Service is a cloud-based software development Platform as a Service (PaaS) and a hosted environment for your application development infrastructure. It provides an open source standards-based solution to develop, collaborate, and deploy applications within Oracle Cloud. When you subscribe to Oracle Application Container Cloud Service, you also get a free entitlement to Oracle Developer Cloud Service. You can use Oracle Developer Cloud Service to run application builds and then deploy to Oracle Application Container Cloud Service. See Deploying an Application to Oracle Application Container Cloud in *Using Oracle Developer Cloud Service*.

- Java Flight Recorder

  Java Flight Recorder (JFR) generates on-demand detailed recordings of the Java Virtual Machine (JVM) and the embedded application it's running. The recorded data includes an execution profile, garbage collection statistics, optimization decisions, object allocation, heap statistics, and latency events for locks and I/O. See *Java Flight Recorder Runtime Guide* in Java Components Documentation.

- Java Mission Control

  Java Mission Control (JMC) is a set of tools that runs on the Java Development Kit (JDK) and interacts with a JVM to deliver advanced, unobtrusive Java monitoring and management. See *Java Mission Control User's Guide* in Java Components Documentation.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Getting Started with Oracle Application Container Cloud Service

Learn how to design and package applications for Oracle Application Container Cloud Service.

Topics:

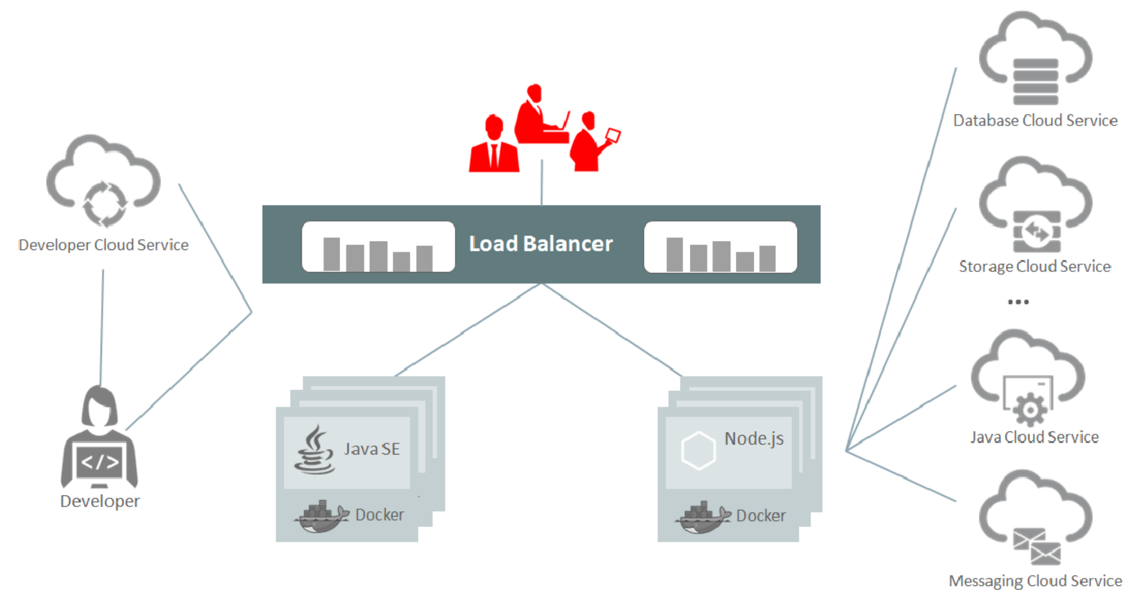
- About Your Application and Oracle Application Container Cloud Service
- Typical Workflow for Developing Applications
- Compare Oracle Cloud Services for Deploying Java Applications

About Your Application and Oracle Application Container Cloud Service

Oracle Application Container Cloud Service lets you deploy Java SE, Node.js, PHP, Python, Ruby, Go, and .NET Core applications to Oracle Cloud. You can also deploy Java EE web applications.

Subscribing to Oracle Application Container Cloud Service makes all languages available when you deploy your application.

Each application instance you deploy to Oracle Application Container Cloud Service runs in its own Docker container. A container is like a very lightweight virtual machine. Your application runs in its own isolated execution space, with its own memory, file system, and network access. Access to these operating system resources takes place without the cost of having to implement an entire virtual operating system.



You can use these key features of Oracle Application Container Cloud Service:

- An open, Docker based, ployglot, cloud native application development platform.
- A preconfigured environment for Java SE, Java EE, Node.js, PHP, Python, Ruby, Go, and .NET Core applications.
  - Support for Java Virtual Machine (JVM) based languages such as JRuby. Any language that uses the JVM can run on this service.
  - Java SE advanced features such as Java Flight Recorder, Java Mission Control, advanced memory management, and ongoing and timely security updates.
- Create, manage your applications using Web-based user interface, REST API, and PSM CLI.
- Bind you applications to the Oracle PaaS services like Oracle Database Cloud Service, Oracle MySQL Cloud Service, Oracle Event Hub Cloud Service, etc.
- Secure your applications using Oracle Identity Cloud Service.
- Cluster your applications to communicate privately, process background jobs with worker applications and enable caching for your services.
- You can incorporate DevOps, Continuos Integration, and Continuos Deployment (CI/CD) for your applications with Oracle Developer Cloud Service or continue using your existing CI/CD tools.
- Easy to use runtime upgrade and monitoring capabilities.
- Enterprise-grade technical support from Oracle.

In addition, Oracle Application Container Cloud Service is fully integrated with other Oracle Cloud services.

A subscription to Oracle Cloud Infrastructure Object Storage Classic is included and must be activated before you can deploy applications to Oracle Application Container Cloud Service.

Typical Workflow for Developing Applications

To manage the life cycle of Oracle Application Container Cloud Service applications, consider the typical workflow described in the following table..

Task	Description	More Information
Design your application	Make sure that your new or existing application meets a few simple design requirements for this service.	Design Considerations
Compile native libraries	If you have an application with native libraries, then run your build on Oracle Developer Cloud Service, or on your own system that runs Oracle Linux or a compatible distribution.	Compile Native Libraries for Your Application
Make a standalone application	In order to deploy your application to Oracle Application Container Cloud Service you must make your application self-contained. Your application must contains all the required dependencies and run independently.	Make a Standalone Application
Create the metadata files	You can define the configuration of your application by using the `manifest.json` and the`deployment.json` files. Use the descriptor files to specify information such as: • The launch command • How many instances of your application to deploy • How to connect to a database • Environment variables	Create Metadata Files
Package your application	To deploy your application, you compress the application files along with the dependencies and required configuration information in a .zip, .tgz, or .tar.gz archive file.	Create the Deployment-Ready Archive

Task	Description	More Information
Deploy your application to the service	You can deploy your application by using the following methods: • The user interface console • The REST API • The command-line interface (CLI) • From Oracle Developer Cloud Service	To deploy from the user interface, see Creating an Application in *Using Oracle Application Container Cloud Service*. To learn about the REST API, see *REST API for Managing Applications*. To deploy from Oracle Developer Cloud Service, see Deploying an Application to Oracle Application Container Cloud in *Using Oracle Developer Cloud Service*.
Monitor your application (only Java applications)	If you have a Java application, then you can monitor it using Java Mission Control and Java Flight Recorder.	See Java Mission Control and Java Flight Recorder.
Get the application's logs	After the application is deployed you can retrieve the application's logs using the Oracle Application Container Cloud Service console, the command-line interface, or the REST API.	Retrieve the Application Logs

Task	Description	More Information
Manage your application	As your application is running, you can: • Change the number of instances. • Alter the amount of memory allocated to each instance. • Download and review application logs. • Upload a new version of the application. • Upload a new `manifest.json` file or `deployment.json` file. • Add or change the values of environment variables. • Update service bindings.	To learn more about the REST API, see *REST API for Managing Applications*. To learn more about the user interface, see Using the Applications Page and Using the Application Console in *Using Oracle Application Container Cloud Service*.

Compare Oracle Cloud Services for Deploying Java Applications

Choose an Oracle Cloud service that best meets the needs of your Java application and development process.

Oracle offers two main cloud services that support Java deployments: Oracle Java Cloud Service and Oracle Application Container Cloud Service. In general, Oracle Java Cloud Service provides a Java solution that is more flexible and customizable, while Oracle Application Container Cloud Service offers a simpler, automated and managed solution for Java applications.

Both services share common capabilities:

- Host your application in a highly-available environment
- Easily scale your application in response to changing capacity requirements
- Cache and retrieve frequently-used data
- Automate deployment though REST APIs, CLI commands, or Oracle Developer Cloud Service

There are important differences between the services:

- Oracle Application Container Cloud Service supports Java Standard Edition applications and Java Enterprise Edition web applications (WAR). Oracle Java Cloud Service supports the full Java EE specification, including enterprise applications (EAR) and Java Message Service (JMS).

- With Oracle Application Container Cloud Service, you can deploy applications that are developed in a variety of languages, including Java, PHP, Python, and Ruby.
- Oracle Application Container Cloud Service cannot be deployed to Oracle Cloud Infrastructure regions. Oracle Java Cloud Service supports both Oracle Cloud Infrastructure and Oracle Cloud Infrastructure Classic regions.
- Oracle Java Cloud Service gives administrators access to Oracle WebLogic Server and the operating system. Oracle Application Container Cloud Service hides this infrastructure from users, and automatically keeps it up-to-date with the latest software and patches.
- When you create an Oracle Java Cloud Service instance, you choose from a list of specific Oracle WebLogic Server releases, including older ones like 11g. With Oracle Application Container Cloud Service, you don't have to worry about the details of the container.
- Deploying your code to Oracle Application Container Cloud Service is fast and easy, but Oracle Java Cloud Service also integrates with popular Integrated Development Environments (IDE).
- Oracle Java Cloud Service offers tools to automate the migration of existing Oracle WebLogic Server environments to the cloud.
- With Oracle Application Container Cloud Service, you can quickly integrate your Java application with other Oracle Cloud resources like databases and message queues. Oracle Java Cloud Service does not offer a similar data binding feature, but does provide out-of-the-box integration with Oracle Database Cloud Service.

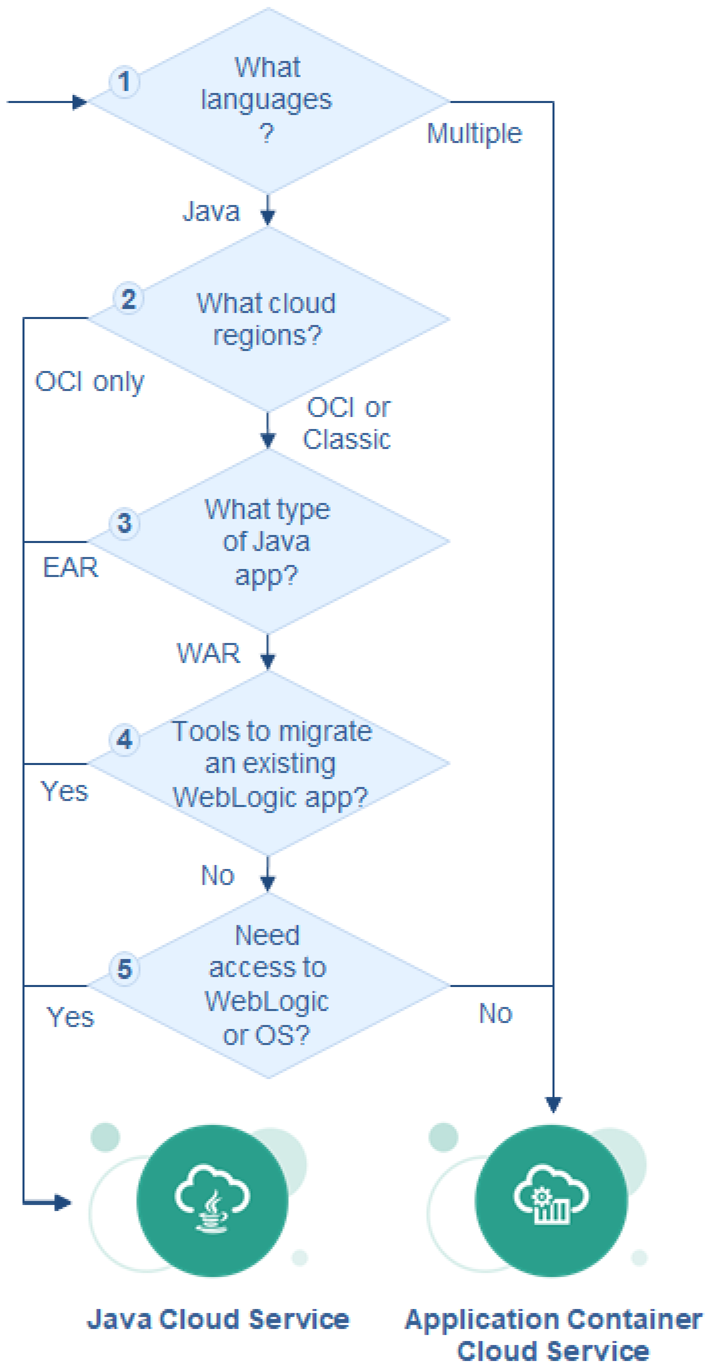
If neither of these services meets your exact requirements, you can create basic compute instances or containers in Oracle Cloud:

- Oracle Cloud Infrastructure Compute
- Oracle Cloud Infrastructure Compute Classic
- Oracle Cloud Infrastructure Container Service Classic
- Oracle Container Engine for Kubernetes
- Oracle Weblogic Server Kubernetes Operator

These infrastructure cloud solutions give you the most flexibility, but you must install, configure, and maintain all of the Java software components.

Decision Tree

Answer the following series of questions to help you choose between Oracle Java Cloud Service and Oracle Application Container Cloud Service.



1. In which language(s) is your application written?

   If the components of your application are written in multiple languages, then use Oracle Application Container Cloud Service.

2. Which regions are available in your Oracle Cloud account?

   If your account has access to Oracle Cloud Infrastructure regions only, then use Oracle Java Cloud Service. Oracle Cloud Infrastructure regions include `us-phoenix-1`, `us-ashburn-1`, `eu-frankfurt-1`, and `uk-london-1`.

3. What type of Java EE application are you developing or migrating?

If your application is packaged as an Enterprise Application (EAR), then use Oracle Java Cloud Service.

4. Are you migrating an existing Oracle WebLogic Server application? Would you prefer tools to help automate the migration of your applications and supporting resources?

   If your answer is yes, then use Oracle Java Cloud Service.

5. Do you require administrative access to Oracle WebLogic Server or the operating system, in order to customize the default configuration?

   If your answer is yes, then use Oracle Java Cloud Service.

   If your answer is no, then use Oracle Application Container Cloud Service.

2

Creating Your Application

When creating your application for deployment on Oracle Application Container Cloud Service, you must make sure it listens on the correct port, is configurable at runtime, and includes all dependent classes, including library classes.

Topics:

Design Considerations

If you're developing a new application or deploying an existing one to run on Oracle Application Container Cloud Service, keep these requirements in mind.

- **Applications must be configurable at runtime.** Most application types must listen on the port provided in the `PORT` environment variable. Applications can also read user-defined environment variables and environment variables from other services. See Make the Application Configurable at Runtime.
- **Applications must include all dependencies.** If your application requires a library to execute, that library must be included in the application when it is deployed. See Make a Standalone Application.

If your application needs to maintain or share state, use Oracle Cloud Infrastructure Object Storage Classic or Oracle Database Cloud Service for storing data. To use Oracle Database Cloud Service, you must configure a service binding, which you can do in the user interface or the `deployment.json` file. See Create Metadata Files.

Other than these requirements, the application and the command that launches it are entirely under your control.

Make the Application Configurable at Runtime

Your application must be able to read settings from environment variables in the application's container. All applications except Java EE web applications and worker applications must read the `HOSTNAME` and `PORT` environment variables, and use the values dynamically.

Up to three types of environment variables are available to all instances of your application:

1. Your application is running inside a Docker container that has a generated host name and port. These are made available to the application in the `HOSTNAME` and `PORT` environment variables.

   If the application is required to listen on the specified port but doesn't, then the application creation and deployment will fail. After deployment, the service pings the application on that port to determine if it's running. The load balancer and application ports are different. The load balancer accepts SSL traffic on port 1443, then directs requests to each application according to the port in the `PORT` environment variable.
2. If your application uses other Oracle Cloud services, then service connection details (such as ports) can also be made available in environment variables.
3. You can also add your own environment variables using the user interface or the `deployment.json` file. See Create Metadata Files.

Note:

The `PORT` and `ORA_PORT` environment variables have the same value. Your application can read the port using either one.

If you're programming in Java 8, then you can use the `Optional` class to retrieve the environment variables without having to use `if` blocks to check for null values, as shown in this code snippet from the Grizzly Jersey sample application:

```
/**
 * Main class
 */
public class Main{

    // Base URI the Grizzly HTTP server will listen on
    public static final String BASE_URI;
    public static final String protocol;
    public static final Optional<String> host;
    public static final String path;
    public static final Optional<String> port;

    static{
      protocol = "http://";
      host = Optional.ofNullable(System.getenv("HOSTNAME"));
      port = Optional.ofNullable(System.getenv("PORT"));
      path = "myapp";
      BASE_URI = protocol + host.orElse("localhost") + ":" + port.orElse("8080") +
"/" + path + "/";
```

```
    }
}
```

Service to Service Communication

Service to service communication is crucial when you build microservices. A microservice-based application typically runs on a single process, and they can interact with each other using a communication protocol, for example, HTTP or TCP. Oracle Application Container Cloud Service allows you to interconnect your applications using different methods depending on the type of application.

Oracle Application Container Cloud Service supports two types of applications:

- Web Applications
- Worker Applications

Web Applications

A web application is a public application that you can access with a public URL. By default, all applications on Oracle Application Container Cloud Service are web applications.

Web applications can be invoked from another web or worker application by using one of the two options:

- Using the public URL. For example, a REST API exposed by application 'appB' can be invoked by another service 'appA' by making use of the public URL.
- Using the internal overlay network. Applications can communicate with each other over a secure internal network if all of the applications are marked as`isClustered` . The application name is used as the host name:
  - If the application binds to the `\PORT` environment variable., then it can be accessed on port 8080 over HTTP. The URL format is `http://<application_name>:8080`.
  - If an application binds to a custom port (for example, 9090) in addition to the `PORT` environment variable and a different protocol, then it can be accessed by using that port and protocol over which it has exposed its service (for example, TCP or HTTP). The URL format is `http://<application_name>:<custom_port>`.

Worker Applications

A worker application is a private application that doesn’t have a public URL.

Worker applications can only be invoked from another web or worker application by using an internal overlay network. They can talk to each other over a secure internal network if all of them are marked as `isClustered`.

To invoke a worker application, the application name is used as the host name and it can be accessed by using the port and the protocol where the service is exposed. For example, a REST API over 8082 or a TCP service over 9090. The URL format is `http://<application_name>:<port>`.

Communication Patterns

A combination of the different application types and clustering capability leads to following access communication patterns:

Access Pattern	Description	Public Network Access (Internet)	Internal (overlay) Network Access (Clustered)
web-web	A web application that invokes another web application.	Yes	Yes
web-worker	A web application that invokes a worker application.	No	Yes
worker-web	A worker application that invokes a web application.	Yes	Yes
worker-worker	A worker application that invokes another worker application	No	Yes

Tutorial - Invoke a worker application from a web application

For more information, see Design Considerations and Clustered Applications.

Compile Native Libraries for Your Application

When you deploy your application, it runs in a Docker container, which comes from an Oracle or Docker Hub image source. If your application has native libraries, those libraries must be compiled on a system that runs on Oracle Linux or a compatible distribution before the application can be deployed.

If your Node.js application only includes JavaScript `.js` files and no native libraries, or if your Java application uses no native libraries, then you can skip this section.

You can build the libraries on your own system if it runs Oracle Linux or a compatible distribution. You can also build your libraries on Oracle Developer Cloud Service. An entitlement for it is included with your subscription. For details about builds, see Managing Project Jobs and Builds in Oracle Developer Cloud Service in *Using Oracle Developer Cloud Service*.

You don't need to compile the Oracle `node-oracledb` driver, it shouldn't be included in the local `node_modules` folder that is included in the application archive. The driver is provided in the Oracle Application Container Cloud Service Node Docker image.

To learn more about the Docker image sources, see Docker Images.

Service Bindings

A service binding provides seamless communication over a private overlay network within your identity domain between Oracle Application Container Cloud Service and another Oracle Cloud service. Also a service binding coordinates to the bound service using environment variables.

A service binding is not necessary to connect to an Oracle Public Cloud service that provides a publicly accessible endpoint (such as Oracle Database Exadata Express Cloud Service) or a REST API (such as Oracle Cloud Infrastructure Object Storage Classic or Oracle Messaging Cloud Service).

In Oracle Application Container Cloud Service, you can add service bindings to other subscribed Oracle Cloud services from the Deployments page of the Application Console or configure them using the `deployment.json` file. To learn more about how to define service bindings in the `deployment.json` file, see Creating Metadata Files.

Clustered Applications

Clustering is a property that can be added to your application. To do this, add `isClustered: true` to the application's `manifest.json` file, for example:

Example 2-1 Enable clustering capability in your application

```
{
  "runtime": {
      "majorVersion": "8"
  },
  "command": "java -jar myapp.jar",
  "isClustered" : true
}
```

Setting this property places the application on a shared overlay network. As a result, applications can use this networking support to communicate directly over a private IP network. This enables applications to collaborate and provide services to each other, and also makes it possible to expose APIs that are accessible only to other applications and not to external clients.

Some of the clustering benefits include:

- Improved performance: Applications don't need to route request through the public internet and through the load balancer.
- Flexibility: Applications can use any port and any protocol to communicate privately among themselves on the internal overlay network.

See Prepare a Clustered Application for Deployment and Service to Service Communication.

Worker Applications

You can deploy a worker application to Oracle Application Container Cloud Service. Applications deployed within your identity domain can access a worker application over a private overlay network. A worker application can't be accessed by end users.

You can't access a worker application using a public URL, a public REST API, or the PaaS Service Manager command-line interface. The worker application doesn't need to read the `PORT` environment variable, although it can. Similar to a public application, a worker application can use environment variables and service bindings, and it can be scaled and monitored.

To specify a worker application, set the following parameters in the `manifest.json` file:

- `"type":"worker"` – (Required) Specifies a worker application.
- `"isClustered":"true"` – (Optional) Specifies that the application is clustered, which is often necessary for a worker application to communicate with a public application.

You can’t change a worker application to a public application by redeploying it. You must delete the application, make the change, and then deploy it as a new application. For more information about how to deploy a worker application, see Deploying an Application.

Docker Images

When you deploy your application, it runs in a Docker container based on a Docker image that´s specified by your application. Docker images are available. Some are Oracle curated images, and others come from Docker Hub.

These are the available images for a Docker container.

Image	Source	Operating System
Java SE	Oracle	Oracle Linux
Java EE	Oracle	Oracle Linux
Node.js	Oracle	Oracle Linux
PHP	Oracle	Oracle Linux
Python	Docker Hub	Debian
Ruby	Docker Hub	Debian
Go	Docker Hub	Debian
.Net	Docker Hub	Debian

Note:

Oracle Application Container Cloud Service automatically gets the latest minor releases for the containers from Docker Hub.

Install Linux Packages for Your Application

To make your application work, install additional packages in a Java SE, PHP, Node.js or Java EE application container. The packages install at runtime from the Oracle Linux 7 YUM repository.

1. Create the linux-packages.txt text file in a text editor.
2. Add one of the available installation options on a new line in the file. The following table describes the install options:

Option	Description	Format	Example
package_install	Install an individual package; it is equivalent to the “yum install -y pkg-name” Linux command.	package_install:package_name	package_install:lynx-2.8.6-27.el6.x86_64
group_install	Install a set of related packages in the specified group, it is equivalent to the “yum -y groupinstall group_name” Linux command.	group_install:group_name	group_install:x11

3. Repeat Step 2 as needed.
4. Save and close the file.
5. Add the file to the root of the compressed file that you use to package your application. See Packaging Your Application.
6. Deploy your application. See Creating the Deployment-Ready Archive.
7. In the recent application logs, search for "Summary of Package Installation" and review if the packages installed successfully. See Exploring the Application Administration Page in *Using Oracle Application Container Cloud Service*.

Example 2-2 linux-packages.txt

```
package_install:make-3.82-23.el7.x86_64
package_install:gcc-4.8.5-4.el7.x86_64
package_install:lynx-2.8.6-27.el6.x86_64
group_install:x11
```

Caching Capability for Your Application

Oracle Application Container Cloud Service features clustered, scalable, in-memory caching with data backup. Data is replicated among cluster members in the cache service to avoid data lost in case of a member failure. If a member fails, data is redistributed among the remaining members to ensure resiliency.

Common use cases for caches are:

- Reducing how often a data source is accessed, which results in better performance and scalability for applications.
- Sharing state information among multiple applications, which can be of different types.

When you add an application cache to your application the clustering capability is enable automatically in your application, that means you don’t have to specify the `isClustered:true` parameter in the `manifest.json` file but is added automatically by Oracle Application Container Cloud Service.

Tutorials - Create Applications Using a Cache Application Learning Series

You can create service bindings to caches in Oracle Application Container Cloud Service. See Typical Workflow for Creating and Using Caches in *Using Caches in Oracle Application Container Cloud Service*.

Secure Applications

Oracle Application Container Cloud Service can use Oracle Identity Cloud Service to authenticate administrators and application users. During deployment, you can create a security application in Oracle Identity Cloud Service to control who can access your application in Oracle Application Container Cloud Service.

When you deploy a Java SE 7 or 8, Node.js, or PHP application in Oracle Application Container Cloud Service, you can secure your application using Oracle Identity Cloud Service with one of these types of authentication:

- Basic — Prompts for a username and password set up in Oracle Identity Cloud Service.

- OAuth — Creates a corresponding application in Oracle Identity Cloud Service to control who can access your application, and redirects to Oracle Identity Cloud Service for authentication.

If you are using the REST API, setting the `authType` form parameter to `basic` or `oauth` is equivalent to selecting Basic or OAuth in the web interface. See Create an Application in *REST API for Managing Applications*.

Tutorial - Secure an application with Oracle Identity Cloud Service

See Using Oracle Identity Cloud Service with Oracle Application Container Cloud Service in *Using Oracle Application Container Cloud Service*.

Network File System

The Network File System (NFS) is a distributed file system that allows a client computer to access files over a network as though the files were on local storage. Oracle Application Container Cloud Service supports mounting of NFS volumes into the application containers running on Oracle Cloud Infrastructure Classic accounts.

To specify the NFS volumes, set the following parameters in the `deployment.json` file:

- **volumes** – Represents the list of volumes. This element can contain more than one block of sub-elements. Each block specifies one volume.
  - **name** – (required) Mounted volume name.
  - **type** – (required) Type of the volume. Currently only `nfs` is supported.
  - **device** – (required) Defines the IP address and the path of the volume. For example, `x.x.x.x:/shares/nfs1`
  - **mount_options** – (optional for Oracle Cloud Infrastructure Classic) If the `mount_options` parameter isn't specified, it takes the default values. If it's present all sub-elements must be specified.
    - **vers=`<value>`** – Oracle Application Container Cloud Service supports NFS `4` for Oracle Cloud Infrastructure Classic accounts. The default value is `4`.
    - **soft/hard** – Specifies whether the program using a file via an NFS connection should stop and wait (hard) for the server to come back online, if the host serving the exported file system is unavailable, or if it should report an error (soft). The default value is `soft`.
    - **timeo=`<value>`** – Specifies the number of seconds to pass before the error is reported.
    - **bg** – This is known as a background mount. It does the mount in the background if it fails the first time and thus lets the system continue booting even if there are NFS problems. With this option, a time out or failure causes the `mount(8)` command to fork a child which continues to attempt to mount the export. The parent immediately returns with a zero exit code.
    - **tcp** – Specifies to use the TCP protocol.
    - **rw**– Specifies read and write access.

Example 2-3 deployment.json

```
{
    "memory": "1G",
    "instances": 1,
    "environment": {},
    "secureEnvironment": [],
    "system_properties": {},
    "services": [],
    "volumes": [{
            "name": "vol1",
            "type": "nfs",
            "device": "100.100.64.47:/fss",
            "mount_options": ["vers=4", "soft", "timeo=180", "bg", "tcp", "rw"]
        }
    ]
}
```

It's customer's responsibility to make sure the NFS server is accessible from the Oracle Application Container Cloud Service application.

Select a Load Balancer Policy

Load Balancing provides automated traffic distribution from one entry point to multiple servers. A load balancer improves resource utilization, facilitates scaling, and helps ensure high availability. Oracle Application Container Cloud Service supports two primary policy types:

- **Round Robin.** Distributes incoming traffic sequentially to each server in a backend set list. After each server has received a connection, the load balancer repeats the list in the same order. Round Robin s the default load balancer policy.
- **IP Hash.** Uses an incoming request's source IP address as a hashing key to route non-sticky traffic to the same backend server. The load balancer routes requests from the same client to the same backend server as long as that server is available. IP Hash ensures that requests from a particular client are always directed to the same backend server, as long as it is available.

Note:

The load balancing policy is set during the creating of the application and it can’t be updated after the application is created.

To specify the load balancing policy in your application set the `loadBalancingPolicy` parameter in the `manifest.json` file. The `loadBalancingPolicy` parameter accepts two values: `ROUND_ROBIN` and `IP_HASH`.

Example 2-4 Round Robin Load Balancing Policy

```
{
  "runtime": {
      "majorVersion": "8"
  },
  "command": "java -jar myapp.jar",
  "loadBalancingPolicy" : "ROUND_ROBIN"
}
```

Example 2-5 IP Hash Load Balancing Policy

```
{
  "runtime": {
      "majorVersion": "8"
  },
  "command": "java -jar myapp.jar",
  "loadBalancingPolicy" : "IP_HASH"
}
```

Tutorial - Select a Load Balancing Policy

3
Packaging Your Application

After your application has been tested locally, create an archive file (.zip, .tgz, .tar.gz) that includes the application and any dependent libraries. You must also create a `manifest.json` file if you need to specify a launch command or other parameters.

When you have your archive, you can upload it, plus an optional `deployment.json` file, using the user interface or the REST API.

See Typical Workflow for Packaging Process to get started, then see other sections that apply to your application type.

Topics:

Typical Workflow for Packaging Process

Depending on your application, you might need to perform most or all of these tasks to prepare it for deployment to Oracle Application Container Cloud Service.

Task	Description	More Information
Make your application self-contained	The application must include everything it needs to run independently. For Java applications, this means all referenced classes must be included.	Make a Standalone Application

Task	Description	More Information
Select the launch command	For PHP applications, a launch command is optional: By default a PHP application is launched from its index file. For Java EE web applications, a launch command is unnecessary. For other application types, you must specify the launch command. Depending on your application, this could be a Java, JavaScript, or shell command.	Select the Launch Command
Create the metadata files	Applications that require a launch command must include a `manifest.json` file. This file specifies information that Oracle Application Container Cloud Service requires to run your application properly. Optionally, you can also specify additional information about instance scaling, environment variables, and service bindings in a `deployment.json` file.	Create Metadata Files
Archive your application	Applications must be archived in a `.zip`, `.tgz`, or `tar.gz` file with the `manifest.json` file at the root if present. This ensures that Oracle Application Container Cloud Service can find the `manifest.json` file. You don't need to include a Java EE web application in a .zip, .tgz, or .tar.gz file unless you need to set values in the `manifest.json` file, which is optional.	Create the Deployment-Ready Archive

Make a Standalone Application

After you develop your application, you need to decide how to include or reference any dependent libraries. See the details for each language.

Java

If your Java application doesn't depend on library classes, then you can create a JAR file. By default, a JAR file includes only the class files generated from the source files.

If your Java application depends on library classes, then your package must include them when it's deployed. You can accomplish this in one of two ways:

- **Create an uber JAR.** When create your application, include all dependent libraries in the JAR file with the application. If you're using Maven, then you can use the Maven build tool and either the assembly or shade plug-in to copy the required libraries into the JAR file. Instructions for creating an uber JAR file with the assembly plug-in are in this tutorial: Creating a Basic REST Web Service using Grizzly, Jersey, and Maven.
- **Use the classpath.** All dependent libraries are included in separate JAR files, but the path to each file is included in the `-classpath` option on the command line. If you include a lot of libraries in your application, then the classpath can get long. If that's the case, then you can put the command line into a Bash shell script and execute that. If you're using Maven, then you can use the appassembler plugin to write the Bash script.

Java EE

Your web application must include all dependent classes that aren't included in WebLogic Server. To deploy your application, you can create either a single `.war` file or a `.zip, .tar`, or `.tar.gz` file with an optional `manifest.json` file and a single `.war` file at the root.

Node

If your application doesn't use any third-party libraries or have any NPM dependencies, then you can compress your project files the code and the `manifest.json` file. If your application has dependencies, then your application and its dependencies must be bundled in a `.zip, .tar`, or `.tar.gz` file. You must install the libraries in a local directory (`node_module`) and then compress your code, the `manifest.json`, the `package.json` files, and the `node_module` directory.

PHP

Compress your project in a `.zip, .tar`, or `.tar.gz` file that contains your code, the `manifest.json`, script files, and the dependencies (if any).

Python

If your application doesn't depend on any third-party libraries, then compress your project in a `.zip, .tar`, or `.tar.gz` file that contains the `.py, manifest.json`, and script files. If your application uses third-party libraries, then you must first install the libraries in a local directory, and then compress your project files along with the `manifest.json` file, scrip files, and the directory with the dependencies.

Example 3-1 Sample start.sh Script for a Python Application

```
#!/bin/sh
export PYTHONPATH=modules
python app.py
```

Ruby

If your application doesn't depend on third-party libraries, then it must be compressed in a `.zip, .tar`, or `.tar.gz` file that contains the `.rb, manifest.json`, and script files. If your application has dependencies, then you must specify them in the `Gemfile` file and create a script to install the dependencies specified in the `Gemfile` file.

After you do that, you can compress your project (`.rb` and script files, and the dependencies directory) and the `manifest.json` file.

Example 3-2 Sample start.sh Script for a Ruby Application

```
#Install the dependencies specified in the Gemfile
bundle install
#Run the database migration to create the Employee table.
bundle exec rake db:migrate
#Run the Sinatra application. Your application must run on the port specified in the
PORT environment variable and in the 0.0.0.0 host.
rackup -p ${PORT} --host 0.0.0.0
```

Go

If your application has dependencies, then you must specify them in a script file and Oracle Application Container Cloud Service manages them when you deploy your application.

Compress your project in a `.zip`, `.tar`, or `.tar.gz` file that contains your code, the `manifest.json` file and script files.

Example 3-3 Sample start.sh Script for a Go Application

```
# Extract LIBAOI libs from Debian package (into ./lib/x86_64-linux-gnu)
dpkg-deb -R libaio1_0.3.110-1_amd64.deb ${APP_HOME}
export PKG_CONFIG_PATH=${APP_HOME}/Oracle/instantclient_12_2

# Add OCI and LIBAIO to shared library path
export LD_LIBRARY_PATH=${APP_HOME}/Oracle/instantclient_12_2:${APP_HOME}/lib/x86_64-
linux-gnu

# Finalize OCI installation by creating required softlink
ln -s -f ${APP_HOME}/Oracle/instantclient_12_2/libclntsh.so.12.1 ${APP_HOME}/Oracle/
instantclient_12_2/libclntsh.so
ln -s -f ${APP_HOME}/Oracle/instantclient_12_2/libocci.so.12.1 ${APP_HOME}/Oracle/
instantclient_12_2/libocci.so

# Install Go dependencies
go get github.com/mattn/go-oci8
go get github.com/ant0ine/go-json-rest/rest

# Launch the application
go run oracle-db.go
```

.Net

To make your standalone application you need to update the project's dependencies and tools:

```
dotnet restore
```

Create a debug build of your application:

```
dotnet build
```

Create a self-contained deployment for the Linux platform:

```
dotnet publish -c Release -r linux-x64
```

Compress your project in a `.zip`, `.tar`, or `.tar.gz` file that contains your code, the `manifest.json` file, and the `publish` directory.

Select the Launch Command

You have total control over how you launch an application. You can launch directly by invoking the specific language command or use a shell script. The application is executed in a Linux container, so most of the rules that apply to running a command in Linux apply.

For Java, Node, PHP, Python, Ruby, Go, and .Net applications you can specify the launch command in the `manifest.json` metadata file that you include with the application. See Creating the manifest.json File.

For more details, see the launch command for each language.

Java

The examples that follow are based on the following assumptions:

- The home directory for your application is stored in the `$APP_HOME` environment variable.
- Your application execution code is stored in a JAR file named `app.jar`, which is located in `$APP_HOME`.
- All required Java libraries are stored in the `lib` directory. The `lib` directory is included as part of the final archive as a subdirectory from the archive root directory.
- The `lib` directory contains the following JAR libraries: `web.jar`, `rest.jar`, `media.jar`.

If the `lib` directory isn't included in the application JAR file, then you must specify the classpath so the JVM can find all the classes necessary to run the application.

Example 3-4   Setting the Classpath in the Manifest.mf File

Given these assumptions, this java command could launch your application.

```
java -jar app.jar
```

The JAR must include a `Manifest.mf` file with the `Main-Class` attribute set to the main class and the `Class-Path` attribute set to the `lib/web.jar lib/rest.jar lib/media.jar` directory.

Example 3-5   Setting the Classpath Using the -cp Option

You can use the `-classpath` or `-cp` option to specify the location of the main application JAR and the dependent JAR files. This option is incompatible with the `-jar` option, so you execute the main class directly. For example:

```
java -cp $APP_HOME/app.jar:$APP_HOME/lib/* com.example.Main
```

Note that on Linux, the path separator is a colon, not a semicolon as on Windows.

Example 3-6   Using an Uber JAR

If the application is packaged as an uber JAR called `uber-app.jar`, with all the dependencies available in the JAR, then an external classpath isn't needed. The uber JAR must include a `Manifest.mf` with the main class. To run the application:

```
java -jar uber-app.jar
```

Java EE

For a Java EE applications, a launch command is unnecessary. If it is present, then it is ignored.

Node

Configure the launch command in the manifest.json file, for example:

Example 3-7 Sample manifest.json File for a Node Application

```
{
  "runtime":{
    "majorVersion":"4"
  },
  "command": "node server.js",
  "release": {},
  "notes": ""
}
```

PHP

A PHP application typically doesn't need a launch command. Unless another file is specified in the application URL, the index file opens first, whether the extension is `.htm`, `.html`, or `.php`. If you use a launch command because you need to run a script before starting your application, the last command in the script must be `apache2-run`.

PHP depends on the Apache HTTP server and Oracle Application Container Cloud Service doesn't start your application automatically if a launch command is present.

Python

Configure the launch command in the manifest.json file, for example:

Example 3-8 Sample manifest.json File for a Python Application

```
{
  "runtime": {
    "majorVersion": "3.6.0"
  },
  "command": "python app.py",
  "notes": "Simple REST Service"
}
```

Ruby

Configure the launch command in the manifest.json file, for example:

Example 3-9 Sample manifest.json File for a Ruby Application

```
{
  "runtime":{
     "majorVersion":"2.4.1"
},
  "command": "ruby app.rb",
  "mode": "rolling"
}
```

Go

Configure the launch command in the manifest.json file, for example:

Example 3-10   Sample manifest.json File for a Go Application

```
{
  "runtime":{
     "majorVersion":"1.8.3"
},
  "command": "go run app.go",
  "mode": "rolling"
}
```

.Net

Configure the launch command in the manifest.json file, for example:

Example 3-11   Sample manifest.json File for a .Net Application

```
{
  "runtime":{
     "majorVersion":"2.0.0-runtime"
    },
    "command": "dotnet publish/sample-app.dll"
}
```

Executing the Application with a Shell Script

As an alternative, you can execute your application using a shell script.

Example 3-12   Sample start.sh Script for a Python Application

```
#!/bin/sh

# Define PYTHONPATH as local modules folder
export PYTHONPATH=${APP_HOME}/modules

# Extract LIBAOI libs from Debian package (into ./lib/x86_64-linux-gnu)
dpkg-deb -R libaio1_0.3.110-1_amd64.deb ${APP_HOME}

# Finalize OCI installation by creating required softlink
ln -s ${APP_HOME}/lib/instantclient_12_2/libclntsh.so.12.1 ${APP_HOME}/lib/
instantclient_12_2/libclntsh.so

# Add OCI and LIBAIO to shared library path
export LD_LIBRARY_PATH=${APP_HOME}/lib/instantclient_12_2:${APP_HOME}/lib/x86_64-
linux-gnu

# Install Python packages into local modules folder
pip --no-cache-dir install -r requirements.txt -t ${PYTHONPATH} --upgrade

python ${APP_HOME}/app.py
```

Example 3-13   Sample manifest.json File for a Python Application

```
{
  "runtime": {
    "majorVersion": "3.6.0"
  },
  "command": "sh ./start.sh"
}
```

Create Metadata Files

You can specify deployment information, such as the launch command, the number of application instances to create, and service bindings, in one or two metadata files that you upload with your application.

When you upload your application to Oracle Application Container Cloud Service using the user interface, you must include a file called `manifest.json` if your application requires a launch command. This file can be included at the root of the archive or specified separately.

The other file, `deployment.json`, is optional and isn't included in the archive. You can specify the values in this file via the user interface, or you can upload the file using the REST API.

Topics:

- Create the manifest.json File
- Create the deployment.json File

Create the manifest.json File

The `manifest.json` file specifies how to launch your application. Optionally, you can include the runtime version and other parameters.

manifest.json Syntax

Syntax

- **runtime**
  - **majorVersion** – (Optional) Major version of the runtime environment. Each language has its own numbering system.
    * For Java SE, use `7`, `8`, `9`, or `10`.
    * For Java EE, use `7`.
    * For Node.js, use `0.10` , `0.12`, `4`, `6`, or `8`.
    * For PHP, use `5.6`, `7.0`, or `7.1`.
    * For Python, use `2.7.13`, `3.6.0`, `3.6.1`.
    * For Ruby, use `2.3.4`, `2.4.1`.
    * For Go, use `1.7.6`, `1.8.3`.
    * For .NET, use `1.1.2-runtime` or `2.0.0-runtime`.
- **type** – (Optional) Determines whether an application is public or private:
  - `web` (the default) – Specifies a public application, which you can access using a public URL, the public REST API, or the command-line interface.

  - `worker` – Specifies a worker application, which is private and runs in the background. The `isClustered` parameter should be set to `true` in some cases. See Worker Applications.

- **command** – (Required except for Java EE and PHP) Launch command to execute after the application has been uploaded.

  Most PHP applications don't need a launch command. Unless another file is specified in the application URL, the `index` file opens first, whether the extension is `.htm`, `.html`, or `.php`. See Select the Launch Command.

  For a Java EE web application, the launch command, if present, is ignored.

- **startupTime** – (Optional) Maximum time in seconds to wait for the application to start. Allowed values are between 10 and 600. The default is 30. If the application doesn't start in the time specified, the application is deemed to have failed to start and is terminated. For example, if your application takes two minutes to start, set startupTime to at least 120.

- **shutdownTime** – (Optional) Maximum time in seconds to wait for the application to stop. Allowed values are between 0 and 600. The default is 0. This allows the application to close connections and free up resources gracefully. For example, if your application takes two minutes to shut down, set shutdownTime to at least 120.

- **release** – (Optional)
  - **build** – User-specified value of build.
  - **commit** – User-specified value of commit.
  - **version** – User-specified application version.

- **notes** – (Optional) Comments.

- **mode** – (Optional) Restart mode for application instances when the application is restarted. The only allowed option is `rolling` for a rolling restart. Omit this parameter to be prompted for a rolling or concurrent restart. See Stopping, Starting, and Restarting an Application in *Using Oracle Application Container Cloud Service*.

- **isClustered** – (Optional) Must be set to `true` for application instances to act as a cluster, with failover capability. See Prepare a Clustered Application for Deployment.

- **home** – (Optional) Context root of the application. The value of the `home` parameter is appended to the application URL.

- **healthcheck** – (Optional) Allows you to define a URL for your application that the system uses for health checks. The URL must return an HTTP response of `200 OK` to indicate that the application is healthy. This parameter is only available for cloud accounts with Oracle Identity Cloud Service.
  - **http-endpoint** – Defines the URI that is appended to the application URL to create the health check URL. For example, if the application URL is `http://myapp.example.com` and the end point is set to `{"http-endpoint":"/health"}`. Then the health check will test `http://myapp.example.com/health`.

Note:

The default URL used for health checks is the application root. If no value is set for `http-endpoint`, then the system will use the application root for health checks. The root application URL must return `200 OK` for the application to be considered "healthy" by the health check feature.

Example 3-14 Sample manifest.json for a Java application

```
{
    "runtime": {
        "majorVersion": "7"
    },
    "type": "web",
    "command": "java -jar myapp.jar",
    "startupTime": "120",
    "release": {
        "build": "150520.1154",
        "commit": "d8c2596364d9584050461",
        "version": "15.1.0"
    },
    "notes": "notes related to release",
    "mode": "rolling",
    "home": "/home.jsp",
    "healthCheck": {
      "http-endpoint": "/health"
    }
}
```

Create the deployment.json File

The `deployment.json` file specifies how much memory to allocate to the application, how many application instances to create initially, additional environment variables, and service bindings to other Oracle Cloud services. You can specify these same options from the user interface, or you can upload this file using the REST API. If no values are specified or the file is omitted, then memory and instance defaults are used. This file is optional.

deployment.json Syntax

- **memory** – The amount of memory in gigabytes made available to the application. Values range from 1G to 20G. The default is 2G.
- **instances** – Number of application instances. The default is 2. The maximum is 64.
- **notes** – Free-form comment field. It can be used, for example, to describe the deployment plan.
- **environment** – Environment variables used by the application. This element can contain any number of name-value pairs.
  - **name** – Environment variable name.
  - **value** – Environment variable value.

- **secureEnvironment** – List of environment variables marked as secured on the user interface. The environment variables to be secured must be present in the **environment** property.
- **java_system_properties** – Java EE system properties used by the application. This element can contain any number of name-value pairs.
  - **name** – Property name.
  - **value** – Property value.
- **services** – Service bindings for connections to other Oracle Cloud services. This element can contain more than one block of sub-elements. Each block specifies one service binding.
  - **identifier** – User-specified identifier.
  - **type** – Type of the service: `JAAS` for Oracle Java Cloud Service, `DBAAS` for Oracle Database Cloud Service, `MYSQLCS` for MySQL Cloud Service, `OEHCS` for an Oracle Event Hub Cloud Service topic, `OEHPCS` for an Oracle Event Hub Cloud Service cluster, `DHCS` for Oracle Data Hub Cloud Service, or `caching` for a cache.
  - **name** – Name of the service, the name of an existing Oracle Java Cloud Service instance, Oracle Database Cloud Service database, MySQL Cloud Service database, Oracle Event Hub Cloud Service topic or cluster, Oracle Data Hub Cloud Service instance, or cache name.
  - **username** – Username used to access the service.
  - **password** – Password for the username.

Note:

The **username** and **password** are not automatically used to authenticate against the target service. The values are placed in the *SERVICE*_USER_NAME and *SERVICE*_USER_PASSWORD environment variables, which your application can access. If the target service requires authentication, then your application must handle it. If the target service doesn’t require authentication, you can’t omit the username and password from the **services** element, but you can specify any values.

Note:

If you download a `deployment.json` file from a deployed application that has a service binding, then you see an additional **id** element under **services**. The value of this element is system-generated. For a new application, do not upload a `deployment.json` file that includes this element. For an existing application, do not change the value of this element.

Example 3-15 deployment.json

```
{
    "memory": "2G",
    "instances": "2",
    "environment": {
        "NO_OF_CONNECTIONS":"25",
        "TWITTER_ID":"JAVA",
        "user": "joe.smith@example.com"
    },
    "secureEnvironment": [
        "user"
    ],
    "services": [{
        "identifier": "ProdService",
        "type": "JAAS",
        "name": "Jaas Service",
        "username": "username",
        "password": "password"
    },
    {
        "identifier": "DBService",
        "type": "DBAAS",
        "name": "MyDB",
        "username": "username",
        "password": "password"
    }]
}
```

Create the Deployment-Ready Archive

Finally, create an archive that includes your application and its `manifest.json` file, if present, at the root. Use a `zip` or `tar` utility.

You don't need to include a Java EE web application in a `.zip`, `.tgz`, or `.tar.gz` file unless you need to set values in the `manifest.json` file, which is optional for this application type.

Don't include `deployment.json` in your compressed archive.

For example, create an archive called `myapp.zip` using `zip`.

```
zip myapp.zip manifest.json myapp.jar
```

Here's an example using `tar` command.

```
tar cvfz myapp.tgz manifest.json myapp.jar
```

Prepare a Java EE Web Application for Deployment

You can deploy a Java EE web application to Oracle Application Container Cloud Service with minimal changes to how it's packaged.

Your Java EE web application must meet some standard Oracle Application Container Cloud Service requirements, while others are optional or not applicable.

- Only Java EE version 7 is supported.
- Your web application doesn't need to read the APP_HOME and PORT environment variables.
- Your web application doesn't use a launch command.
- Your web application must include all dependent classes that are not included in WebLogic Server.

- A `manifest.json` file is optional. You can deploy either a single `.war` file or a `.zip`, `.tar`, or `.tar.gz` file with an optional `manifest.json` file and a single `.war` file at the root. The `isClustered` parameter is not supported.
- If your web application requires a service binding to a DBCS or MySQLCS database, you must specify it in a `deployment.json` file. A WebLogic data source is automatically created with the default name `jdbc/`*`service-binding-name`*`DS`. You can specify database driver and data source parameters in this file if necessary. Names of data source properties align with WebLogic data source MBean properties.
- You can define any needed environment variables and system properties in a `deployment.json` file.
- Your web application can interact with a cache using the Java API or the REST API.

For details about the `manifest.json` and `deployment.json` files, see Create Metadata Files.

After deployment, your web application has these features:

- Your web application is deployed to WebLogic Server on the back end.
- You can scale memory and instances just as you can with any application.
- You can stop, start, and restart just as you can with any application.

Most standard web application features are supported, with a few exceptions.

- XA and RAC integration are not supported.
- ADF is not supported.
- Although a `weblogic.xml` file isn't required, it is used if present.

To learn more about how to deploy an application, see Deploy an Application.

Prepare a Cloud Foundry Application for Deployment

You can deploy a Cloud Foundry application to Oracle Application Container Cloud Service with minimal changes to how it's packaged, using your existing `manifest.yml` file.

You can deploy your Cloud Foundry application to Oracle Application Container Cloud Service as is, if it meets all of these conditions:

- The `manifest.yml` file contains information for only one application.
- The `manifest.yml` file contains no service bindings to databases.
- The `manifest.yml` file is located in the root directory of the application ZIP file to be uploaded.
- All dependencies are included in a single file, either the application main class or a JAR file. Unlike Cloud Foundry, Oracle Application Container Cloud Service doesn't download dependencies.

If your application requires a service binding to a database, you must specify it in a `deployment.json` file. See Create Metadata Files. All other conditions are requirements.

Oracle Application Container Cloud Service interprets the `manifest.yml` file as follows:

- Supported `manifest.yml` attributes are `name`, `memory`, `instances`, `path`, `env`, and `timeout`.
- The launch command is automatically generated based on the `path` attribute.
- The `memory` value is rounded up to the nearest gigabyte.
- Service bindings and unsupported options are ignored.
- If present, `manifest.json` and `deployment.json` files override equivalent options in the `manifest.yml` file.

To learn more about how to deploy an application, see Deploy an Application.

Prepare a Clustered Application for Deployment

Configuring the Metadata Files and Creating the Archive

Set the following parameters in the `manifest.json` file using the launch command you need:

```
{
   "runtime": {
      "majorVersion": "8"
   },
   "command": "./start.sh",
   "isClustered" : "true"
}
```

Set the following parameters in the `deployment.json` file using the values you need:

```
{
   "memory": "1G",
   "instances": "2",
   "environment": {
      "MIP":"228.0.0.4",
      "MPORT":"45565"
   }
}
```

For details about other possible parameters in the `manifest.json` and `deployment.json` files, see Create Metadata Files.

You can include the `manifest.json` file at the root of the application archive, or you can reference it separately at deployment time. Do not include the `deployment.json` file, which must be referenced separately.

To learn more about how to deploy an application, see Deploy an Application.

Setting the Mulitcast IP Address and Port

Your clustered instances must be able to discover each other in order to share session data. One way they can do this is to use multicasting.

To use multicasting, you must select the multicast IP address and port that your clustered application will use. For example, the default values used by Apache Tomcat are 228.0.0.4 and 45564 respectively. Valid multicast IP addresses are described at the IPv4 Multicast Address Space Registry.

It's very important that the multicast IP address and port combination be unique in your identity domain. If different clusters use the same combination, they behave like a fused cluster, causing problems for all applications involved.

To avoid having to set the multicast IP address and port manually in each application instance, set them as environment variables in the `deployment.json` file and pass them to the application instances in the launch command. This is recommended as a best practice.

Like most Oracle Application Container Cloud Service applications, a clustered application must read the standard PORT environment variable, which is different from the multicast port.

For example, Tomcat stores the multicast IP address, multicast port, and standard PORT in the `server.xml` file. You can create a template of this file and use the launch command to copy this file for each instance and substitute environment variable values in each copy.

Copy the `conf/server.xml` file and create a new file named `conf/server.template.xml`. Edit the `<Service name="Catalina">` section to reference the standard PORT environment variable:

```
<Connector port="__PORT__" protocol="HTTP/1.1" connectionTimeout="20000" ... >
```

Edit the `<Engine name="Catalina" ... >` subsection to reference the multicast IP address and port:

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster">
   <Channel className="org.apache.catalina.tribes.group.GroupChannel">
      <Membership className="org.apache.catalina.tribes.membership.McastService"
         address="__MIP__"
         port="__MPORT__"
         frequency="500"
         dropTime="3000"/>
   </Channel>
</Cluster>
```

Use a `sed` command in the launch script that replaces the multicast address and port and the standard PORT with environment variables:

```
sed "s/__PORT__/${PORT}/g; s/__MIP__/${MIP}/g; s/__MPORT__/${MPORT}/g" conf/
server.template.xml > conf/server.xml
```

Note:

For local testing, you can omit the `sed` command from the launch script and use the actual PORT, multicast IP, and multicast port values in the application.

For a full tutorial of how to create an example clustered application that uses multicasting based on Tomcat, see Creating a Tomcat Cluster with TCP Session Replication.

Configuring a Web Application

For failover to work, a web application must have `<distributable/>` set in its `web.xml` file.

You can download a useful sample application based on Tomcat at clusterjsp.zip.

Prepare an Application Stored on GitHub for Deployment

Oracle Application Container Cloud Service GitHub integration allows you to deploy your applications directly from GitHub. Applications are deployed using the Oracle command line interface or REST API.

To deploy an application from GitHub, follow these steps:

1. Obtain the GitHub URL for the Git repository containing your application. (This would be the URL you use to `clone` the repository).
2. Execute a deployment command for you application (see the following sample commands).
3. Observe the deployment in the Oracle Application Container Cloud Console.

The GitHub integration has the following limitations.

- Builds are only supported from the master branch.
- Only Java SE, Java EE, and Node.js applications are supported.
  - **Java SE and Java EE applications:**
    - Builds are Maven based.
    - Your `pom.xml` file must be in the root directory of your master branch.
  - **Node.js applications:**
    - The application must have a `package.json` file in the root directory.
- Applications must deployed using the command line interface or REST API.

Deploying Using the Command-Line Interface

To deploy your web application using the command-line interface, use the `accs push` command. See psm accs push in *PaaS Service Manager Command Line Interface Reference*.

Example 3-16 Deploying a Java Application from a GitHub Repository by Using the Command-Line Interface

```
psm accs push -n MyJavaApp -r java -s Monthly \
-g https://github.com/YourGitProject/MyRepoName.git \
-m /local-path-to-manifest.json -d /local-path-to-deployment.json
```

After you execute the command, you are asked if the repository is public. If you answer no, you will be prompted for your user name and password for the private repository.

Note:

Two factor authentication is currently not supported.

Running the PSM command again generates a build based on the files in the repository. If the source has changed, a new version of the application is deployed.

Tutorial - Deploy an application from GitHub by using the command-line interface

Deploying Using the REST API

To deploy your web application using the REST API, use the `gitRepoUrl` option to specify the git repository. This example shows how to deploy a Java application called `MyJavaApp` by submitting a `POST` request using cURL.

Example 3-17 Deploying a Node.js Application from a GitHub Repository by Using the REST API

```
curl -X POST -u joe@example.com:password\
  https://apaas.oraclecloud.com/paas/service/apaas/api/v1.1/apps/
ExampleIdentityDomain \
  -H "X-ID-TENANT-NAME:ExampleIdentityDomain" \
  -H "Cache-Control: no-cache" \
  -H "content-type: multipart/form-data;"  \
  -F "name=MyNodeApp" \
  -F "runtime=node" \
  -F "subscription=MONTHLY" \
  -F "deployment=@Local-path-to-deployment-json\deployment.json" \
  -F "gitRepoUrl=https://github.com/YourGitProject/hello-world.git" \
  -F "manifest=@Local-path-to-manifest-json\manifest.json" \
  -F "gitUserName=YourUserName" \
  -F "gitPassword=YourPassword"
```

Both `gitUserName` and `gitPassword` are optional. Only provide these values if your repository is private on GitHub.

Any option on the command line (such as subscription or name) takes precedence over the same option in a metadata file, if there is a difference.

Tutorial - Deploy an application from GitHub by using the REST API

For more information about the REST API, see *REST API for Oracle Application Container Cloud Service*.

Deploy an Application

You can deploy an application to Oracle Application Container Cloud Service by using the service user interface console, the command-line interface, or by using the REST API.

Deploying the Archive by Using the Service User Interface Console

To deploy your application by using the Oracle Application Container Cloud Service console, see Creating an Application in *Using Oracle Application Container Cloud Service*

Tutorial - Deploy a Java application to Oracle Cloud

Deploying the Archive by Using the Command-line Interface

To deploy your application using the command-line interface, use the `accs push` command.

Example 3-18 Deploying a Java Application by Using the Command-line Interface

```
psm accs push -n MyJavaApp -r java -s Monthly \
-p /home/myapp.zip \
-m /local-path-to-manifest.json -d /local-path-to-deployment.json
```

To learn more, see psm accs push in *PaaS Service Manager Command Line Interface Reference*.

Tutorial - Deploy an application by using the command-line interface

Deploying the Archive by Using the REST API

To deploy your application by using the REST API, create your archive and place it in your Oracle Cloud Infrastructure Object Storage Classic account.

Example 3-19 Creating a Storage Container

```
curl -i -X PUT -H -u joe@example.com:password \
https://ExampleIdentityDomain.storage.oraclecloud.com/v1/Storage-
ExampleIdentityDomain/MyPrivateApp
```

Example 3-20 Uploading the Archive to the Storage Container

```
curl -i -X PUT -u joe@example.com:password \
https://ExampleIdentityDomain.storage.oraclecloud.com/v1/Storage-
ExampleIdentityDomain/MyPrivateApp/MyPrivateApp.zip \
-T local-path/MyPrivateApp.zip
```

Example 3-21 Deploying a Java Application by Using the REST API

```
curl -X POST -u joe@example.com:password \
-H "X-ID-TENANT-NAME:ExampleIdentityDomain" \
-H "Content-Type: multipart/form-data" -F "name=MyPrivateApp" \
-F "runtime=java" -F "subscription=Monthly" \
-F "deployment=@deployment.json" \
-F "archiveURL=mydomain/binaries/myprivapp.zip" \
-F "notes=notes for deployment" \
https://apaas.oraclecloud.com/paas/service/apaas/api/v1.1/apps/ExampleIdentityDomain
```

Tutorial - Deploying an Express application to Oracle Application Container Cloud Service
Any option on the command line (such as subscription or name) takes precedence over the same option in a metadata file, if there is a difference.

To learn more, see Create an Application in *REST API for Oracle Application Container Cloud Service*.

4 Sample Applications

Use these tutorials with their sample applications to help you develop and customize your own applications.

The following pages in the Oracle Help Center list tutorials for various languages and topics.

- Create Your First Applications
- Create Java SE Applications
- Create Java EE Applications
- Create Node.js Applications
- Create PHP Applications
- Create Python Applications
- Create Ruby Applications
- Create Go Applications
- Create Caching Applications
- Create .Net Applications

5

Monitoring Your Application

You can monitor your Java applications using Java Flight Recorder and Java Mission Control. Additionally, applications can write to log files that are stored on Oracle Cloud Infrastructure Object Storage Classic.

Topics:

Java Mission Control and Java Flight Recorder

Java Flight Recorder and Java Mission Control are included in your subscription.

Java Mission Control (JMC) allows you to monitor and manage Java applications without introducing the performance overhead normally associated with these types of tools. JMC uses data collected for normal adaptive dynamic optimization of the Java Virtual Machine (JVM). Besides minimizing the performance overhead, this approach eliminates the problem of the observer effect, which occurs when monitoring tools alter the execution characteristics of the system.

Java Flight Recorder (JFR) collects and saves detailed performance characteristics for historic analysis and profiling. When used as a plug-in for the JMC client, JFR presents diagnostic information in logically grouped tables, charts, and dials.

You can record 60 seconds of information at a time on each application. If you have multiple instances, then each instance generates a recording. Then you can examine the data retrieved using Java Mission Control, and make necessary adjustments based on that data.

See *Java Mission Control User's Guide* and *Java Flight Recorder Runtime Guide* in Java Components Documentation.

Retrieve the Application Logs

You may want to check your application logs to monitor the application or troubleshoot a problem. Information that your application sends to `stdout` or `stderr` is captured in the logs. The logs are stored on Oracle Cloud Infrastructure Object Storage Classic. You can download the logs using the user interface console, the command-line interface or the REST API.

Using the User Interface Console

To learn how to get your application's logs by using the Oracle Application Container Cloud Service console, see Retrieving the Application Logs in *Using Oracle Application Container Cloud Service*

Using the Command-line Interface

To download the logs by using the command-line interface first you need to generate the logs by using the `psm accs get-logs` command then you can get the logs of each instance with the `psm accs log` command or all of them with the `psm accs logs` command.

Example 5-1 Generating the Logs for the Application

```
psm accs get-logs -n employees-app -i web.1 -of json
```

Example 5-2 Getting the Logs for the `web.1` Instance

```
psm accs log -n employees-app -i web.1 -of json
```

Example 5-3 Getting the Logs for all Instances

```
psm accs logs -n employees-app -of json
```

To get more information about the `psm accs` commands, see psm accs Commands in *PaaS Service Manager Command Line Interface Reference*

Using the REST API to Retrieve a Log

To download the log for an application from Oracle Cloud Infrastructure Object Storage Classic, use the following command as a reference. You will need the cURL utility and the storage service information that you received when you subscribed to the service.

First you will need an authentication token. Here's an example of a cURL command for requesting an authentication token:

Example 5-4 Requesting an Authentication Token

```
curl -v -X GET \
     -H "X-Storage-User: myService-myIdentityDomain:myUsername" \
     -H "X-Storage-Pass: myPassword" \
     https://storage.us2.oraclecloud.com/auth/v1.0
```

After you have the token, you can request the log, as in this example:

Example 5-5 Requesting the Logs

```
curl -v -X GET \
     -H "X-Auth-Token: AUTH_tkb4fdf39c92e9f62cca9b7c196f8b6e6b" \
     -o destinationFileName \
     https://storage.us2.oraclecloud.com/v1/Storage-myIdentityDomain/myContainer/
myApplicationLog
```

To learn more about the Oracle Cloud Infrastructure Object Storage Classic REST API, see *REST API for Standard Storage in Oracle Storage Cloud Service*.

6

Troubleshooting Oracle Application Container Cloud Service

This section describes common problems that you might encounter when using Oracle Application Container Cloud Service and explains how to solve them.

Topics

My application doesn’t deploy

Your application is of a type that requires a requires a launch command. You upload your application archive in the Create Application dialog but nothing happens. The following error message flashes on the screen:

```
Unsuccessful upload, because the manifest file named manifest.json could not be
found.
```

Here are some common causes of this problem:

- The `manifest.json` file is actually missing. It wasn’t at the root of the archive or specified during deployment.
- There is a typo in the name of the `manifest.json` file.
- The `manifest.json` file isn’t located in the root directory of the archive. This is the most common reason for this problem.

  Typically, when you zip something to share, you put all the files in a subdirectory and then zip the subdirectory. However, this results in a root directory that contains the subdirectory. Oracle Application Container Cloud Service won’t be able to find the manifest file and therefore will be unable to deploy your application. See Create the Deployment-Ready Archive.

My application failed to deploy.

You upload your application file to Oracle Application Container Cloud Service and it starts to deploy then it shows the “Application failed to deploy” message.

You application failed to deploy. To find out why, do the following:

1. Go to the Applications page. See Using the Applications Page in *Using Oracle Application Container Cloud Service*.
2. Click the application name.
3. Click the **Administration** tab then click **Logs**.

4. Expand **Log Capture History** to view the log history.

Here are the most common causes of this problem:

- Your launch command is incorrect, possibly due to a typo. See Select the Launch Command.
- Your application archive is missing a dependent library. Make sure your application can launch stand-alone, separate from your build environment. See Make a Standalone Application.

You can redeploy your application. See Redeploying an Application in *Using Oracle Application Container Cloud Service*.

My application deploys but doesn't run.

Your application runs locally and has deployed successfully, but when you try to test it in Oracle Application Container Cloud Service, you get no response.

The most common cause of this problem is that your application is of a type that is required to read the `PORT` environment variable provided by the Oracle Application Container Cloud Service container but doesn't read them. This results in your application listening on the wrong port, unavailable for testing. Oracle Application Container Cloud Service typically listens on SSL port 443 (HTTPS). See Make the Application Configurable at Runtime.

My clustered application deploys but doesn't connect.

Your clustered application deploys successfully, but it can't connect to other cluster-enabled applications.

When you look at the log, you may see a `java.net.UnknownHostException` error.

The most common cause of this problem is that your application doesn't have `isClustered` set to `true` in the `manifest.json` file. The `isClustered` parameter cannot be reset once an application is deployed, so you must delete the application, set the `isClustered` parameter in the `manifest.json` file to `true`, and deploy the application as if it were new.

For descriptions of all the parameters in the `manifest.json` file, see Create Metadata Files.

My REST request fails with a 403 error.

You submit an Oracle Application Container Cloud Service REST API request and get a 403 Forbidden response.

The most common cause of this error is forgetting to include the user name and password in the request. In a cURL command, you specify these using the `-u` or `--user` option. See the Use cURL section in *REST API for Managing Applications*.