Running and Licensing Oracle Programs in Containers and Kubernetes

Version 1.0

Purpose statement

This document explains how to configure environments that run Oracle Programs in containers and Kubernetes, in order to determine and manage licensing requirements for these Oracle Programs.

Disclaimer

Containers and Kubernetes

For the purposes of this document “**container technology**” refers to software that leverages technologies based on Linux container technologies to deliver, encapsulate and run software as a package of bundled libraries. At runtime, these packages may share an operating system, but they do not have access to the entire operating system - they can only see the contents of the package, and devices assigned to the package. We will refer to these runtime packages of encapsulated and bundled software as “**containers**”. The most well known implementations of container technology at the time of this writing are Docker and CRI-O containers.

Container technologies require that the software libraries used by containers are prepackaged and bundled into a binary package called a “**container image**” or “**image**”. Images are portable between computers. Computers that are equipped with an operating system and a container runtime are referred to as “**hosts**”. A host may be a physical computer or virtual machine that runs the container runtime.

Images are used as an encapsulation entity to package everything needed to instantiate the process that will be executed within the container runtime on a host. Images are typically downloaded from image registries to hosts where containers will be run. Once an image is downloaded to a host, then the software encapsulated in the image is ready to be started and run on that specific host. The process of downloading images to hosts is often called “**pulling**” images, based on the use of the term “**pull**” in the command line interface of Docker.

Pulling an image copies the image from a remote image registry, and stores the image into the local file system of the host that has pulled the image. For example, in the case of Docker, the image is stored in a directory named “/var/lib/docker”). Every image pulled by a host is executable or runnable as a container only on the host that has pulled that image. Images pulled by one host are not executable or runnable on another host. This is true even if the downloaded image, and the directory in which it is stored, are available on a shared file system that is accessible to multiple hosts. For another host to run a container from a given image, that host must first pull its own copy of the image, and write a copy of the image to its own local file system.

Containers that may run across multiple hosts may be managed and synchronized by container orchestration software. Kubernetes is recognized as the leading container orchestration software at the time of this writing. Kubernetes automates deployment, scaling and lifecycle operations of containers across a Kubernetes cluster consisting of multiple hosts, also known as “**Kubernetes nodes”**. Containers are executed on Kubernetes nodes within the boundaries of a Kubernetes lifecycle management entity called a “**Kubernetes pod”**. Kubernetes defines additional Kubernetes terms such as Kubernetes service, Kubernetes deployment, and Kubernetes scheduling. See the Kubernetes documentation (https://kubernetes.io/docs/home) for detailed definitions.

Kubernetes performs scheduling decisions, based on various criteria, in order to start and run pods and associated containers on Kubernetes nodes within Kubernetes clusters. Once a pod and its associated containers are scheduled to run on given Kubernetes node, then Kubernetes will instruct that Kubernetes node to pull the relevant images to that Kubernetes node, and to instantiate containers based on those images. Kubernetes administrators are able to influence the scheduling process, and can control which Kubernetes nodes may run specific pods and pull specific images.

Overview of Licensing of Oracle Programs in Containers and Kubernetes

Container images that have been pulled to a host or a Kubernetes node may contain Oracle Programs. This may lead to licensing requirements on that host or that Kubernetes node for the Oracle Programs encapsulated within the image. As stated in the Oracle Partitioning Policy document:

> “Once a container image (e.g. a Docker image) containing Oracle Programs has been pulled to a host, or to a Kubernetes node in a Kubernetes cluster, (either a virtual machine or a physical machine), that host or Kubernetes node must be licensed for the Oracle Programs for the number of processors on that host or Kubernetes node. If the host or Kubernetes node is a physical machine, the number of processors on that host or Kubernetes node equals the number of processors on that physical machine. If the host or Kubernetes node is a virtual machine, then the number of processors on that host or Kubernetes node is subject to the guidelines documented in this Partitioning Policy.”

Oracle recommends use of the procedures below for configuring environments that run Oracle Programs in containers and Kubernetes, in order to determine and manage licensing requirements for the Oracle Programs that may run in these environments.

Configuring Hosts to Run Oracle Programs in Containers

Customers are responsible for identifying and managing which images contain binaries with Oracle Programs.

Every host that has pulled an image containing Oracle Programs must have appropriate licenses to run the Oracle Programs. The following guidelines should be used for configuring these hosts in order to determine and manage licensing requirements.

1. Users running Oracle Programs in containers should control the number of hosts that are pulling images containing Oracle Programs. Every pull operation executed on a host, for an image containing Oracle Programs, is equivalent to installing the Oracle Programs on that host and will create licensing requirements for that host.
2. The host may be physical or virtual. If the host is physical, then licenses for the Oracle Programs are required for all processors on that physical host. If the host is virtual, then the Oracle Partitioning Policy (https://www.oracle.com/assets/partitioning-070609.pdf) determines the number of processors that require a license for the Oracle Programs on that virtual host.
3. Because container file systems are specific to the container runtime (e.g. the Docker daemon) that owns them and cannot be shared, an image pulled to one host's container file system cannot be used to run a container on another host. Placement of Oracle Programs in container file systems on a storage server (e.g. a SAN) creates Oracle licensing requirements for the host that pulled the container images into that container file system. However, this does not, by itself, create Oracle licensing requirements for other hosts that have access to the same storage server. By controlling which hosts pull images containing Oracle Programs, users may limit Oracle licensing requirements for these Oracle Programs to only the hosts that have pulled the images, even when additional hosts have access to the same storage server where images are stored.

Container technologies allow users to assign a subset of host resources (such as RAM or CPU resources) to the container. For example the command `docker run –cpus=”1.0” <image>` assigns only 1 CPU to the container regardless of how many CPUs are available on the host. Oracle does not recognize these technologies as hard partitioning technologies as defined in the Oracle Partitioning Policy document. Users should not use these container technology features with the expectation that they will limit Oracle licensing requirements.

Configuring Kubernetes to run Oracle Programs

Customers are responsible for identifying and managing which images contain binaries with Oracle Programs.

Every Kubernetes node that has pulled an image containing Oracle Programs must have appropriate licenses to run the Oracle Programs. All guidelines above for configuring hosts to run Oracle Programs in containers also apply to configuring Kubernetes nodes and clusters to run Oracle Programs. Kubernetes provides two features that enable users to control which Kubernetes nodes can potentially run Kubernetes pods, and therefore which Kubernetes nodes will potentially pull images. These features are “node labels” and “node selectors”, and users may leverage these features to limit Oracle licensing requirements for Oracle Programs to certain Kubernetes nodes within a Kubernetes cluster.

A node label is a piece of metadata (key value pair) associated with a Kubernetes node. The node label is added to the Kubernetes node’s configuration using standard Kubernetes tools (e.g. `kubectl`). For example, a user may label a Kubernetes node with a label such as “labelkey=labelvalue”.

Node selectors are fields included in a Kubernetes pod configuration, and control where the Kubernetes pod is scheduled. A node selector tells Kubernetes to schedule the pod only on nodes that have a label that matches the pod’s node selector. If the node selector for a pod was “labelkey=labelvalue”, then Kubernetes would only schedule the Kubernetes pod on Kubernetes nodes that have the corresponding node label.

Configuring Kubernetes to run Oracle Programs on Certain Kubernetes Nodes Using Generic Kubernetes Features

To leverage these Kubernetes features to limit Oracle licensing requirements for Oracle Programs to certain Kubernetes nodes within a Kubernetes clusters, you should perform the following steps using `kubectl` and YAML editing tools:

1. Run “`kubectl get nodes`” to obtain the names of the Kubernetes nodes in the Kubernetes cluster.
2. Choose the subset of Kubernetes nodes where you want to run Oracle Programs.

3. For every Kubernetes node where you want to run Oracle Programs, execute the following command to label the nodes:
   ```
   kubectl label nodes <node-name> <labelkey>=<labelvalue>
   ```
   For example:
   ```
   kubectl label nodes mynode1 oracle=true
   ```

4. For Kubernetes pods containing Oracle Programs, add a nodeSelector field in the "spec." section of your pod's YAML configuration file that matches the node label above, using the following syntax:
   ```
   (…)
       nodeSelector:
          <labelkey>: <labelvalue>
   (…)
   ```
   For example:
   ```
   (…)
     nodeSelector:
       oracle: true
   (…)
   ```

5. You can then cause the Kubernetes pod with the node selector to be scheduled on a Kubernetes node with the node label by executing, for example:
   ```
   kubectl apply -f <yaml_file>
   ```

To verify the assignment of Kubernetes pods to Kubernetes nodes you should execute the following command:
```
kubectl get pods [-n namespace | --all-namespaces] -o wide
```

After executing this command you should see a table containing information about the Kubernetes pods that should be running on Kubernetes nodes within the Kubernetes cluster. The table should look like the following:

```
NAMESPACE     NAME                    (…)            NODE           (…)
                    (… potentially few other rows …)
<name-of-ns>  <name-of-your-pod>    (…)    <name-of-the-node>   (…)
                    (… potentially few other rows …)
```

If the value of `<name-of-the-node>` in the row with your Kubernetes pod is the name of one of the Kubernetes nodes you have labeled in step 3 above, this verifies the assignment of the Kubernetes pod to the proper Kubernetes node. If the above verification fails, then you have made a mistake during the process of labeling the Kubernetes nodes and using the node selector. Before you repeat the procedure you should remove the deployment by logging into the Kubernetes node that may have pulled the image with Oracle Programs and manually removing the local copy of the pulled image (for example through the "`docker rmi`" command).

Configuring Kubernetes to run Oracle Programs on Certain Kubernetes Nodes Using Utilities such as the WebLogic Server Kubernetes Operator

Some Oracle Programs may have built utilities (such as the WebLogic Server Kubernetes Operator) that assist with management of Oracle Programs running in Kubernetes, and that leverage the nodeSelector feature of Kubernetes. You may use the features of these utilities to control which Kubernetes nodes can potentially run Kubernetes pods containing Oracle Programs, and which Kubernetes nodes will potentially pull images containing Oracle Programs, in order to limit Oracle licensing requirements for Oracle Programs to certain Kubernetes nodes within a Kubernetes cluster.

For example, you may use the WebLogic Server Kubernetes Operator to limit licensing requirements for Oracle WebLogic Server to certain Kubernetes nodes within a Kubernetes cluster. To accomplish this you would need to execute the first 3 steps above using generic Kubernetes features as described above. Instead of steps 4-5 above, however, you would indicate through the WebLogic Domain Kubernetes Custom Resource object how the WebLogic Server Kubernetes Operator should provision Kubernetes pods with WebLogic Server binaries, and if the WebLogic Server Kubernetes Operator should use the nodeSelector feature of Kubernetes.

When using the WebLogic Server Kubernetes Operator, the WebLogic Domain Kubernetes Custom Resource is described and defined by a YAML file, often referred to as domain.yaml. For more information, see the following documentation: https://oracle.github.io/weblogic-kubernetes-operator/userguide/managing-domains/domain-resource/.

This YAML file contains three sections: metadata, spec and status. (The detailed reference is at https://github.com/oracle/weblogic-kubernetes-operator/blob/master/docs/domains/Domain.md. Inside the "`spec`" section of this YAML file you may define a "`serverPod`" element that may contain the "`nodeSelector`" element.

This “`serverPod`” element may be applied to following scopes:

- `spec.serverPod` – for domain wide settings
- `spec.adminServer.serverPod` – for settings that are applied to the WebLogic Admin Server
- `spec.clusters[*].serverPod` – for settings that are applied to named clusters
- `spec.managedServer[*].serverPod` – for settings that are applied to named WebLogic Managed Servers

When you want to schedule Kubernetes pods including WebLogic Server binaries to run only on given Kubernetes nodes you should add the following entries to properly scoped “`serverPod`” element:

```
(…)
    nodeSelector:
       <labelkey>: <labelvalue>
(…)
```

For example:

```
(…)
    nodeSelector:
       oracle: true
(…)
```

You may verify the assignment of Kubernetes pods to Kubernetes nodes by executing the following command as described above:

```
kubectl get pods [-n namespace | --all-namespaces] -o wide
```

Connect with us

Call **+1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

blogs.oracle.com facebook.com/oracle twitter.com/oracle