

# Oracle Cloud Native Environment Container Runtimes for Release 1.7



F79933-02  
October 2023



Oracle Cloud Native Environment Container Runtimes for Release 1.7,  
F79933-02

Copyright © 2022, 2023, Oracle and/or its affiliates.

# Contents

## Preface

---

Conventions	iv
Documentation Accessibility	iv
Access to Oracle Support for Accessibility	iv
Diversity and Inclusion	iv

## 1 Introduction to Container Runtimes

---

Introduction to runC	1-1
Introduction to Kata Containers	1-1
Setting Runtime Classes	1-2

## 2 Using runC

---

Installing runC	2-1
Creating runC Containers	2-1

## 3 Using Kata Containers

---

Installing Kata Containers	3-1
Checking Hardware	3-1
Creating Kata Containers	3-1

# Preface

This document contains information about using the container runtimes available with Oracle Cloud Native Environment.

## Conventions

The following text conventions are used in this document:

Convention	Meaning
<b>boldface</b>	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
<i>italic</i>	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.

## Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <https://www.oracle.com/corporate/accessibility/>.

For information about the accessibility of the Oracle Help Center, see the Oracle Accessibility Conformance Report at <https://www.oracle.com/corporate/accessibility/templates/t2-11535.html>.

## Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit <https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab>.

## Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain

compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

# 1

## Introduction to Container Runtimes

This chapter introduces the container runtimes available in Oracle Cloud Native Environment. The available container runtimes are compliant with the Open Container Initiative (OCI) Runtime Specification.

This chapter provides introductory information about runC and Kata Containers.

This document doesn't try to explain how to use images to create containers in any detail, nor does it try to explain how to create and use Kubernetes pods or deployments.

For more detailed information on creating and managing containers using Kubernetes, see [Kubernetes Module](#).

### Introduction to runC

runC is a container runtime based on the Linux Foundation's Runtime Specification (`runtime-spec`). runC is developed by the Open Container Initiative.

runC is a component of Oracle Cloud Native Environment. runC is a Cloud Native Computing Foundation (CNCF) compliant environment to deploy microservices, and to orchestrate containers.

runC is based on a stable release of the upstream runC project. Differences between Oracle versions of the software and upstream releases are limited to Oracle specific fixes and patches for specific bugs.

For upstream runC documentation, see:

<https://github.com/opencontainers/runc/blob/main/man/runc.8.md>

For more information about runC, see:

<https://github.com/opencontainers/runc>

### Introduction to Kata Containers

#### Important:

Kata Containers is deprecated and might be removed in a future release.

You can provide extra security and isolation of workloads using Kata Containers. Kata Containers is based on the upstream Kata Containers OpenStack Foundation project. Kata Containers delivers the framework for creating lightweight virtual machines, that can easily plug into a container ecosystem. Kata Containers offers extra levels of security, while maintaining the development and deployment speed of traditional containers.

Kata Containers is a component of Oracle Cloud Native Environment. Kata Containers is a Cloud Native Computing Foundation (CNCF) compliant environment to deploy microservices, and to orchestrate containers.

Kata Containers is based on a stable release of the upstream Kata Containers project. Differences between Oracle versions of the software and upstream releases are limited to Oracle specific fixes and patches for specific bugs.

For upstream Kata Containers documentation, see:

<https://github.com/kata-containers/documentation>

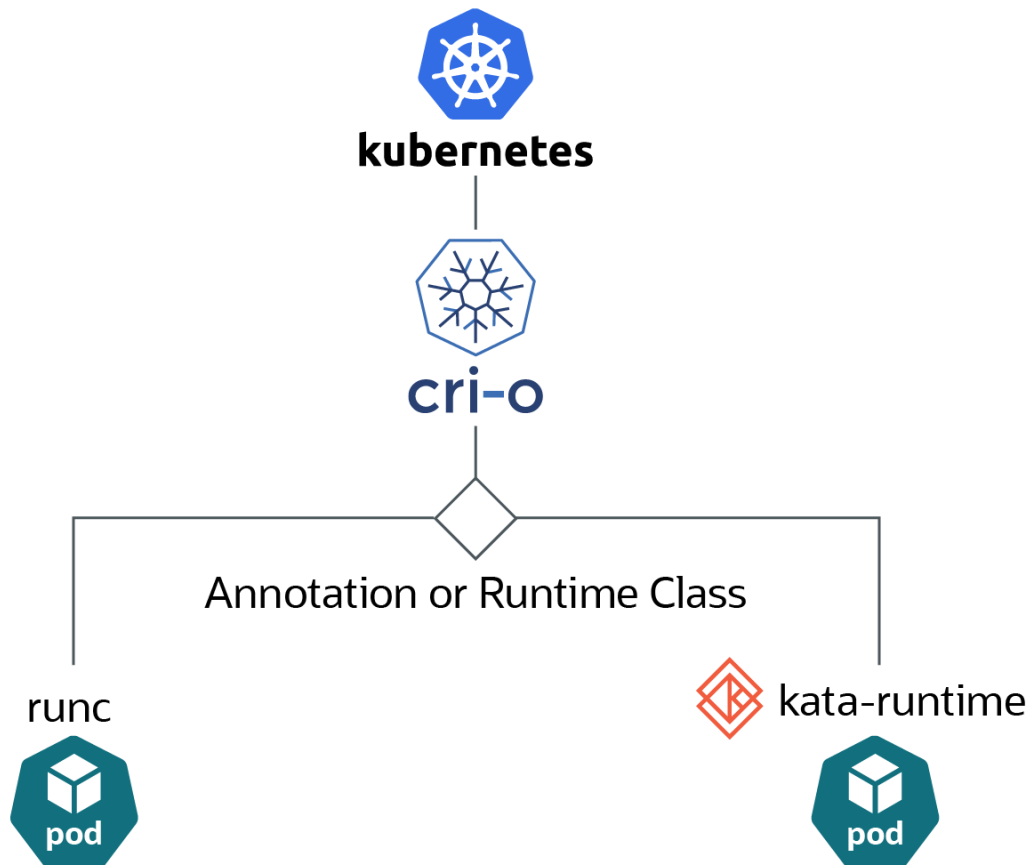
For more information about Kata Containers, see:

<https://katacontainers.io/>

## Setting Runtime Classes

CRI-O uses a Kubernetes annotation or Runtime class set in the pod configuration file to decide whether to run a pod using `runc` or `kata-runtime`.

Figure 1-1 Kubernetes Runtimes



You can create Kubernetes runtime classes to specify whether containers are run as the default runtime, `runc`, or using `kata-runtime`. The examples in this book use the

name `native` to specify the use of `runc`, and the name `kata-containers` to specify the use of `kata-runtime`. You can use any name you like.

To create a runtime class:

1. Create a file for a runtime class for Kata Containers named `kata-runtime.yaml` with the following contents:

```
kind: RuntimeClass
apiVersion: node.k8s.io/v1beta1
metadata:
  name: kata-containers
handler: kata
```

Load the runtime class to the Kubernetes deployment:

```
kubectl apply -f kata-runtime.yaml
```

The runtime class `kata-containers` can now be used in pod configuration files to specify a container is to be run as a Kata container, using the `kata-containers` runtime. For examples of creating pods using this runtime class, see [Creating Kata Containers](#).

2. (Optional) To specify a runtime for `runc`, you can do this in a similar way. This is an optional configuration step. As `runc` is the default runtime, pods automatically run using `runc` unless you specify otherwise. This file is named `runc-runtime.yaml`:

```
kind: RuntimeClass
apiVersion: node.k8s.io/v1beta1
metadata:
  name: native
handler: runc
```

Load the runtime class to the Kubernetes deployment:

```
kubectl apply -f runc-runtime.yaml
```

The runtime class `native` can be used in pod configuration files to specify a container is to be run as a runC container, using the `runc` runtime.

3. You can see a list of the available runtime classes for a Kubernetes cluster using the `kubectl get runtimeclass`. For example:

```
kubectl get runtimeclass
```

The output looks similar to:

NAME	CREATED AT
<code>kata-containers</code>	<date>
<code>native</code>	<date>



# 2

## Using runC

This chapter discusses creating runC containers.

### Installing runC

To deploy runC containers you must first set up an Oracle Cloud Native Environment, including the `kubernetes` module. For information on installing and deploying Oracle Cloud Native Environment, see [Getting Started](#).

### Creating runC Containers

RunC is the default runtime when you create containers using the `kubectl` command. No special runtime class is needed. For information creating containers using `runc` as the runtime engine, see [Kubernetes Module](#).

# 3

## Using Kata Containers

This chapter discusses creating Kata containers. This information can be used to verify the installation is successful, and that you can create containers using `kata-runtime` as the runtime engine.

### ! Important:

Kata Containers is deprecated and might be removed in a future release.

## Installing Kata Containers

To deploy Kata Containers you must first set up an Oracle Cloud Native Environment, including the `kubernetes` module. For information on installing and deploying Oracle Cloud Native Environment, see [Getting Started](#).

## Checking Hardware

You can test whether the hardware can run Kata Containers using the `kata-runtime kata-check` command. To use this command you must first have a running Kubernetes deployment. On a Kubernetes worker node, run:

```
sudo kata-runtime kata-check
```

For more information on using the `kata-runtime` command, use the `kata-runtime --help` command.

## Creating Kata Containers

This section provide an example of creating a Kubernetes pod configuration file, which is used to create a container using `kata-runtime` as the runtime engine. Before you create Kata Containers, set up a Kubernetes runtime class for `kata-runtime`. For information on setting up a runtime class, see [Setting Runtime Classes](#).

### Example 3-1 Creating an NGINX container

This example uses a Kubernetes pod configuration file to create a Kata container. The pod configuration file creates an NGINX web server container, which is often used when testing containers.

To create an NGINX Kata container:

1. On a host that's set up to use the `kubectl` command to connect to the Kubernetes cluster, create a Kubernetes pod configuration file. To specify the container is to be run as a Kata container, use the notation `runtimeClassName: kata-containers` in the pod file.

When CRI-O finds this runtime class in a pod file, it passes the container to `kata-runtime` to run the container.

This pod file is named `kata-nginx.yaml`.

```
apiVersion: v1
kind: Pod
metadata:
  name: kata-nginx
spec:
  runtimeClassName: kata-containers
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

2. Create the Kata container using the `kata-nginx.yaml` file with the `kubectl apply` command:

```
kubectl apply -f kata-nginx.yaml
```

3. To check the pod has been created, use the `kubectl get pods` command:

```
kubectl get pods
```

The output looks similar to:

NAME	READY	STATUS	RESTARTS	AGE
kata-nginx	1/1	Running	0	11s

4. Use the `kubectl describe` command to show a more detailed view of the pod, including which worker node is hosting the pod and the Container ID.

```
kubectl describe pod kata-nginx
```

The output looks similar to:

```
Name:          kata-nginx
Namespace:     default
Priority:       0
PriorityClassName: <none>
Node:          worker1.example.com/192.0.2.24
Start Time:    <date> 01:53:35 +0100
Labels:        <none>
Annotations:   kubect1.kubernetes.io/last-applied-
configuration: {"apiVersion":"v1","kind":"Pod","metadata":
{"annotations":{}}...
Status:        Running
IP:            10.244.3.3
Containers:
  mycontainer:
```

```

Container ID:  cri-o://
8f7d91a1893638498b3bbf74271e4b24361830e29ac65cc361a4c0...
Image:         nginx
Image ID:     docker.io/library/
nginx@sha256:099019968725f0fc12c4b69b289a347...
Port:         80/TCP
Host Port:    0/TCP
State:        Running
...

```

5. You can list the pods on a worker node using the `crictl pods` command. For example, on a worker node, run:

```
sudo crictl pods
```

The output looks similar to:

POD ID	CREATED	STATE	NAME
03564d1e87df9	1 hours ago	Ready	kata-nginx
default	0		
3bfabc5c7eea5	22 hours ago	Ready	kube-flannel-ds-6fkld
system	0		kube-
bb0de1bff1cdf	22 hours ago	Ready	kube-proxy-cc7tb
system	0		kube-

You can see the `kata-nginx` container is running on this worker node.

For more information on using the `crictl` command, use the `crictl --help` command.

6. You can get more detailed information about the containers on a worker node using the `crictl ps` command. For example:

```
sudo crictl ps
```

The output looks similar to:

CONTAINER	IMAGE	NAME	POD ID
<b>8f7d91a189363</b>	docker.io/library/nginx...	nginx	...
03564d1e87df9			
0e9db3f09163a	0a95ca9313ebb9fc3708d8...	kube-flannel	...
3bfabc5c7eea5			
f8350c6fe0c55	container-registry.ora...	kube-proxy	...
bb0de1bff1cdf			

Note the Container ID is a shortened version of the Container ID shown in the pod description.

7. To get detailed information about a container, use the `crictl inspectp` command using the POD ID. For example:

```
sudo crictl inspectp 03564d1e87df9
```

The output looks similar to:

```
{
  "status": {
    "id":
"03564d1e87df9d7330e949e67e18252d2a02b0fac585293667d7dd7b92857b9b",
    "metadata": {
      "attempt": 0,
      "name": "kata-nginx",
      "namespace": "default",
      "uid": "bfd5be6-d4f7-11e9-8ad8-52540037f605"
    },
    "state": "SANDBOX_READY",
    "createdAt": "<date>",
    "network": {
      "ip": "10.244.3.3"
    }
  }
  ...
}
```

- To confirm the container is running using `kata-runtime`, use the `kata-runtime list` command. For example:

```
sudo kata-runtime list
```

The output looks similar to:

ID	PID	STATUS	
BUNDLE	...		
03564d1e87df9d7330e949e67e1825...	20140	running	/run/ containers/storage/...
<b>8f7d91a1893638498b3bbf74271e4b...</b>	20191	running	/run/ containers/storage...

Note the ID is the same as the Container ID shown in the pod description.

- You can delete the pod using the `kubectl delete` command on the control plane node:

```
kubectl delete pod kata-nginx
```