



OpenShift sandboxed containers 1.6

User guide

Deploying sandboxed containers in OpenShift Container Platform

OpenShift sandboxed containers 1.6 User guide

Deploying sandboxed containers in OpenShift Container Platform

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Deploying OpenShift sandboxed containers in OpenShift Container Platform on bare metal, public cloud, and IBM platforms.

Table of Contents

PREFACE	5
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION	5
CHAPTER 1. ABOUT OPENSIFT SANDBOXED CONTAINERS	6
1.1. FEATURES	6
1.2. COMPATIBILITY WITH OPENSIFT CONTAINER PLATFORM	7
1.3. NODE ELIGIBILITY CHECKS	8
1.4. COMMON TERMS	9
1.5. OPENSIFT SANDBOXED CONTAINERS OPERATOR	10
1.6. OPENSIFT VIRTUALIZATION	10
1.7. STORAGE CONSIDERATIONS	10
1.7.1. Block volume support	10
1.8. FIPS COMPLIANCE	11
CHAPTER 2. DEPLOYING WORKLOADS ON BARE METAL	13
2.1. PREPARING YOUR ENVIRONMENT	13
2.1.1. Resource requirements	13
2.1.2. Installing the OpenShift sandboxed containers Operator	15
2.1.2.1. Installing the Operator by using the web console	15
2.1.2.2. Installing the Operator by using the CLI	16
2.1.2.3. Additional resources	17
2.1.3. Creating the NodeFeatureDiscovery CR	18
2.2. DEPLOYING WORKLOADS BY USING THE WEB CONSOLE	19
2.2.1. Creating a KataConfig custom resource	19
2.2.2. Configuring workload objects	21
2.3. DEPLOYING WORKLOADS BY USING THE COMMAND LINE	22
2.3.1. Optional: Provisioning local block volumes by using the Local Storage Operator	22
2.3.2. Optional: Deploying nodes on a block device	24
2.3.3. Creating a KataConfig custom resource	25
2.3.4. Optional: Modifying pod overhead	27
2.3.5. Configuring workload objects	27
CHAPTER 3. DEPLOYING WORKLOADS ON PUBLIC CLOUD	29
3.1. DEPLOYING WORKLOADS ON AWS	29
3.1.1. Preparing your environment	29
3.1.1.1. Resource requirements	29
3.1.1.2. Enabling ports for AWS	31
3.1.1.3. Installing the OpenShift sandboxed containers Operator	32
3.1.1.3.1. Installing the Operator by using the web console	32
3.1.1.3.2. Installing the Operator by using the CLI	32
3.1.1.3.3. Additional resources	34
3.1.2. Deploying workloads by using the web console	34
3.1.2.1. Creating a secret	34
3.1.2.2. Creating a config map	35
3.1.2.3. Creating a KataConfig custom resource	37
3.1.2.3.1. Optional: Verifying the pod VM image	39
3.1.2.4. Optional: Modifying the number of peer pod VMs per node	40
3.1.2.5. Configuring workload objects	40
3.1.3. Deploying workloads by using the command line	42
3.1.3.1. Creating a secret	42
3.1.3.2. Creating a config map	43
3.1.3.3. Creating a KataConfig custom resource	45

3.1.3.3.1. Optional: Verifying the pod VM image	46
3.1.3.4. Optional: Modifying the number of peer pod VMs per node	47
3.1.3.5. Configuring workload objects	47
3.2. DEPLOYING WORKLOADS ON AZURE	49
3.2.1. Preparing your environment	49
3.2.1.1. Resource requirements	49
3.2.1.2. Installing the OpenShift sandboxed containers Operator	51
3.2.1.2.1. Installing the Operator by using the web console	51
3.2.1.2.2. Installing the Operator by using the CLI	52
3.2.1.2.3. Additional resources	53
3.2.2. Deploying workloads by using the web console	53
3.2.2.1. Creating a secret	53
3.2.2.2. Creating a config map	55
3.2.2.3. Creating an SSH key secret	57
3.2.2.4. Creating a KataConfig custom resource	57
3.2.2.4.1. Optional: Verifying the pod VM image	59
3.2.2.5. Optional: Modifying the number of peer pod VMs per node	60
3.2.2.6. Configuring workload objects	60
3.2.3. Deploying workloads by using the command line	62
3.2.3.1. Creating a secret	62
3.2.3.2. Creating a config map	63
3.2.3.3. Creating an SSH key secret	65
3.2.3.4. Creating a KataConfig custom resource	65
3.2.3.4.1. Optional: Verifying the pod VM image	67
3.2.3.5. Optional: Modifying the number of peer pod VMs per node	68
3.2.3.6. Configuring workload objects	68
CHAPTER 4. DEPLOYING WORKLOADS ON IBM	70
4.1. PREPARING YOUR ENVIRONMENT	70
4.1.1. Resource requirements	71
4.1.2. Installing the OpenShift sandboxed containers Operator	72
4.1.2.1. Installing the Operator by using the web console	72
4.1.2.2. Installing the Operator by using the CLI	73
4.1.2.3. Additional resources	74
4.2. DEPLOYING WORKLOADS BY USING THE COMMAND LINE	74
4.2.1. Configuring a libvirt volume	75
4.2.2. Creating a KVM guest image	76
4.2.3. Building a peer pod VM image	77
4.2.4. Creating a secret	79
4.2.5. Creating a config map	80
4.2.6. Creating an SSH key secret	80
4.2.7. Creating a KataConfig custom resource	81
4.2.8. Optional: Modifying the number of peer pod VMs per node	83
4.2.9. Configuring workload objects	83
CHAPTER 5. MONITORING	85
5.1. ABOUT METRICS	85
5.2. VIEWING METRICS	85
CHAPTER 6. UNINSTALLING	87
6.1. UNINSTALLING BY USING THE WEB CONSOLE	87
6.1.1. Deleting workload pods	87
6.1.2. Deleting the KataConfig CR	87
6.1.3. Uninstalling the Operator	88

6.1.4. Deleting the KataConfig CRD	89
6.2. UNINSTALLING BY USING THE CLI	89
6.2.1. Deleting workload pods	90
6.2.2. Deleting the KataConfig CR	90
6.2.3. Uninstalling the Operator	91
6.2.4. Deleting the KataConfig CRD	92
CHAPTER 7. UPGRADING	93
7.1. UPGRADING RESOURCES	93
7.2. UPGRADING THE OPERATOR	93
CHAPTER 8. TROUBLESHOOTING	94
8.1. COLLECTING DATA FOR RED HAT SUPPORT	94
Using the must-gather tool	94
8.2. COLLECTING LOG DATA	95
8.2.1. Enabling debug logs for CRI-O runtime	95
8.2.2. Viewing debug logs for components	96
Additional resources	97
APPENDIX A. KATACONFIG STATUS MESSAGES	98

PREFACE

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

You can provide feedback or report an error by submitting the **Create Issue** form in Jira. The Jira issue will be created in the Red Hat Hybrid Cloud Infrastructure Jira project, where you can track the progress of your feedback.

1. Ensure that you are logged in to Jira. If you do not have a Jira account, you must create a [Red Hat Jira account](#).
2. Launch the [Create Issue form](#).
3. Complete the **Summary**, **Description**, and **Reporter** fields.
In the **Description** field, include the documentation URL, chapter or section number, and a detailed description of the issue.
4. Click **Create**.

CHAPTER 1. ABOUT OPENSIFT SANDBOXED CONTAINERS

OpenShift sandboxed containers for OpenShift Container Platform integrates Kata Containers as an optional runtime, providing enhanced security and isolation by running containerized applications in lightweight virtual machines. This integration provides a more secure runtime environment for sensitive workloads without significant changes to existing OpenShift workflows. This runtime supports containers in dedicated virtual machines (VMs), providing improved workload isolation.

1.1. FEATURES

OpenShift sandboxed containers provides the following features:

Run privileged or untrusted workloads

You can safely run workloads that require specific privileges, without the risk of compromising cluster nodes by running privileged containers. Workloads that require special privileges include the following:

- Workloads that require special capabilities from the kernel, beyond the default ones granted by standard container runtimes such as CRI-O, for example to access low-level networking features.
- Workloads that need elevated root privileges, for example to access a specific physical device. With OpenShift sandboxed containers, it is possible to pass only a specific device through to the virtual machines (VM), ensuring that the workload cannot access or misconfigure the rest of the system.
- Workloads for installing or using **set-uid** root binaries. These binaries grant special privileges and, as such, can present a security risk. With OpenShift sandboxed containers, additional privileges are restricted to the virtual machines, and grant no special access to the cluster nodes.

Some workloads require privileges specifically for configuring the cluster nodes. Such workloads should still use privileged containers, because running on a virtual machine would prevent them from functioning.

Ensure isolation for sensitive workloads

The OpenShift sandboxed containers for Red Hat OpenShift Container Platform integrates Kata Containers as an optional runtime, providing enhanced security and isolation by running containerized applications in lightweight virtual machines. This integration provides a more secure runtime environment for sensitive workloads without significant changes to existing OpenShift workflows. This runtime supports containers in dedicated virtual machines (VMs), providing improved workload isolation.

Ensure kernel isolation for each workload

You can run workloads that require custom kernel tuning (such as **sysctl**, scheduler changes, or cache tuning) and the creation of custom kernel modules (such as **out of tree** or special arguments).

Share the same workload across tenants

You can run workloads that support many users (tenants) from different organizations sharing the same OpenShift Container Platform cluster. The system also supports running third-party workloads from multiple vendors, such as container network functions (CNFs) and enterprise applications. Third-party CNFs, for example, may not want their custom settings interfering with packet tuning or with **sysctl** variables set by other applications. Running inside a completely isolated kernel is helpful in preventing "noisy neighbor" configuration problems.

Ensure proper isolation and sandboxing for testing software

You can run containerized workloads with known vulnerabilities or handle issues in an existing application. This isolation enables administrators to give developers administrative control over pods, which is useful when the developer wants to test or validate configurations beyond those an administrator would typically grant. Administrators can, for example, safely and securely delegate kernel packet filtering (eBPF) to developers. eBPF requires **CAP_ADMIN** or **CAP_BPF** privileges, and is therefore not allowed under a standard CRI-O configuration, as this would grant access to every process on the Container Host worker node. Similarly, administrators can grant access to intrusive tools such as **SystemTap**, or support the loading of custom kernel modules during their development.

Ensure default resource containment through VM boundaries

By default, OpenShift sandboxed containers manages resources such as CPU, memory, storage, and networking in a robust and secure way. Since OpenShift sandboxed containers deploys on VMs, additional layers of isolation and security give a finer-grained access control to the resource. For example, an errant container will not be able to assign more memory than is available to the VM. Conversely, a container that needs dedicated access to a network card or to a disk can take complete control over that device without getting any access to other devices.

1.2. COMPATIBILITY WITH OPENSIFT CONTAINER PLATFORM

The required functionality for the OpenShift Container Platform platform is supported by two main components:

- Kata Runtime: This includes Red Hat Enterprise Linux CoreOS (RHCOS) and [updates](#) with every OpenShift Container Platform release.
- OpenShift sandboxed containers Operator: Install the Operator using either the web console or OpenShift CLI (**oc**).

The OpenShift sandboxed containers Operator is a [Rolling Stream Operator](#), which means the latest version is the only supported version. It works with all currently supported versions of the OpenShift Container Platform. For more information, see [OpenShift Container Platform Life Cycle Policy](#) for additional details.

The Operator depends on the features that come with the RHCOS host and the environment it runs in.



NOTE

You must install Red Hat Enterprise Linux CoreOS (RHCOS) on the worker nodes. RHEL nodes are not supported.

The following compatibility matrix between OpenShift sandboxed containers and OpenShift Container Platform releases identifies compatible features and environments.

Table 1.1. Supported architectures

Architecture	OpenShift Container Platform version
x86_64	4.8 or later
s390x	4.14 or later

There are two ways to deploy Kata containers runtime:

- bare metal
- Peer pods

Peer pods technology began with OpenShift sandboxed containers 1.5 / OpenShift Container Platform 4.14, allowing the deployment of OpenShift sandboxed containers in public clouds.

Table 1.2. Feature availability by OpenShift version

Feature	Deployment method	OpenShift Container Platform 4.15	OpenShift Container Platform 4.16
Confidential Containers ^[1]	bare metal	No	No
	Peer pods	Developer Preview	Developer Preview
GPU support ^[2]	bare metal	No	No
	Peer pods	Developer Preview	Developer Preview

1. Confidential Containers are only supported on AMD SEV-SNP.
2. GPU functionality is not available on s390x.

Table 1.3. Supported platforms for OpenShift sandboxed containers

Platform	Peer pods	GPU	Confidential containers
AWS Cloud Computing Services	Yes	Developer Preview	No
Microsoft Azure Cloud Computing Services	Yes	Developer Preview	Developer Preview

Additional resources

- [Developer Preview Support Scope](#)
- [Deploying workloads on AWS](#)
- [Deploying workloads on Azure](#)
- [Deploying workloads on bare metal](#)

1.3. NODE ELIGIBILITY CHECKS

Before you deploy OpenShift sandboxed containers, you can check whether the nodes in your bare-metal cluster can run OpenShift sandboxed containers. The most common reason for node ineligibility is lack of virtualization support. If you run sandboxed workloads on ineligible nodes, you will experience errors.

High-level workflow

1. Install the Node Feature Discovery Operator.
2. Create the **NodeFeatureDiscovery** custom resource (CR).
3. Enable node eligibility checks when you create the **Kataconfig** CR. You can run node eligibility checks on all worker nodes or on selected nodes.

Additional resources

- [Installing the Node Feature Discovery Operator](#)

1.4. COMMON TERMS

The following terms are used throughout the documentation.

Sandbox

A sandbox is an isolated environment where programs can run. In a sandbox, you can run untested or untrusted programs without risking harm to the host machine or the operating system.

In the context of OpenShift sandboxed containers, sandboxing is achieved by running workloads in a different kernel using virtualization, providing enhanced control over the interactions between multiple workloads that run on the same host.

Pod

A pod is a construct that is inherited from Kubernetes and OpenShift Container Platform. It represents resources where containers can be deployed. Containers run inside of pods, and pods are used to specify resources that can be shared between multiple containers.

In the context of OpenShift sandboxed containers, a pod is implemented as a virtual machine. Several containers can run in the same pod on the same virtual machine.

OpenShift sandboxed containers Operator

An Operator is a software component that automates operations, which are actions that a human operator could do on the system.

The OpenShift sandboxed containers Operator is tasked with managing the lifecycle of sandboxed containers on a cluster. You can use the OpenShift sandboxed containers Operator to perform tasks such as the installation and removal of sandboxed containers, software updates, and status monitoring.

Kata Containers

Kata Containers is a core upstream project that is used to build OpenShift sandboxed containers. OpenShift sandboxed containers integrate Kata Containers with OpenShift Container Platform.

KataConfig

KataConfig objects represent configurations of sandboxed containers. They store information about the state of the cluster, such as the nodes on which the software is deployed.

Runtime class

A **RuntimeClass** object describes which runtime can be used to run a given workload. A runtime class that is named **kata** is installed and deployed by the OpenShift sandboxed containers Operator. The runtime class contains information about the runtime that describes resources that the runtime needs to operate, such as the [pod overhead](#).

Peer pod

A peer pod in OpenShift sandboxed containers extends the concept of a standard pod. Unlike a standard sandboxed container, where the virtual machine is created on the worker node itself, in a peer pod, the virtual machine is created through a remote hypervisor using any supported hypervisor or cloud provider API. The peer pod acts as a regular pod on the worker node, with its corresponding VM running elsewhere. The remote location of the VM is transparent to the user and is specified by the runtime class in the pod specification. The peer pod design circumvents the need for nested virtualization.

1.5. OPENSIFT SANDBOXED CONTAINERS OPERATOR

The OpenShift sandboxed containers Operator encapsulates all of the components from Kata containers. It manages installation, lifecycle, and configuration tasks.

The OpenShift sandboxed containers Operator is packaged in the [Operator bundle format](#) as two container images:

- The bundle image contains metadata and is required to make the operator OLM-ready.
- The second container image contains the actual controller that monitors and manages the **KataConfig** resource.

The OpenShift sandboxed containers Operator is based on the Red Hat Enterprise Linux CoreOS (RHCOS) extensions concept. RHCOS extensions are a mechanism to install optional OpenShift Container Platform software. The OpenShift sandboxed containers Operator uses this mechanism to deploy sandboxed containers on a cluster.

The sandboxed containers RHCOS extension contains RPMs for Kata, QEMU, and its dependencies. You can enable them by using the **MachineConfig** resources that the Machine Config Operator provides.

Additional resources

- [Adding extensions to RHCOS](#)

1.6. OPENSIFT VIRTUALIZATION

You can deploy OpenShift sandboxed containers on clusters with OpenShift Virtualization.

To run OpenShift Virtualization and OpenShift sandboxed containers at the same time, your virtual machines must be live migratable so that they do not block node reboots. See [About live migration](#) in the OpenShift Virtualization documentation for details.

1.7. STORAGE CONSIDERATIONS

1.7.1. Block volume support

OpenShift Container Platform can statically provision raw block volumes. These volumes do not have a file system, and can provide performance benefits for applications that either write to the disk directly or implement their own storage service.

You can use a local block device as persistent volume (PV) storage with OpenShift sandboxed containers. This block device can be provisioned using the Local Storage Operator (LSO).

The Local Storage Operator is not installed in OpenShift Container Platform by default. See [Installing the Local Storage Operator](#) for installation instructions.

Raw block volumes for OpenShift sandboxed containers are provisioned by specifying **volumeMode: Block** in the PV specification.

Block volume example

```
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "openshift-local-storage"
spec:
  nodeSelector:
    nodeSelectorTerms:
      - matchExpressions:
          - key: kubernetes.io/hostname
            operator: In
            values:
              - worker-0
  storageClassDevices:
    - storageClassName: "local-sc"
      forceWipeDevicesAndDestroyAllData: false
      volumeMode: Block 1
      devicePaths:
        - /path/to/device 2
```

- 1** **volumeMode** must be set to **Block** to indicate that this PV is a raw block volume.
- 2** Replace this value with your actual local disks filepath to the **LocalVolume** resource **by-id**. PVs are created for these local disks when the provisioner is deployed successfully. You must also use this path to label the node that uses the block device when deploying OpenShift sandboxed containers.

1.8. FIPS COMPLIANCE

OpenShift Container Platform is designed for Federal Information Processing Standards (FIPS) 140-2 and 140-3. When running Red Hat Enterprise Linux (RHEL) or Red Hat Enterprise Linux CoreOS (RHCOS) booted in FIPS mode, OpenShift Container Platform core components use the RHEL cryptographic libraries that have been submitted to NIST for FIPS 140-2/140-3 Validation on only the **x86_64**, **ppc64le**, and **s390x** architectures.

For more information about the NIST validation program, see [Cryptographic Module Validation Program](#). For the latest NIST status for the individual versions of RHEL cryptographic libraries that have been submitted for validation, see [Compliance Activities and Government Standards](#).

OpenShift sandboxed containers can be used on FIPS enabled clusters.

When running in FIPS mode, OpenShift sandboxed containers components, VMs, and VM images are adapted to comply with FIPS.



NOTE

FIPS compliance for OpenShift sandboxed containers only applies to the **kata** runtime class. The peer pod runtime class, **kata-remote**, is not yet fully supported and has not been tested for FIPS compliance.

FIPS compliance is one of the most critical components required in highly secure environments, to ensure that only supported cryptographic technologies are allowed on nodes.



IMPORTANT

The use of FIPS Validated / Modules in Process cryptographic libraries is only supported on OpenShift Container Platform deployments on the **x86_64** architecture.

To understand Red Hat's view of OpenShift Container Platform compliance frameworks, refer to the Risk Management and Regulatory Readiness chapter of the [OpenShift Security Guide Book](#).

CHAPTER 2. DEPLOYING WORKLOADS ON BARE METAL

You can deploy OpenShift sandboxed containers workloads on on-premise bare-metal servers with Red Hat Enterprise Linux CoreOS (RHCOS) installed on the worker nodes.



NOTE

- RHEL nodes are not supported.
- Nested virtualization is not supported.

You can use any installation method including [user-provisioned](#), [installer-provisioned](#), or [Assisted Installer](#) to deploy your cluster.

You can also install OpenShift sandboxed containers on Amazon Web Services (AWS) bare-metal instances. Bare-metal instances offered by other cloud providers are not supported.

Deployment workflow

You deploy OpenShift sandboxed containers workloads by performing the following steps:

1. Prepare your environment.
2. Create a **KataConfig** custom resource.
3. Configure your workload objects to use the **kata** runtime class.

2.1. PREPARING YOUR ENVIRONMENT

Perform the following steps to prepare your environment:

1. Ensure that your cluster has sufficient resources.
2. Install the OpenShift sandboxed containers Operator.
3. Optional: Configure [node eligibility checks](#) to ensure that your worker nodes support OpenShift sandboxed containers:
 - a. Install the Node Feature Discovery (NFD) Operator. See the [NFD Operator documentation](#) for details.
 - b. Create a **NodeFeatureDiscovery** custom resource (CR) to define the node configuration parameters that the NFD Operator checks.

2.1.1. Resource requirements

OpenShift sandboxed containers lets users run workloads on their OpenShift Container Platform clusters inside a sandboxed runtime (Kata). Each pod is represented by a virtual machine (VM). Each VM runs in a QEMU process and hosts a **kata-agent** process that acts as a supervisor for managing container workloads, and the processes running in those containers. Two additional processes add more overhead:

- **containerd-shim-kata-v2** is used to communicate with the pod.
- **virtiofsd** handles host file system access on behalf of the guest.

Each VM is configured with a default amount of memory. Additional memory is hot-plugged into the VM for containers that explicitly request memory.

A container running without a memory resource consumes free memory until the total memory used by the VM reaches the default allocation. The guest and its I/O buffers also consume memory.

If a container is given a specific amount of memory, then that memory is hot-plugged into the VM before the container starts.

When a memory limit is specified, the workload is terminated if it consumes more memory than the limit. If no memory limit is specified, the kernel running on the VM might run out of memory. If the kernel runs out of memory, it might terminate other processes on the VM.

Default memory sizes

The following table lists some the default values for resource allocation.

Resource	Value
Memory allocated by default to a virtual machine	2Gi
Guest Linux kernel memory usage at boot	~110Mi
Memory used by the QEMU process (excluding VM memory)	~30Mi
Memory used by the virtiofsd process (excluding VM I/O buffers)	~10Mi
Memory used by the containerd-shim-kata-v2 process	~20Mi
File buffer cache data after running dnf install on Fedora	~300Mi* [1]

File buffers appear and are accounted for in multiple locations:

- In the guest where it appears as file buffer cache.
- In the **virtiofsd** daemon that maps allowed user-space file I/O operations.
- In the QEMU process as guest memory.



NOTE

Total memory usage is properly accounted for by the memory utilization metrics, which only count that memory once.

[Pod overhead](#) describes the amount of system resources that a pod on a node uses. You can get the current pod overhead for the Kata runtime by using **oc describe runtimeclass kata** as shown below.

Example

-

```
$ oc describe runtimeclass kata
```

Example output

```
kind: RuntimeClass
apiVersion: node.k8s.io/v1
metadata:
  name: kata
overhead:
  podFixed:
    memory: "500Mi"
    cpu: "500m"
```

You can change the pod overhead by changing the **spec.overhead** field for a **RuntimeClass**. For example, if the configuration that you run for your containers consumes more than 350Mi of memory for the QEMU process and guest kernel data, you can alter the **RuntimeClass** overhead to suit your needs.



NOTE

The specified default overhead values are supported by Red Hat. Changing default overhead values is not supported and can result in technical issues.

When performing any kind of file system I/O in the guest, file buffers are allocated in the guest kernel. The file buffers are also mapped in the QEMU process on the host, as well as in the **virtiofsd** process.

For example, if you use 300Mi of file buffer cache in the guest, both QEMU and **virtiofsd** appear to use 300Mi additional memory. However, the same memory is used in all three cases. Therefore, the total memory usage is only 300Mi, mapped in three different places. This is correctly accounted for when reporting the memory utilization metrics.

2.1.2. Installing the OpenShift sandboxed containers Operator

You can install the OpenShift sandboxed containers Operator by using the OpenShift Container Platform web console or command line interface (CLI).

2.1.2.1. Installing the Operator by using the web console

You can install the OpenShift sandboxed containers Operator by using the Red Hat OpenShift Container Platform web console.

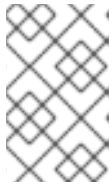
Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

- In the OpenShift Container Platform web console, navigate to **Operators** → **OperatorHub**.
- In the **Filter by keyword** field, type **OpenShift sandboxed containers**.
- Select the **OpenShift sandboxed containers Operator** tile and click **Install**.
- On the **Install Operator** page, select **stable** from the list of available **Update Channel** options.

5. Verify that **Operator recommended Namespace** is selected for **Installed Namespace**. This installs the Operator in the mandatory **openshift-sandboxed-containers-operator** namespace. If this namespace does not yet exist, it is automatically created.



NOTE

Attempting to install the OpenShift sandboxed containers Operator in a namespace other than **openshift-sandboxed-containers-operator** causes the installation to fail.

6. Verify that **Automatic** is selected for **Approval Strategy**. **Automatic** is the default value, and enables automatic updates to OpenShift sandboxed containers when a new z-stream release is available.
7. Click **Install**.

The OpenShift sandboxed containers Operator is now installed on your cluster.

Verification

1. Navigate to **Operators → Installed Operators**.
2. Verify that the OpenShift sandboxed containers Operator is displayed.

Additional resources

- [Using Operator Lifecycle Manager on restricted networks](#) .
- [Configuring proxy support in Operator Lifecycle Manager](#) for disconnected environments.

2.1.2.2. Installing the Operator by using the CLI

You can install the OpenShift sandboxed containers Operator by using the CLI.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a **Namespace.yaml** manifest file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sandboxed-containers-operator
```

2. Create the namespace by running the following command:

```
$ oc create -f Namespace.yaml
```

3. Create an **OperatorGroup.yaml** manifest file:

```

apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
  targetNamespaces:
    - openshift-sandboxed-containers-operator

```

4. Create the operator group by running the following command:

```
$ oc create -f OperatorGroup.yaml
```

5. Create a **Subscription.yaml** manifest file:

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
  channel: stable
  installPlanApproval: Automatic
  name: sandboxed-containers-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: sandboxed-containers-operator.v1.6.0

```

6. Create the subscription by running the following command:

```
$ oc create -f Subscription.yaml
```

The OpenShift sandboxed containers Operator is now installed on your cluster.

Verification

- Ensure that the Operator is correctly installed by running the following command:

```
$ oc get csv -n openshift-sandboxed-containers-operator
```

Example output

```

NAME                                DISPLAY                                VERSION    REPLACES
PHASE
openshift-sandboxed-containers  openshift-sandboxed-containers-operator  1.6.0    1.5.3
Succeeded

```

2.1.2.3. Additional resources

- [Using Operator Lifecycle Manager on restricted networks](#)
- [Configuring proxy support in Operator Lifecycle Manager](#) for disconnected environments

2.1.3. Creating the NodeFeatureDiscovery CR

You create a **NodeFeatureDiscovery** custom resource (CR) to define the configuration parameters that the Node Feature Discovery (NFD) Operator checks to determine that the worker nodes can support OpenShift sandboxed containers.



NOTE

To install the **kata** runtime on only selected worker nodes that you know are eligible, apply the **feature.node.kubernetes.io/runtime.kata=true** label to the selected nodes and set **checkNodeEligibility: true** in the **KataConfig** CR.

To install the **kata** runtime on all worker nodes, set **checkNodeEligibility: false** in the **KataConfig** CR.

In both these scenarios, you do not need to create the **NodeFeatureDiscovery** CR. You should only apply the **feature.node.kubernetes.io/runtime.kata=true** label manually if you are sure that the node is eligible to run OpenShift sandboxed containers.

The following procedure applies the **feature.node.kubernetes.io/runtime.kata=true** label to all eligible nodes and configures the **KataConfig** resource to check for node eligibility.

Prerequisites

- You have installed the NFD Operator.

Procedure

1. Create an **nfd.yaml** manifest file according to the following example:

```
apiVersion: nfd.openshift.io/v1
kind: NodeFeatureDiscovery
metadata:
  name: nfd-kata
  namespace: openshift-nfd
spec:
  workerConfig:
    configData: |
      sources:
        custom:
          - name: "feature.node.kubernetes.io/runtime.kata"
            matchOn:
              - cpuld: ["SSE4", "VMX"]
                loadedKMod: ["kvm", "kvm_intel"]
              - cpuld: ["SSE4", "SVM"]
                loadedKMod: ["kvm", "kvm_amd"]
# ...
```

2. Create the **NodeFeatureDiscovery** CR:

```
$ oc create -f nfd.yaml
```

The **NodeFeatureDiscovery** CR applies the **feature.node.kubernetes.io/runtime.kata=true** label to all qualifying worker nodes.

1. Create a **kata-config.yaml** manifest file according to the following example:

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: example-kataconfig
spec:
  checkNodeEligibility: true
```

2. Create the **KataConfig** CR:

```
$ oc create -f kata-config.yaml
```

Verification

- Verify that qualifying nodes in the cluster have the correct label applied:

```
$ oc get nodes --selector='feature.node.kubernetes.io/runtime.kata=true'
```

Example output

NAME	STATUS	ROLES	AGE	VERSION
compute-3.example.com	Ready	worker	4h38m	v1.25.0
compute-2.example.com	Ready	worker	4h35m	v1.25.0

2.2. DEPLOYING WORKLOADS BY USING THE WEB CONSOLE

You can deploy OpenShift sandboxed containers workloads by using the web console.

2.2.1. Creating a KataConfig custom resource

You must create a **KataConfig** custom resource (CR) to install **kata** as a **RuntimeClass** on your worker nodes.

The **kata** runtime class is installed on all worker nodes by default. If you want to install **kata** only on specific nodes, you can add labels to those nodes and then define the label in the **KataConfig** CR.

OpenShift sandboxed containers installs **kata** as a *secondary, optional* runtime on the cluster and not as the primary runtime.



IMPORTANT

Creating the **KataConfig** CR automatically reboots the worker nodes. The reboot can take from 10 to more than 60 minutes. The following factors might increase the reboot time:

- A larger OpenShift Container Platform deployment with a greater number of worker nodes.
- Activation of the BIOS and Diagnostics utility.
- Deployment on a hard disk drive rather than an SSD.
- Deployment on physical nodes such as bare metal, rather than on virtual nodes.
- A slow CPU and network.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- Optional: You have installed the Node Feature Discovery Operator if you want to enable node eligibility checks.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators → Installed Operators**.
2. Select the OpenShift sandboxed containers Operator.
3. On the **KataConfig** tab, click **Create KataConfig**.
4. Enter the following details:
 - **Name**: Optional: The default name is **example-kataconfig**.
 - **Labels**: Optional: Enter any relevant, identifying attributes to the **KataConfig** resource. Each label represents a key-value pair.
 - **checkNodeEligibility**: Optional: Select to use the Node Feature Discovery Operator (NFD) to detect node eligibility.
 - **kataConfigPoolSelector**. Optional: To install **kata** on selected nodes, add a match expression for the labels on the selected nodes:
 - a. Expand the **kataConfigPoolSelector** area.
 - b. In the **kataConfigPoolSelector** area, expand **matchExpressions**. This is a list of label selector requirements.
 - c. Click **Add matchExpressions**.
 - d. In the **Key** field, enter the label key the selector applies to.
 - e. In the **Operator** field, enter the key's relationship to the label values. Valid operators are **In**, **NotIn**, **Exists**, and **DoesNotExist**.

- f. Expand the **Values** area and then click **Add value**.
 - g. In the **Value** field, enter **true** or **false** for **key** label value.
 - **logLevel**: Define the level of log data retrieved for nodes with the **kata** runtime class.
5. Click **Create**. The **KataConfig** CR is created and installs the **kata** runtime class on the worker nodes.
Wait for the **kata** installation to complete and the worker nodes to reboot before verifying the installation.

Verification

1. On the **KataConfig** tab, click the **KataConfig** CR to view its details.
2. Click the **YAML** tab to view the **status** stanza.
The **status** stanza contains the **conditions** and **kataNodes** keys. The value of **status.kataNodes** is an array of nodes, each of which lists nodes in a particular state of **kata** installation. A message appears each time there is an update.
3. Click **Reload** to refresh the YAML.
When all workers in the **status.kataNodes** array display the values **installed** and **conditions.InProgress: False** with no specified reason, the **kata** is installed on the cluster.

See [KataConfig status messages](#) for details.

2.2.2. Configuring workload objects

You deploy an OpenShift sandboxed containers workload by configuring **kata** as the runtime class for the following pod-templated objects:

- **Pod** objects
- **ReplicaSet** objects
- **ReplicationController** objects
- **StatefulSet** objects
- **Deployment** objects
- **DeploymentConfig** objects



IMPORTANT

Do not deploy workloads in the **openshift-sandboxed-containers-operator** namespace. Create a dedicated namespace for these resources.

Prerequisites

- You have created a secret object for your provider.
- You have created a config map for your provider.
- You have created a **KataConfig** custom resource (CR).

Procedure

1. In the OpenShift Container Platform web console, navigate to **Workloads** → workload type, for example, **Pods**.
2. On the workload type page, click an object to view its details.
3. Click the **YAML** tab.
4. Add **spec.runtimeClassName: kata** to the manifest of each pod-templated workload object as in the following example:

```
apiVersion: v1
kind: <object>
# ...
spec:
  runtimeClassName: kata
# ...
```

OpenShift Container Platform creates the workload object and begins scheduling it.

Verification

- Inspect the **spec.runtimeClassName** field of a pod-templated object. If the value is **kata**, then the workload is running on OpenShift sandboxed containers, using peer pods.

2.3. DEPLOYING WORKLOADS BY USING THE COMMAND LINE

You can deploy OpenShift sandboxed containers workloads by using the command line.

2.3.1. Optional: Provisioning local block volumes by using the Local Storage Operator

Local block volumes for OpenShift sandboxed containers can be provisioned using the Local Storage Operator (LSO). The local volume provisioner looks for any block volume devices at the paths specified in the defined resource.

Prerequisites

- The Local Storage Operator is installed.
- You have a local disk that meets the following conditions:
 - It is attached to a node.
 - It is not mounted.
 - It does not contain partitions.

Procedure

1. Create the local volume resource. This resource must define the nodes and paths to the local volumes.

**NOTE**

Do not use different storage class names for the same device. Doing so will create multiple persistent volumes (PVs).

Example: Block

```

apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "openshift-local-storage" ❶
spec:
  nodeSelector: ❷
  nodeSelectorTerms:
    - matchExpressions:
      - key: kubernetes.io/hostname
        operator: In
        values:
          - ip-10-0-136-143
          - ip-10-0-140-255
          - ip-10-0-144-180
  storageClassDevices:
    - storageClassName: "local-sc" ❸
      forceWipeDevicesAndDestroyAllData: false ❹
      volumeMode: Block
      devicePaths: ❺
        - /path/to/device ❻

```

- ❶ The namespace where the Local Storage Operator is installed.
- ❷ Optional: A node selector containing a list of nodes where the local storage volumes are attached. This example uses the node hostnames, obtained from **oc get node**. If a value is not defined, then the Local Storage Operator will attempt to find matching disks on all available nodes.
- ❸ The name of the storage class to use when creating persistent volume objects.
- ❹ This setting defines whether or not to call **wipefs**, which removes partition table signatures (magic strings) making the disk ready to use for Local Storage Operator provisioning. No other data besides signatures is erased. The default is "false" (**wipefs** is not invoked). Setting **forceWipeDevicesAndDestroyAllData** to "true" can be useful in scenarios where previous data can remain on disks that need to be re-used. In these scenarios, setting this field to true eliminates the need for administrators to erase the disks manually.
- ❺ The path containing a list of local storage devices to choose from. You must use this path when deploying sandboxed container nodes on a block device.
- ❻ Replace this value with your actual local disks filepath to the **LocalVolume** resource **by-id**, such as **/dev/disk/by-id/wwn**. PVs are created for these local disks when the provisioner is deployed successfully.

2. Create the local volume resource in your OpenShift Container Platform cluster. Specify the file you just created:

```
$ oc create -f <local-volume>.yaml
```

- Verify that the provisioner was created and that the corresponding daemon sets were created:

```
$ oc get all -n openshift-local-storage
```

Example output

```

NAME                                READY STATUS RESTARTS AGE
pod/diskmaker-manager-9wzms         1/1   Running 0      5m43s
pod/diskmaker-manager-jgvjp         1/1   Running 0      5m43s
pod/diskmaker-manager-tbdsj         1/1   Running 0      5m43s
pod/local-storage-operator-7db4bd9f79-t6k87 1/1   Running 0      14m

NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP  PORT(S)
AGE
service/local-storage-operator-metrics ClusterIP      172.30.135.36 <none>
8383/TCP,8686/TCP 14m

NAME                                DESIRED CURRENT READY UP-TO-DATE AVAILABLE
NODE SELECTOR AGE
daemonset.apps/diskmaker-manager 3      3      3      3      3      <none>
5m43s

NAME                                READY UP-TO-DATE AVAILABLE AGE
deployment.apps/local-storage-operator 1/1   1      1      14m

NAME                                DESIRED CURRENT READY AGE
replicaset.apps/local-storage-operator-7db4bd9f79 1      1      1      14m

```

Note the **desired** and **current** number of daemon set processes. A **desired** count of **0** indicates that the label selectors were invalid.

- Verify that the persistent volumes were created:

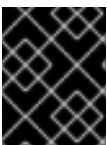
```
$ oc get pv
```

Example output

```

NAME          CAPACITY ACCESS MODES RECLAIM POLICY STATUS CLAIM
STORAGECLASS REASON AGE
local-pv-1cec77cf 100Gi RWO Delete Available local-sc 88m
local-pv-2ef7cd2a 100Gi RWO Delete Available local-sc
82m
local-pv-3fa1c73 100Gi RWO Delete Available local-sc 48m

```



IMPORTANT

Editing the **LocalVolume** object does not change existing persistent volumes because doing so might result in a destructive operation.

2.3.2. Optional: Deploying nodes on a block device

If you provisioned local block volumes for OpenShift sandboxed containers, you can choose to deploy nodes on any block device at the paths specified in the defined volume resource.

Prerequisites

- You provisioned a block device using the Local Storage Operator

Procedure

- Run the following command for each node you want to deploy using a block device:

```
$ oc debug node/worker-0 -- chcon -vt container_file_t /host/path/to/device
```

+ The **/path/to/device** must be the same path you defined when creating the local storage resource.

+ .Example output

```
system_u:object_r:container_file_t:s0 /host/path/to/device
```

2.3.3. Creating a KataConfig custom resource

You must create a **KataConfig** custom resource (CR) to install **kata** as a runtime class on your worker nodes.

Creating the **KataConfig** CR triggers the OpenShift sandboxed containers Operator to do the following:

- Install the required RHCOS extensions, such as QEMU and **kata-containers**, on your RHCOS node.
- Ensure that the **CRI-O** runtime is configured with the correct runtime handlers.
- Create a **RuntimeClass** CR named **kata** with a default configuration. This enables users to configure workloads to use **kata** as the runtime by referencing the CR in the **RuntimeClassName** field. This CR also specifies the resource overhead for the runtime.

OpenShift sandboxed containers installs **kata** as a *secondary, optional* runtime on the cluster and not as the primary runtime.

IMPORTANT

Creating the **KataConfig** CR automatically reboots the worker nodes. The reboot can take from 10 to more than 60 minutes. Factors that impede reboot time are as follows:

- A larger OpenShift Container Platform deployment with a greater number of worker nodes.
- Activation of the BIOS and Diagnostics utility.
- Deployment on a hard disk drive rather than an SSD.
- Deployment on physical nodes such as bare metal, rather than on virtual nodes.
- A slow CPU and network.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- Optional: You have installed the Node Feature Discovery Operator if you want to enable node eligibility checks.

Procedure

1. Create a **cluster-kataconfig.yaml** manifest file according to the following example:

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  checkNodeEligibility: false 1
  logLevel: info
```

- 1** Optional: Set `checkNodeEligibility` to **true** to run node eligibility checks.

2. Optional: To install **kata** on selected nodes, specify the node labels according to the following example:

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  kataConfigPoolSelector:
    matchLabels:
      <label_key>: '<label_value>' 1
  # ...
```

- 1** Specify the labels of the selected nodes.

3. Create the **KataConfig** CR:

```
$ oc create -f cluster-kataconfig.yaml
```

The new **KataConfig** CR is created and installs **kata** as a runtime class on the worker nodes.

Wait for the **kata** installation to complete and the worker nodes to reboot before verifying the installation.

Verification

- Monitor the installation progress by running the following command:

```
$ watch "oc describe kataconfig | sed -n /^Status:$/,/^Events/p"
```

When the status of all workers under **kataNodes** is **installed** and the condition **InProgress** is **False** without specifying a reason, the **kata** is installed on the cluster.

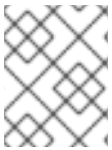
See [KataConfig status messages](#) for details.

2.3.4. Optional: Modifying pod overhead

Pod overhead describes the amount of system resources that a pod on a node uses. You can modify the pod overhead by changing the **spec.overhead** field for a **RuntimeClass** custom resource. For example, if the configuration that you run for your containers consumes more than 350Mi of memory for the QEMU process and guest kernel data, you can alter the **RuntimeClass** overhead to suit your needs.

When performing any kind of file system I/O in the guest, file buffers are allocated in the guest kernel. The file buffers are also mapped in the QEMU process on the host, as well as in the **virtiofsd** process.

For example, if you use 300Mi of file buffer cache in the guest, both QEMU and **virtiofsd** appear to use 300Mi additional memory. However, the same memory is being used in all three cases. Therefore, the total memory usage is only 300Mi, mapped in three different places. This is correctly accounted for when reporting the memory utilization metrics.



NOTE

The default values are supported by Red Hat. Changing default overhead values is not supported and can result in technical issues.

Procedure

1. Obtain the **RuntimeClass** object by running the following command:

```
$ oc describe runtimeclass kata
```

2. Update the **overhead.podFixed.memory** and **cpu** values and save as **RuntimeClass.yaml**:

```
kind: RuntimeClass
apiVersion: node.k8s.io/v1
metadata:
  name: kata
overhead:
  podFixed:
    memory: "500Mi"
    cpu: "500m"
```

2.3.5. Configuring workload objects

You deploy an OpenShift sandboxed containers workload by configuring **kata** as the runtime class for the following pod-templated objects:

- **Pod** objects
- **ReplicaSet** objects
- **ReplicationController** objects
- **StatefulSet** objects
- **Deployment** objects

- **DeploymentConfig** objects



IMPORTANT

Do not deploy workloads in the **openshift-sandboxed-containers-operator** namespace. Create a dedicated namespace for these resources.

Prerequisites

- You have created a secret object for your provider.
- You have created a config map for your provider.
- You have created a **KataConfig** custom resource (CR).

Procedure

1. Add **spec.runtimeClassName: kata** to the manifest of each pod-templated workload object as in the following example:

```
apiVersion: v1
kind: <object>
# ...
spec:
  runtimeClassName: kata
# ...
```

OpenShift Container Platform creates the workload object and begins scheduling it.

Verification

- Inspect the **spec.runtimeClassName** field of a pod-templated object. If the value is **kata**, then the workload is running on OpenShift sandboxed containers, using peer pods.

CHAPTER 3. DEPLOYING WORKLOADS ON PUBLIC CLOUD

You can deploy OpenShift sandboxed containers workloads on AWS Cloud Computing Services and Microsoft Azure Cloud Computing Services.

Cluster requirements

- You have installed Red Hat OpenShift Container Platform 4.13 or later.
- Your cluster has at least one worker node.

3.1. DEPLOYING WORKLOADS ON AWS

You can deploy OpenShift sandboxed containers workloads on AWS Cloud Computing Services by using the OpenShift Container Platform web console or the command line interface (CLI).

Deployment workflow

1. Enable ports.
2. Create a secret for AWS.
3. Create a config map for AWS.
4. Create a **KataConfig** custom resource.
5. Optional: Modify the peer pod VM limit per node.
6. Configure your workload objects to use the **kata-remote** runtime class.

3.1.1. Preparing your environment

Perform the following steps to prepare your environment:

1. Ensure that your cluster has sufficient resources.
2. Install the OpenShift sandboxed containers Operator.
3. Enable ports 15150 and 9000 to allow internal communication with peer pods.

3.1.1.1. Resource requirements

Peer pod virtual machines (VMs) require resources in two locations:

- The worker node. The worker node stores metadata, Kata shim resources (**containerd-shim-kata-v2**), remote-hypervisor resources (**cloud-api-adaptor**), and the tunnel setup between the worker nodes and the peer pod VM.
- The cloud instance. This is the actual peer pod VM running in the cloud.

The CPU and memory resources used in the Kubernetes worker node are handled by the [pod overhead](#) included in the RuntimeClass (**kata-remote**) definition used for creating peer pods.

The total number of peer pod VMs running in the cloud is defined as Kubernetes Node extended resources. This limit is per node and is set by the **limit** attribute in the **peerpodConfig** custom resource (CR).

The **peerpodConfig** CR, named **peerpodconfig-openshift**, is created when you create the **kataConfig** CR and enable peer pods, and is located in the **openshift-sandboxed-containers-operator** namespace.

The following **peerpodConfig** CR example displays the default **spec** values:

```
apiVersion: confidentialcontainers.org/v1alpha1
kind: PeerPodConfig
metadata:
  name: peerpodconfig-openshift
  namespace: openshift-sandboxed-containers-operator
spec:
  cloudSecretName: peer-pods-secret
  configMapName: peer-pods-cm
  limit: "10" 1
  nodeSelector:
    node-role.kubernetes.io/kata-oc: ""
```

1 The default limit is 10 VMs per node.

The extended resource is named **kata.peerpods.io/vm**, and enables the Kubernetes scheduler to handle capacity tracking and accounting.

You can edit the limit per node based on the requirements for your environment. See "Modifying the VM limit per node in peer pods" for more information.

A [mutating webhook](#) adds the extended resource **kata.peerpods.io/vm** to the pod specification. It also removes any resource-specific entries from the pod specification, if present. This enables the Kubernetes scheduler to account for these extended resources, ensuring the peer pod is only scheduled when resources are available.

The mutating webhook modifies a Kubernetes pod as follows:

- The mutating webhook checks the pod for the expected **RuntimeClassName** value, specified in the **TARGET_RUNTIME_CLASS** environment variable. If the value in the pod specification does not match the value in the **TARGET_RUNTIME_CLASS**, the webhook exits without modifying the pod.
- If the **RuntimeClassName** values match, the webhook makes the following changes to the pod spec:
 1. The webhook removes every resource specification from the **resources** field of all containers and init containers in the pod.
 2. The webhook adds the extended resource (**kata.peerpods.io/vm**) to the spec by modifying the resources field of the first container in the pod. The extended resource **kata.peerpods.io/vm** is used by the Kubernetes scheduler for accounting purposes.



NOTE

The mutating webhook excludes specific system namespaces in OpenShift Container Platform from mutation. If a peer pod is created in those system namespaces, then resource accounting using Kubernetes extended resources does not work unless the pod spec includes the extended resource.

As a best practice, define a cluster-wide policy to only allow peer pod creation in specific namespaces.

3.1.1.2. Enabling ports for AWS

You must enable ports 15150 and 9000 to allow internal communication with peer pods running on AWS.

Prerequisites

- You have installed the OpenShift sandboxed containers Operator.
- You have installed the AWS command line tool.
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Log in to your OpenShift Container Platform cluster and retrieve the instance ID:

```
$ INSTANCE_ID=$(oc get nodes -l 'node-role.kubernetes.io/worker' -o
  jsonpath='{.items[0].spec.providerID}' | sed 's#[^ ]*/##g')
```

2. Retrieve the AWS region:

```
$ AWS_REGION=$(oc get infrastructure/cluster -o
  jsonpath='{.status.platformStatus.aws.region}')
```

3. Retrieve the security group IDs and store them in an array:

```
$ AWS_SG_IDS=$(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --query
  'Reservations[*].Instances[*].SecurityGroups[*].GroupId' --output text --region
  $AWS_REGION))
```

4. For each security group ID, authorize the peer pods shim to access kata-agent communication, and set up the peer pods tunnel:

```
$ for AWS_SG_ID in "${AWS_SG_IDS[@]}"; do
    aws ec2 authorize-security-group-ingress --group-id $AWS_SG_ID --protocol tcp --port
    15150 --source-group $AWS_SG_ID --region $AWS_REGION

    aws ec2 authorize-security-group-ingress --group-id $AWS_SG_ID --protocol tcp --port
    9000 --source-group $AWS_SG_ID --region $AWS_REGION
done
```

The ports are now enabled.

3.1.1.3. Installing the OpenShift sandboxed containers Operator

You can install the OpenShift sandboxed containers Operator by using the OpenShift Container Platform web console or command line interface (CLI).

3.1.1.3.1. Installing the Operator by using the web console

You can install the OpenShift sandboxed containers Operator by using the Red Hat OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators** → **OperatorHub**.
2. In the **Filter by keyword** field, type **OpenShift sandboxed containers**.
3. Select the **OpenShift sandboxed containers Operator** tile and click **Install**.
4. On the **Install Operator** page, select **stable** from the list of available **Update Channel** options.
5. Verify that **Operator recommended Namespace** is selected for **Installed Namespace**. This installs the Operator in the mandatory **openshift-sandboxed-containers-operator** namespace. If this namespace does not yet exist, it is automatically created.



NOTE

Attempting to install the OpenShift sandboxed containers Operator in a namespace other than **openshift-sandboxed-containers-operator** causes the installation to fail.

6. Verify that **Automatic** is selected for **Approval Strategy**. **Automatic** is the default value, and enables automatic updates to OpenShift sandboxed containers when a new z-stream release is available.
7. Click **Install**.

The OpenShift sandboxed containers Operator is now installed on your cluster.

Verification

1. Navigate to **Operators** → **Installed Operators**.
2. Verify that the OpenShift sandboxed containers Operator is displayed.

Additional resources

- [Using Operator Lifecycle Manager on restricted networks](#) .
- [Configuring proxy support in Operator Lifecycle Manager](#) for disconnected environments.

3.1.1.3.2. Installing the Operator by using the CLI

You can install the OpenShift sandboxed containers Operator by using the CLI.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a **Namespace.yaml** manifest file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sandboxed-containers-operator
```

2. Create the namespace by running the following command:

```
$ oc create -f Namespace.yaml
```

3. Create an **OperatorGroup.yaml** manifest file:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
  targetNamespaces:
  - openshift-sandboxed-containers-operator
```

4. Create the operator group by running the following command:

```
$ oc create -f OperatorGroup.yaml
```

5. Create a **Subscription.yaml** manifest file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
  channel: stable
  installPlanApproval: Automatic
  name: sandboxed-containers-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: sandboxed-containers-operator.v1.6.0
```

6. Create the subscription by running the following command:

```
$ oc create -f Subscription.yaml
```

The OpenShift sandboxed containers Operator is now installed on your cluster.

Verification

- Ensure that the Operator is correctly installed by running the following command:

```
$ oc get csv -n openshift-sandboxed-containers-operator
```

Example output

```
NAME                                DISPLAY                                VERSION  REPLACES
PHASE
openshift-sandboxed-containers  openshift-sandboxed-containers-operator  1.6.0    1.5.3
Succeeded
```

3.1.1.3.3. Additional resources

- [Using Operator Lifecycle Manager on restricted networks](#)
- [Configuring proxy support in Operator Lifecycle Manager](#) for disconnected environments

3.1.2. Deploying workloads by using the web console

You can deploy OpenShift sandboxed containers workloads by using the web console.

3.1.2.1. Creating a secret

You must create a **Secret** object on your OpenShift Container Platform cluster. The secret stores cloud provider credentials for creating the pod virtual machine (VM) image and peer pod instances. By default, the OpenShift sandboxed containers Operator creates the secret based on the credentials used to create the cluster. However, you can manually create a secret that uses different credentials.

Prerequisites

- **AWS_ACCESS_KEY_ID**
- **AWS_SECRET_ACCESS_KEY**

You can generate these values in the AWS console.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators → Installed Operators**.
2. Click the OpenShift sandboxed containers Operator tile.
3. Click the Import icon (+) on the top right corner.
4. In the **Import YAML** window, paste the following YAML manifest:

```
apiVersion: v1
```

```

kind: Secret
metadata:
  name: peer-pods-secret
  namespace: openshift-sandboxed-containers-operator
type: Opaque
stringData:
  AWS_ACCESS_KEY_ID: "<aws_access_key>" ❶
  AWS_SECRET_ACCESS_KEY: "<aws_secret_access_key>" ❷

```

- ❶ Specify the **AWS_ACCESS_KEY_ID** value.
- ❷ Specify the **AWS_SECRET_ACCESS_KEY** value.

5. Click **Save** to apply the changes.



NOTE

If you update the peer pods secret, you must restart the **peerpodconfig-ctrl-cao-daemon** DaemonSet to apply the changes.

After you update the secret, click **Save** to apply the changes. Then restart the **cloud-api-adaptor** pods by running the following command:

```
$ oc set env ds/peerpodconfig-ctrl-cao-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

Restarting a daemon set recreates peer pods. It does not update existing pods.

Verification

- Navigate to **Workloads** → **Secrets** to view the secret.

3.1.2.2. Creating a config map

You must create a config map on your OpenShift Container Platform cluster for your cloud provider.

You must set the Amazon Machine Image (AMI) ID. You can retrieve this value before you create the config map.

Procedure

1. Obtain the following values from your AWS instance:
 - a. Retrieve and record the instance ID:

```
$ INSTANCE_ID=$(oc get nodes -l 'node-role.kubernetes.io/worker' -o jsonpath='{.items[0].spec.providerID}' | sed 's#[^ ]*/##g')
```

This is used to retrieve other values for the secret object.

- b. Retrieve and record the AWS region:

```
$ AWS_REGION=$(oc get infrastructure/cluster -o
jsonpath='{.status.platformStatus.aws.region}') && echo "AWS_REGION:
\"$AWS_REGION\""
```

- c. Retrieve and record the AWS subnet ID:

```
$ AWS_SUBNET_ID=$(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --
query 'Reservations[*].Instances[*].SubnetId' --region ${AWS_REGION} --output text) &&
echo "AWS_SUBNET_ID: \"$AWS_SUBNET_ID\""
```

- d. Retrieve and record the AWS VPC ID:

```
$ AWS_VPC_ID=$(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --query
'Reservations[*].Instances[*].VpcId' --region ${AWS_REGION} --output text) && echo
"AWS_VPC_ID: \"$AWS_VPC_ID\""
```

- e. Retrieve and record the AWS security group IDs:

```
$ AWS_SG_IDS=$(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --query
'Reservations[*].Instances[*].SecurityGroups[*].GroupId' --region ${AWS_REGION} --
output text)
&& echo "AWS_SG_IDS: \"$AWS_SG_IDS\""
```

- In the OpenShift Container Platform web console, navigate to **Operators → Installed Operators**.
- Select the OpenShift sandboxed containers Operator from the list of operators.
- Click the Import icon (+) in the top right corner.
- In the **Import YAML** window, paste the following YAML manifest:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: peer-pods-cm
  namespace: openshift-sandboxed-containers-operator
data:
  CLOUD_PROVIDER: "aws"
  VXLAN_PORT: "9000"
  PODVM_INSTANCE_TYPE: "t3.medium" 1
  PODVM_INSTANCE_TYPES: "t2.small,t2.medium,t3.large" 2
  PROXY_TIMEOUT: "5m"
  DISABLECVM: "true"
  PODVM_AMI_ID: "<podvm_ami_id>" 3
  AWS_REGION: "<aws_region>" 4
  AWS_SUBNET_ID: "<aws_subnet_id>" 5
  AWS_VPC_ID: "<aws_vpc_id>" 6
  AWS_SG_IDS: "<aws_sg_ids>" 7
```

- Defines the default instance type that is used when a type is not defined in the workload.
- Lists all of the instance types you can specify when creating the pod. This allows you to define smaller instance types for workloads that need less memory and fewer CPUs or

some smaller instance types for workloads that need less memory and fewer CPUs or larger instance types for larger workloads.

- 3 Optional: By default, this value is populated when you run the **KataConfig** CR, using an AMI ID based on your cluster credentials. If you create your own AMI, specify the correct AMI ID.
- 4 Specify the **AWS_REGION** value you retrieved.
- 5 Specify the **AWS_SUBNET_ID** value you retrieved.
- 6 Specify the **AWS_VPC_ID** value you retrieved.
- 7 Specify the **AWS_SG_IDS** value you retrieved.

6. Click **Save** to apply the changes.

A config map is created for your cloud provider.



NOTE

If you update the peer pods config map, you must restart the **peerpodconfig-ctrl-caa-daemon** daemonset to apply the changes.

After you update the config map, click **Save** to apply the changes. Then restart the **cloud-api-adaptor** pods by running the following command:

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

Restarting the daemonset recreates the peer pods. It does not update the existing pods.

Verification

- Navigate to **Workloads** → **ConfigMaps** to view the new config map.

3.1.2.3. Creating a KataConfig custom resource

You must create a **KataConfig** custom resource (CR) to install **kata-remote** as a **RuntimeClass** on your worker nodes.

The **kata-remote** runtime class is installed on all worker nodes by default. If you want to install **kata-remote** only on specific nodes, you can add labels to those nodes and then define the label in the **KataConfig** CR.

OpenShift sandboxed containers installs **kata-remote** as a *secondary, optional* runtime on the cluster and not as the primary runtime.



IMPORTANT

Creating the **KataConfig** CR automatically reboots the worker nodes. The reboot can take from 10 to more than 60 minutes. The following factors might increase the reboot time:

- A larger OpenShift Container Platform deployment with a greater number of worker nodes.
- Activation of the BIOS and Diagnostics utility.
- Deployment on a hard disk drive rather than an SSD.
- Deployment on physical nodes such as bare metal, rather than on virtual nodes.
- A slow CPU and network.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators → Installed Operators**.
2. Select the OpenShift sandboxed containers Operator.
3. On the **KataConfig** tab, click **Create KataConfig**.
4. Enter the following details:
 - **Name:** Optional: The default name is **example-kataconfig**.
 - **Labels:** Optional: Enter any relevant, identifying attributes to the **KataConfig** resource. Each label represents a key-value pair.
 - **enablePeerPods:** Select for public cloud, IBM Z®, and IBM® LinuxONE deployments.
 - **kataConfigPoolSelector.** Optional: To install **kata-remote** on selected nodes, add a match expression for the labels on the selected nodes:
 - a. Expand the **kataConfigPoolSelector** area.
 - b. In the **kataConfigPoolSelector** area, expand **matchExpressions**. This is a list of label selector requirements.
 - c. Click **Add matchExpressions**.
 - d. In the **Key** field, enter the label key the selector applies to.
 - e. In the **Operator** field, enter the key's relationship to the label values. Valid operators are **In**, **NotIn**, **Exists**, and **DoesNotExist**.
 - f. Expand the **Values** area and then click **Add value**.
 - g. In the **Value** field, enter **true** or **false** for **key** label value.

- **LogLevel**: Define the level of log data retrieved for nodes with the **kata-remote** runtime class.
5. Click **Create**. The **KataConfig** CR is created and installs the **kata-remote** runtime class on the worker nodes.
Wait for the **kata-remote** installation to complete and the worker nodes to reboot before verifying the installation.

Verification

1. On the **KataConfig** tab, click the **KataConfig** CR to view its details.
2. Click the **YAML** tab to view the **status** stanza.
The **status** stanza contains the **conditions** and **kataNodes** keys. The value of **status.kataNodes** is an array of nodes, each of which lists nodes in a particular state of **kata-remote** installation. A message appears each time there is an update.
3. Click **Reload** to refresh the YAML.
When all workers in the **status.kataNodes** array display the values **installed** and **conditions.InProgress: False** with no specified reason, the **kata-remote** is installed on the cluster.

See [KataConfig status messages](#) for details.

3.1.2.3.1. Optional: Verifying the pod VM image

After **kata-remote** is installed on your cluster, the OpenShift sandboxed containers Operator creates a pod VM image, which is used to create peer pods. This process can take a long time because the image is created on the cloud instance. You can verify that the pod VM image was created successfully by checking the config map that you created for the cloud provider.

Procedure

1. Navigate to **Workloads** → **ConfigMaps**.
2. Click the provider config map to view its details.
3. Click the **YAML** tab.
4. Check the **status** stanza of the YAML file.
If the **PODVM_AMI_ID** parameter is populated, the pod VM image was created successfully.

Troubleshooting

1. Retrieve the events log by running the following command:

```
$ oc get events -n openshift-sandboxed-containers-operator --field-selector
involvedObject.name=osc-podvm-image-creation
```

2. Retrieve the job log by running the following command:

```
$ oc logs -n openshift-sandboxed-containers-operator jobs/osc-podvm-image-creation
```

If you cannot resolve the issue, submit a Red Hat Support case and attach the output of both logs.

3.1.2.4. Optional: Modifying the number of peer pod VMs per node

You can change the limit of peer pod virtual machines (VMs) per node by editing the **peerpodConfig** custom resource (CR).

Procedure

1. Check the current limit by running the following command:

```
$ oc get peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-operator \
-o jsonpath='{.spec.limit}'
```

2. Modify the **limit** attribute of the **peerpodConfig** CR by running the following command:

```
$ oc patch peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-operator \
--type merge --patch '{"spec":{"limit":<value>}}' 1
```

- 1** Replace <value> with the limit you want to define.

3.1.2.5. Configuring workload objects

You deploy an OpenShift sandboxed containers workload by configuring **kata-remote** as the runtime class for the following pod-templated objects:

- **Pod** objects
- **ReplicaSet** objects
- **ReplicationController** objects
- **StatefulSet** objects
- **Deployment** objects
- **DeploymentConfig** objects



IMPORTANT

Do not deploy workloads in the **openshift-sandboxed-containers-operator** namespace. Create a dedicated namespace for these resources.

You can define whether the workload should be deployed using the default instance type, which you defined in the config map, by adding an annotation to the YAML file.

If you do not want to define the instance type manually, you can add an annotation to use an automatic instance type, based on the memory available.

Prerequisites

- You have created a secret object for your provider.
- You have created a config map for your provider.

- You have created a **KataConfig** custom resource (CR).

Procedure

- In the OpenShift Container Platform web console, navigate to **Workloads** → workload type, for example, **Pods**.
- On the workload type page, click an object to view its details.
- Click the **YAML** tab.
- Add **spec.runtimeClassName: kata-remote** to the manifest of each pod-templated workload object as in the following example:

```
apiVersion: v1
kind: <object>
# ...
spec:
  runtimeClassName: kata-remote
# ...
```

- Add an annotation to the pod-templated object to use a manually defined instance type or an automatic instance type:

- To use a manually defined instance type, add the following annotation:

```
apiVersion: v1
kind: <object>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.machine_type: t3.medium 1
# ...
```

- 1** Specify the instance type that you defined in the config map.

- To use an automatic instance type, add the following annotations:

```
apiVersion: v1
kind: <Pod>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.default_vcpus: <vcpus>
    io.katacontainers.config.hypervisor.default_memory: <memory>
# ...
```

Define the amount of memory available for the workload to use. The workload will run on an automatic instance type based on the amount of memory available.

- Click **Save** to apply the changes.
OpenShift Container Platform creates the workload object and begins scheduling it.

Verification

- Inspect the **spec.runtimeClassName** field of a pod-templated object. If the value is **kata-remote**, then the workload is running on OpenShift sandboxed containers, using peer pods.

3.1.3. Deploying workloads by using the command line

You can deploy OpenShift sandboxed containers workloads by using the command line.

3.1.3.1. Creating a secret

You must create a **Secret** object on your OpenShift Container Platform cluster. The secret stores cloud provider credentials for creating the pod virtual machine (VM) image and peer pod instances. By default, the OpenShift sandboxed containers Operator creates the secret based on the credentials used to create the cluster. However, you can manually create a secret that uses different credentials.

Prerequisites

- **AWS_ACCESS_KEY_ID**
- **AWS_SECRET_ACCESS_KEY**

You can generate these values in the AWS console.

Procedure

1. Create a **peer-pods-secret.yaml** manifest file according to the following example:

```
apiVersion: v1
kind: Secret
metadata:
  name: peer-pods-secret
  namespace: openshift-sandboxed-containers-operator
type: Opaque
stringData:
  AWS_ACCESS_KEY_ID: "<aws_access_key>" 1
  AWS_SECRET_ACCESS_KEY: "<aws_secret_access_key>" 2
```

- 1 Specify the **AWS_ACCESS_KEY_ID** value.
- 2 Specify the **AWS_SECRET_ACCESS_KEY** value.

2. Create the **secret** object by applying the manifest:

```
$ oc apply -f peer-pods-secret.yaml
```



NOTE

If you update the peer pods secret, you must restart the **peerpodconfig-ctrl-cao-daemon** DaemonSet to apply the changes.

After you update the secret, apply the manifest. Then restart the **cloud-api-adaptor** pods by running the following command:

```
$ oc set env ds/peerpodconfig-ctrl-cao-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

Restarting a daemon set recreates peer pods. It does not update existing pods.

3.1.3.2. Creating a config map

You must create a config map on your OpenShift Container Platform cluster for your cloud provider.

You must set the Amazon Machine Image (AMI) ID. You can retrieve this value before you create the config map.

Procedure

1. Obtain the following values from your AWS instance:

- a. Retrieve and record the instance ID:

```
$ INSTANCE_ID=$(oc get nodes -l 'node-role.kubernetes.io/worker' -o jsonpath='{.items[0].spec.providerID}' | sed 's#[^ ]*/##g')
```

This is used to retrieve other values for the secret object.

- b. Retrieve and record the AWS region:

```
$ AWS_REGION=$(oc get infrastructure/cluster -o jsonpath='{.status.platformStatus.aws.region}') && echo "AWS_REGION: \"$AWS_REGION\""
```

- c. Retrieve and record the AWS subnet ID:

```
$ AWS_SUBNET_ID=$(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --query 'Reservations[*].Instances[*].SubnetId' --region ${AWS_REGION} --output text) && echo "AWS_SUBNET_ID: \"$AWS_SUBNET_ID\""
```

- d. Retrieve and record the AWS VPC ID:

```
$ AWS_VPC_ID=$(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --query 'Reservations[*].Instances[*].VpcId' --region ${AWS_REGION} --output text) && echo "AWS_VPC_ID: \"$AWS_VPC_ID\""
```

- e. Retrieve and record the AWS security group IDs:

```
$ AWS_SG_IDS=$(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --query 'Reservations[*].Instances[*].SecurityGroups[*].GroupId' --region ${AWS_REGION} --output text)
```

```
&& echo "AWS_SG_IDS: \"\$AWS_SG_IDS\""
```

2. Create a **peer-pods-cm.yaml** manifest according to the following example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: peer-pods-cm
  namespace: openshift-sandboxed-containers-operator
data:
  CLOUD_PROVIDER: "aws"
  VXLAN_PORT: "9000"
  PODVM_INSTANCE_TYPE: "t3.medium" 1
  PODVM_INSTANCE_TYPES: "t2.small,t2.medium,t3.large" 2
  PROXY_TIMEOUT: "5m"
  DISABLECVM: "true"
  PODVM_AMI_ID: "<podvm_ami_id>" 3
  AWS_REGION: "<aws_region>" 4
  AWS_SUBNET_ID: "<aws_subnet_id>" 5
  AWS_VPC_ID: "<aws_vpc_id>" 6
  AWS_SG_IDS: "<aws_sg_ids>" 7
```

- 1 Defines the default instance type that is used when a type is not defined in the workload.
- 2 Lists all of the instance types you can specify when creating the pod. This allows you to define smaller instance types for workloads that need less memory and fewer CPUs or larger instance types for larger workloads.
- 3 Optional: By default, this value is populated when you run the **KataConfig** CR, using an AMI ID based on your cluster credentials. If you create your own AMI, specify the correct AMI ID.
- 4 Specify the **AWS_REGION** value you retrieved.
- 5 Specify the **AWS_SUBNET_ID** value you retrieved.
- 6 Specify the **AWS_VPC_ID** value you retrieved.
- 7 Specify the **AWS_SG_IDS** value you retrieved.

3. Apply the manifest to create a config map:

```
$ oc apply -f peer-pods-cm.yaml
```

A config map is created for your cloud provider.



NOTE

If you update the peer pods config map, you must restart the **peerpodconfig-ctrl-cao-daemon** daemonset to apply the changes.

After you update the config map, apply the manifest. Then restart the **cloud-api-adaptor** pods by running the following command:

```
$ oc set env ds/peerpodconfig-ctrl-cao-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

Restarting the daemonset recreates the peer pods. It does not update the existing pods.

3.1.3.3. Creating a KataConfig custom resource

You must create a **KataConfig** custom resource (CR) to install **kata-remote** as a runtime class on your worker nodes.

Creating the **KataConfig** CR triggers the OpenShift sandboxed containers Operator to do the following:

- Create a **RuntimeClass** CR named **kata-remote** with a default configuration. This enables users to configure workloads to use **kata-remote** as the runtime by referencing the CR in the **RuntimeClassName** field. This CR also specifies the resource overhead for the runtime.

OpenShift sandboxed containers installs **kata-remote** as a *secondary, optional* runtime on the cluster and not as the primary runtime.



IMPORTANT

Creating the **KataConfig** CR automatically reboots the worker nodes. The reboot can take from 10 to more than 60 minutes. Factors that impede reboot time are as follows:

- A larger OpenShift Container Platform deployment with a greater number of worker nodes.
- Activation of the BIOS and Diagnostics utility.
- Deployment on a hard disk drive rather than an SSD.
- Deployment on physical nodes such as bare metal, rather than on virtual nodes.
- A slow CPU and network.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a **cluster-kataconfig.yaml** manifest file according to the following example:

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
```

```

name: cluster-kataconfig
spec:
  enablePeerPods: true
  logLevel: info

```

- Optional: To install **kata-remote** on selected nodes, specify the node labels according to the following example:

```

apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  kataConfigPoolSelector:
    matchLabels:
      <label_key>: '<label_value>' 1
# ...

```

- Specify the labels of the selected nodes.

- Create the **KataConfig** CR:

```
$ oc create -f cluster-kataconfig.yaml
```

The new **KataConfig** CR is created and installs **kata-remote** as a runtime class on the worker nodes.

Wait for the **kata-remote** installation to complete and the worker nodes to reboot before verifying the installation.

Verification

- Monitor the installation progress by running the following command:

```
$ watch "oc describe kataconfig | sed -n /^Status:$/,/^Events/p"
```

When the status of all workers under **kataNodes** is **installed** and the condition **InProgress** is **False** without specifying a reason, the **kata-remote** is installed on the cluster.

See [KataConfig status messages](#) for details.

3.1.3.3.1. Optional: Verifying the pod VM image

After **kata-remote** is installed on your cluster, the OpenShift sandboxed containers Operator creates a pod VM image, which is used to create peer pods. This process can take a long time because the image is created on the cloud instance. You can verify that the pod VM image was created successfully by checking the config map that you created for the cloud provider.

Procedure

- Obtain the config map you created for the peer pods:

```
$ oc get configmap peer-pods-cm -n openshift-sandboxed-containers-operator -o yaml
```

2. Check the **status** stanza of the YAML file.
If the **PODVM_AMI_ID** parameter is populated, the pod VM image was created successfully.

Troubleshooting

1. Retrieve the events log by running the following command:

```
$ oc get events -n openshift-sandboxed-containers-operator --field-selector
involvedObject.name=osc-podvm-image-creation
```

2. Retrieve the job log by running the following command:

```
$ oc logs -n openshift-sandboxed-containers-operator jobs/osc-podvm-image-creation
```

If you cannot resolve the issue, submit a Red Hat Support case and attach the output of both logs.

3.1.3.4. Optional: Modifying the number of peer pod VMs per node

You can change the limit of peer pod virtual machines (VMs) per node by editing the **peerpodConfig** custom resource (CR).

Procedure

1. Check the current limit by running the following command:

```
$ oc get peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-operator \
-o jsonpath='{.spec.limit}'
```

2. Modify the **limit** attribute of the **peerpodConfig** CR by running the following command:

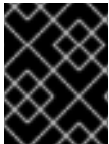
```
$ oc patch peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-
operator \
--type merge --patch '{"spec":{"limit": "<value>"}}' 1
```

- 1 Replace <value> with the limit you want to define.

3.1.3.5. Configuring workload objects

You deploy an OpenShift sandboxed containers workload by configuring **kata-remote** as the runtime class for the following pod-templated objects:

- **Pod** objects
- **ReplicaSet** objects
- **ReplicationController** objects
- **StatefulSet** objects
- **Deployment** objects
- **DeploymentConfig** objects



IMPORTANT

Do not deploy workloads in the **openshift-sandboxed-containers-operator** namespace. Create a dedicated namespace for these resources.

You can define whether the workload should be deployed using the default instance type, which you defined in the config map, by adding an annotation to the YAML file.

If you do not want to define the instance type manually, you can add an annotation to use an automatic instance type, based on the memory available.

Prerequisites

- You have created a secret object for your provider.
- You have created a config map for your provider.
- You have created a **KataConfig** custom resource (CR).

Procedure

1. Add **spec.runtimeClassName: kata-remote** to the manifest of each pod-templated workload object as in the following example:

```
apiVersion: v1
kind: <object>
# ...
spec:
  runtimeClassName: kata-remote
# ...
```

2. Add an annotation to the pod-templated object to use a manually defined instance type or an automatic instance type:

- To use a manually defined instance type, add the following annotation:

```
apiVersion: v1
kind: <object>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.machine_type: t3.medium 1
# ...
```

- 1** Specify the instance type that you defined in the config map.

- To use an automatic instance type, add the following annotations:

```
apiVersion: v1
kind: <Pod>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.default_vcpus: <vcpus>
    io.katacontainers.config.hypervisor.default_memory: <memory>
# ...
```

Define the amount of memory available for the workload to use. The workload will run on an automatic instance type based on the amount of memory available.

3. Apply the changes to the workload object by running the following command:

```
$ oc apply -f <object.yaml>
```

OpenShift Container Platform creates the workload object and begins scheduling it.

Verification

- Inspect the **spec.runtimeClassName** field of a pod-templated object. If the value is **kata-remote**, then the workload is running on OpenShift sandboxed containers, using peer pods.

3.2. DEPLOYING WORKLOADS ON AZURE

You can deploy OpenShift sandboxed containers workloads on Microsoft Azure Cloud Computing Services by using the OpenShift Container Platform web console or the command line interface (CLI).

Deployment workflow

1. Create a secret for your Azure access keys.
2. Create a config map to define Azure instance sizes and other parameters.
3. Create an SSH key secret.
4. Create a **KataConfig** custom resource.
5. Optional: Modify the peer pod VM limit per node.
6. Configure your workload objects to use the **kata-remote** runtime class.

3.2.1. Preparing your environment

Perform the following steps to prepare your environment:

1. Ensure that your cluster has sufficient resources.
2. Install the OpenShift sandboxed containers Operator.

3.2.1.1. Resource requirements

Peer pod virtual machines (VMs) require resources in two locations:

- The worker node. The worker node stores metadata, Kata shim resources (**containerd-shim-kata-v2**), remote-hypervisor resources (**cloud-api-adaptor**), and the tunnel setup between the worker nodes and the peer pod VM.
- The cloud instance. This is the actual peer pod VM running in the cloud.

The CPU and memory resources used in the Kubernetes worker node are handled by the [pod overhead](#) included in the RuntimeClass (**kata-remote**) definition used for creating peer pods.

The total number of peer pod VMs running in the cloud is defined as Kubernetes Node extended resources. This limit is per node and is set by the **limit** attribute in the **peerpodConfig** custom resource (CR).

The **peerpodConfig** CR, named **peerpodconfig-openshift**, is created when you create the **kataConfig** CR and enable peer pods, and is located in the **openshift-sandboxed-containers-operator** namespace.

The following **peerpodConfig** CR example displays the default **spec** values:

```
apiVersion: confidentialcontainers.org/v1alpha1
kind: PeerPodConfig
metadata:
  name: peerpodconfig-openshift
  namespace: openshift-sandboxed-containers-operator
spec:
  cloudSecretName: peer-pods-secret
  configMapName: peer-pods-cm
  limit: "10" 1
  nodeSelector:
    node-role.kubernetes.io/kata-oc: ""
```

1 The default limit is 10 VMs per node.

The extended resource is named **kata.peerpods.io/vm**, and enables the Kubernetes scheduler to handle capacity tracking and accounting.

You can edit the limit per node based on the requirements for your environment. See "Modifying the VM limit per node in peer pods" for more information.

A [mutating webhook](#) adds the extended resource **kata.peerpods.io/vm** to the pod specification. It also removes any resource-specific entries from the pod specification, if present. This enables the Kubernetes scheduler to account for these extended resources, ensuring the peer pod is only scheduled when resources are available.

The mutating webhook modifies a Kubernetes pod as follows:

- The mutating webhook checks the pod for the expected **RuntimeClassName** value, specified in the **TARGET_RUNTIME_CLASS** environment variable. If the value in the pod specification does not match the value in the **TARGET_RUNTIME_CLASS**, the webhook exits without modifying the pod.
- If the **RuntimeClassName** values match, the webhook makes the following changes to the pod spec:
 1. The webhook removes every resource specification from the **resources** field of all containers and init containers in the pod.
 2. The webhook adds the extended resource (**kata.peerpods.io/vm**) to the spec by modifying the resources field of the first container in the pod. The extended resource **kata.peerpods.io/vm** is used by the Kubernetes scheduler for accounting purposes.

**NOTE**

The mutating webhook excludes specific system namespaces in OpenShift Container Platform from mutation. If a peer pod is created in those system namespaces, then resource accounting using Kubernetes extended resources does not work unless the pod spec includes the extended resource.

As a best practice, define a cluster-wide policy to only allow peer pod creation in specific namespaces.

3.2.1.2. Installing the OpenShift sandboxed containers Operator

You can install the OpenShift sandboxed containers Operator by using the OpenShift Container Platform web console or command line interface (CLI).

3.2.1.2.1. Installing the Operator by using the web console

You can install the OpenShift sandboxed containers Operator by using the Red Hat OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators** → **OperatorHub**.
2. In the **Filter by keyword** field, type **OpenShift sandboxed containers**.
3. Select the **OpenShift sandboxed containers Operator** tile and click **Install**.
4. On the **Install Operator** page, select **stable** from the list of available **Update Channel** options.
5. Verify that **Operator recommended Namespace** is selected for **Installed Namespace**. This installs the Operator in the mandatory **openshift-sandboxed-containers-operator** namespace. If this namespace does not yet exist, it is automatically created.

**NOTE**

Attempting to install the OpenShift sandboxed containers Operator in a namespace other than **openshift-sandboxed-containers-operator** causes the installation to fail.

6. Verify that **Automatic** is selected for **Approval Strategy**. **Automatic** is the default value, and enables automatic updates to OpenShift sandboxed containers when a new z-stream release is available.
7. Click **Install**.

The OpenShift sandboxed containers Operator is now installed on your cluster.

Verification

1. Navigate to **Operators** → **Installed Operators**.

2. Verify that the OpenShift sandboxed containers Operator is displayed.

Additional resources

- [Using Operator Lifecycle Manager on restricted networks](#) .
- [Configuring proxy support in Operator Lifecycle Manager](#) for disconnected environments.

3.2.1.2.2. Installing the Operator by using the CLI

You can install the OpenShift sandboxed containers Operator by using the CLI.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a **Namespace.yaml** manifest file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sandboxed-containers-operator
```

2. Create the namespace by running the following command:

```
$ oc create -f Namespace.yaml
```

3. Create an **OperatorGroup.yaml** manifest file:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
  targetNamespaces:
    - openshift-sandboxed-containers-operator
```

4. Create the operator group by running the following command:

```
$ oc create -f OperatorGroup.yaml
```

5. Create a **Subscription.yaml** manifest file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
```



```
channel: stable
installPlanApproval: Automatic
name: sandboxed-containers-operator
source: redhat-operators
sourceNamespace: openshift-marketplace
startingCSV: sandboxed-containers-operator.v1.6.0
```

6. Create the subscription by running the following command:

```
$ oc create -f Subscription.yaml
```

The OpenShift sandboxed containers Operator is now installed on your cluster.

Verification

- Ensure that the Operator is correctly installed by running the following command:

```
$ oc get csv -n openshift-sandboxed-containers-operator
```

Example output

```
NAME                                DISPLAY                                VERSION  REPLACES
PHASE
openshift-sandboxed-containers  openshift-sandboxed-containers-operator  1.6.0   1.5.3
Succeeded
```

3.2.1.2.3. Additional resources

- [Using Operator Lifecycle Manager on restricted networks](#)
- [Configuring proxy support in Operator Lifecycle Manager](#) for disconnected environments

3.2.2. Deploying workloads by using the web console

You can deploy OpenShift sandboxed containers workloads by using the web console.

3.2.2.1. Creating a secret

You must create a **Secret** object on your OpenShift Container Platform cluster. The secret stores cloud provider credentials for creating the pod virtual machine (VM) image and peer pod instances. By default, the OpenShift sandboxed containers Operator creates the secret based on the credentials used to create the cluster. However, you can manually create a secret that uses different credentials.

Prerequisites

- You have installed and configured the Azure CLI tool.

Procedure

1. Retrieve the Azure subscription ID:

```
$ AZURE_SUBSCRIPTION_ID=$(az account list --query "[?isDefault].id" -o tsv) && echo
"AZURE_SUBSCRIPTION_ID: \"$AZURE_SUBSCRIPTION_ID\""
```

-
2. Generate the RBAC content. This generates the client ID, client secret, and the tenant ID:

```
$ az ad sp create-for-rbac --role Contributor --scopes  
/subscriptions/$AZURE_SUBSCRIPTION_ID --query "{ client_id: appId, client_secret:  
password, tenant_id: tenant }
```

Example output:

```
{  
  "client_id": `AZURE_CLIENT_ID`,  
  "client_secret": `AZURE_CLIENT_SECRET`,  
  "tenant_id": `AZURE_TENANT_ID`  
}
```

-
-
3. Record the RBAC output to use in the **secret** object.
4. In the OpenShift Container Platform web console, navigate to **Operators → Installed Operators**.
5. Click the OpenShift sandboxed containers Operator tile.
6. Click the Import icon (+) on the top right corner.
7. In the **Import YAML** window, paste the following YAML manifest:

```
apiVersion: v1  
kind: Secret  
metadata:  
  name: peer-pods-secret  
  namespace: openshift-sandboxed-containers-operator  
type: Opaque  
stringData:  
  AZURE_CLIENT_ID: "<azure_client_id>" 1  
  AZURE_CLIENT_SECRET: "<azure_client_secret>" 2  
  AZURE_TENANT_ID: "<azure_tenant_id>" 3  
  AZURE_SUBSCRIPTION_ID: "<azure_subscription_id>" 4
```

- 1 Specify the **AZURE_CLIENT_ID** value.
- 2 Specify the **AZURE_CLIENT_SECRET** value.
- 3 Specify the **AZURE_TENANT_ID** value.
- 4 Specify the **AZURE_SUBSCRIPTION_ID** value.

-
-
-
-
-
-
-
8. Click **Save** to apply the changes.



NOTE

If you update the peer pods secret, you must restart the **peerpodconfig-ctrl-cao-daemon** DaemonSet to apply the changes.

After you update the secret, click **Save** to apply the changes. Then restart the **cloud-api-adapter** pods by running the following command:

```
$ oc set env ds/peerpodconfig-ctrl-cao-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

Restarting a daemon set recreates peer pods. It does not update existing pods.

Verification

- Navigate to **Workloads** → **Secrets** to view the secret.

3.2.2.2. Creating a config map

You must create a config map on your OpenShift Container Platform cluster for your cloud provider.

Procedure

1. Obtain the following values from your Azure instance:

- a. Retrieve and record the Azure VNet name:

```
$ AZURE_VNET_NAME=$(az network vnet list --resource-group
${AZURE_RESOURCE_GROUP} --query "[].{Name:name}" --output tsv)
```

This value is used to retrieve the Azure subnet ID.

- b. Retrieve and record the Azure subnet ID:

```
$ AZURE_SUBNET_ID=$(az network vnet subnet list --resource-group
${AZURE_RESOURCE_GROUP} --vnet-name $AZURE_VNET_NAME --query "[].{Id:id}
| [? contains(Id, 'worker')]" --output tsv) && echo "AZURE_SUBNET_ID:
\"$AZURE_SUBNET_ID\""
```

- c. Retrieve and record the Azure network security group (NSG) ID:

```
$ AZURE_NS_G_ID=$(az network nsg list --resource-group
${AZURE_RESOURCE_GROUP} --query "[].{Id:id}" --output tsv) && echo
"AZURE_NS_G_ID: \"$AZURE_NS_G_ID\""
```

- d. Retrieve and record the Azure resource group:

```
$ AZURE_RESOURCE_GROUP=$(oc get infrastructure/cluster -o
jsonpath='{.status.platformStatus.azure.resourceGroupName}') && echo
"AZURE_RESOURCE_GROUP: \"$AZURE_RESOURCE_GROUP\""
```

- e. Retrieve and record the Azure region:

```
$ AZURE_REGION=$(az group show --resource-group
${AZURE_RESOURCE_GROUP} --query "{Location:location}" --output tsv) && echo
"AZURE_REGION: \"$AZURE_REGION\""
```

- In the OpenShift Container Platform web console, navigate to **Operators → Installed Operators**.
- Select the OpenShift sandboxed containers Operator from the list of operators.
- Click the Import icon (+) in the top right corner.
- In the **Import YAML** window, paste the following YAML manifest:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: peer-pods-cm
  namespace: openshift-sandboxed-containers-operator
data:
  CLOUD_PROVIDER: "azure"
  VXLAN_PORT: "9000"
  AZURE_INSTANCE_SIZE: "Standard_B2als_v2" 1
  AZURE_INSTANCE_SIZES:
"Standard_B2als_v2,Standard_D2as_v5,Standard_D4as_v5,Standard_D2ads_v5" 2
  AZURE_SUBNET_ID: "<azure_subnet_id>" 3
  AZURE_NS_G_ID: "<azure_nsg_id>" 4
  PROXY_TIMEOUT: "5m"
  DISABLECVM: "true"
  AZURE_IMAGE_ID: "<azure_image_id>" 5
  AZURE_REGION: "<azure_region>" 6
  AZURE_RESOURCE_GROUP: "<azure_resource_group>" 7
```

- 1 Defines the default instance size that is used when a type is not defined in the workload.
- 2 Lists all of the instance sizes you can specify when creating the pod. This allows you to define smaller instance sizes for workloads that need less memory and fewer CPUs or larger instance sizes for larger workloads.
- 3 Specify the **AZURE_SUBNET_ID** value that you retrieved.
- 4 Specify the **AZURE_NS_G_ID** value that you retrieved.
- 5 Optional: By default, this value is populated when you run the **KataConfig** CR, using an Azure image ID based on your cluster credentials. If you create your own Azure image, specify the correct image ID.
- 6 Specify the **AZURE_REGION** value you retrieved.
- 7 Specify the **AZURE_RESOURCE_GROUP** value you retrieved.

- Click **Save** to apply the changes.
A config map is created for your cloud provider.



NOTE

If you update the peer pods config map, you must restart the **peerpodconfig-ctrl-caa-daemon** daemonset to apply the changes.

After you update the config map, click **Save** to apply the changes. Then restart the **cloud-api-adaptor** pods by running the following command:

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

Restarting the daemonset recreates the peer pods. It does not update the existing pods.

Verification

- Navigate to **Workloads** → **ConfigMaps** to view the new config map.

3.2.2.3. Creating an SSH key secret

You must create an SSH key **secret** object for Azure.

Procedure

1. Log in to your OpenShift Container Platform cluster.
2. Generate an SSH key pair by running the following command:


```
$ ssh-keygen -f ./id_rsa -N ""
```
3. In the OpenShift Container Platform web console, navigate to **Workloads** → **Secrets**.
4. On the **Secrets** page, verify that you are in the **openshift-sandboxed-containers-operator** project.
5. Click **Create** and select **Key/value secret**
6. In the **Secret name** field, enter **ssh-key-secret**.
7. In the **Key** field, enter **id_rsa.pub**.
8. In the **Value** field, paste your public SSH key.
9. Click **Create**.
The SSH key secret is created.
10. Delete the SSH keys you created:

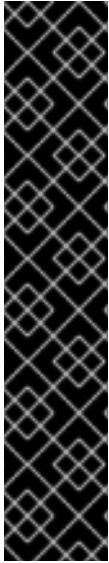
```
$ shred -remove id_rsa.pub id_rsa
```

3.2.2.4. Creating a KataConfig custom resource

You must create a **KataConfig** custom resource (CR) to install **kata-remote** as a **RuntimeClass** on your worker nodes.

The **kata-remote** runtime class is installed on all worker nodes by default. If you want to install **kata-remote** only on specific nodes, you can add labels to those nodes and then define the label in the **KataConfig** CR.

OpenShift sandboxed containers installs **kata-remote** as a *secondary, optional* runtime on the cluster and not as the primary runtime.



IMPORTANT

Creating the **KataConfig** CR automatically reboots the worker nodes. The reboot can take from 10 to more than 60 minutes. The following factors might increase the reboot time:

- A larger OpenShift Container Platform deployment with a greater number of worker nodes.
- Activation of the BIOS and Diagnostics utility.
- Deployment on a hard disk drive rather than an SSD.
- Deployment on physical nodes such as bare metal, rather than on virtual nodes.
- A slow CPU and network.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators → Installed Operators**.
2. Select the OpenShift sandboxed containers Operator.
3. On the **KataConfig** tab, click **Create KataConfig**.
4. Enter the following details:
 - **Name:** Optional: The default name is **example-kataconfig**.
 - **Labels:** Optional: Enter any relevant, identifying attributes to the **KataConfig** resource. Each label represents a key-value pair.
 - **enablePeerPods:** Select for public cloud, IBM Z®, and IBM® LinuxONE deployments.
 - **kataConfigPoolSelector.** Optional: To install **kata-remote** on selected nodes, add a match expression for the labels on the selected nodes:
 - a. Expand the **kataConfigPoolSelector** area.
 - b. In the **kataConfigPoolSelector** area, expand **matchExpressions**. This is a list of label selector requirements.
 - c. Click **Add matchExpressions**.
 - d. In the **Key** field, enter the label key the selector applies to.

- e. In the **Operator** field, enter the key's relationship to the label values. Valid operators are **In**, **NotIn**, **Exists**, and **DoesNotExist**.
 - f. Expand the **Values** area and then click **Add value**.
 - g. In the **Value** field, enter **true** or **false** for **key** label value.
- **logLevel**: Define the level of log data retrieved for nodes with the **kata-remote** runtime class.
5. Click **Create**. The **KataConfig** CR is created and installs the **kata-remote** runtime class on the worker nodes.
Wait for the **kata-remote** installation to complete and the worker nodes to reboot before verifying the installation.

Verification

1. On the **KataConfig** tab, click the **KataConfig** CR to view its details.
2. Click the **YAML** tab to view the **status** stanza.
The **status** stanza contains the **conditions** and **kataNodes** keys. The value of **status.kataNodes** is an array of nodes, each of which lists nodes in a particular state of **kata-remote** installation. A message appears each time there is an update.
3. Click **Reload** to refresh the YAML.
When all workers in the **status.kataNodes** array display the values **installed** and **conditions.InProgress: False** with no specified reason, the **kata-remote** is installed on the cluster.

See [KataConfig status messages](#) for details.

3.2.2.4.1. Optional: Verifying the pod VM image

After **kata-remote** is installed on your cluster, the OpenShift sandboxed containers Operator creates a pod VM image, which is used to create peer pods. This process can take a long time because the image is created on the cloud instance. You can verify that the pod VM image was created successfully by checking the config map that you created for the cloud provider.

Procedure

1. Navigate to **Workloads** → **ConfigMaps**.
2. Click the provider config map to view its details.
3. Click the **YAML** tab.
4. Check the **status** stanza of the YAML file.
If the **AZURE_IMAGE_ID** parameter is populated, the pod VM image was created successfully.

Troubleshooting

1. Retrieve the events log by running the following command:

```
$ oc get events -n openshift-sandboxed-containers-operator --field-selector
involvedObject.name=osc-podvm-image-creation
```

- Retrieve the job log by running the following command:

```
$ oc logs -n openshift-sandboxed-containers-operator jobs/osc-podvm-image-creation
```

If you cannot resolve the issue, submit a Red Hat Support case and attach the output of both logs.

3.2.2.5. Optional: Modifying the number of peer pod VMs per node

You can change the limit of peer pod virtual machines (VMs) per node by editing the **peerpodConfig** custom resource (CR).

Procedure

- Check the current limit by running the following command:

```
$ oc get peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-operator \
-o jsonpath='{.spec.limit}'
```

- Modify the **limit** attribute of the **peerpodConfig** CR by running the following command:

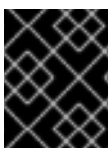
```
$ oc patch peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-operator \
--type merge --patch '{"spec":{"limit":"<value>"}}' 1
```

- Replace <value> with the limit you want to define.

3.2.2.6. Configuring workload objects

You deploy an OpenShift sandboxed containers workload by configuring **kata-remote** as the runtime class for the following pod-templated objects:

- **Pod** objects
- **ReplicaSet** objects
- **ReplicationController** objects
- **StatefulSet** objects
- **Deployment** objects
- **DeploymentConfig** objects



IMPORTANT

Do not deploy workloads in the **openshift-sandboxed-containers-operator** namespace. Create a dedicated namespace for these resources.

You can define whether the workload should be deployed using the default instance size, which you defined in the config map, by adding an annotation to the YAML file.

If you do not want to define the instance size manually, you can add an annotation to use an automatic instance size, based on the memory available.

Prerequisites

- You have created a secret object for your provider.
- You have created a config map for your provider.
- You have created a **KataConfig** custom resource (CR).

Procedure

1. In the OpenShift Container Platform web console, navigate to **Workloads** → workload type, for example, **Pods**.
2. On the workload type page, click an object to view its details.
3. Click the **YAML** tab.
4. Add **spec.runtimeClassName: kata-remote** to the manifest of each pod-templated workload object as in the following example:

```
apiVersion: v1
kind: <object>
# ...
spec:
  runtimeClassName: kata-remote
# ...
```

5. Add an annotation to the pod-templated object to use a manually defined instance size or an automatic instance size:

- To use a manually defined instance size, add the following annotation:

```
apiVersion: v1
kind: <object>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.machine_type: Standard_B2als_v2 1
# ...
```

- 1** Specify the instance size that you defined in the config map.

- To use an automatic instance size, add the following annotations:

```
apiVersion: v1
kind: <Pod>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.default_vcpus: <vcpus>
    io.katacontainers.config.hypervisor.default_memory: <memory>
# ...
```

Define the amount of memory available for the workload to use. The workload will run on an automatic instance size based on the amount of memory available.

- Click **Save** to apply the changes.
OpenShift Container Platform creates the workload object and begins scheduling it.

Verification

- Inspect the **spec.runtimeClassName** field of a pod-templated object. If the value is **kata-remote**, then the workload is running on OpenShift sandboxed containers, using peer pods.

3.2.3. Deploying workloads by using the command line

You can deploy OpenShift sandboxed containers workloads by using the command line.

3.2.3.1. Creating a secret

You must create a **Secret** object on your OpenShift Container Platform cluster. The secret stores cloud provider credentials for creating the pod virtual machine (VM) image and peer pod instances. By default, the OpenShift sandboxed containers Operator creates the secret based on the credentials used to create the cluster. However, you can manually create a secret that uses different credentials.

Prerequisites

- You have installed and configured the Azure CLI tool.

Procedure

- Retrieve the Azure subscription ID:

```
$ AZURE_SUBSCRIPTION_ID=$(az account list --query "[?isDefault].id" -o tsv) && echo "AZURE_SUBSCRIPTION_ID: \"$AZURE_SUBSCRIPTION_ID\""
```

- Generate the RBAC content. This generates the client ID, client secret, and the tenant ID:

```
$ az ad sp create-for-rbac --role Contributor --scopes /subscriptions/$AZURE_SUBSCRIPTION_ID --query "{ client_id: appId, client_secret: password, tenant_id: tenant }
```

Example output:

```
{
  "client_id": `AZURE_CLIENT_ID`,
  "client_secret": `AZURE_CLIENT_SECRET`,
  "tenant_id": `AZURE_TENANT_ID`
}
```

- Record the RBAC output to use in the **secret** object.
- Create a **peer-pods-secret.yaml** manifest file according to the following example:

```
apiVersion: v1
kind: Secret
metadata:
```

```

name: peer-pods-secret
namespace: openshift-sandboxed-containers-operator
type: Opaque
stringData:
  AZURE_CLIENT_ID: "<azure_client_id>" 1
  AZURE_CLIENT_SECRET: "<azure_client_secret>" 2
  AZURE_TENANT_ID: "<azure_tenant_id>" 3
  AZURE_SUBSCRIPTION_ID: "<azure_subscription_id>" 4

```

- 1 Specify the **AZURE_CLIENT_ID** value.
- 2 Specify the **AZURE_CLIENT_SECRET** value.
- 3 Specify the **AZURE_TENANT_ID** value.
- 4 Specify the **AZURE_SUBSCRIPTION_ID** value.

5. Create the **secret** object by applying the manifest:

```
$ oc apply -f peer-pods-secret.yaml
```

NOTE

If you update the peer pods secret, you must restart the **peerpodconfig-ctrl-cao-daemon** DaemonSet to apply the changes.

After you update the secret, apply the manifest. Then restart the **cloud-api-adaptor** pods by running the following command:

```
$ oc set env ds/peerpodconfig-ctrl-cao-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

Restarting a daemon set recreates peer pods. It does not update existing pods.

3.2.3.2. Creating a config map

You must create a config map on your OpenShift Container Platform cluster for your cloud provider.

Procedure

1. Obtain the following values from your Azure instance:
 - a. Retrieve and record the Azure VNet name:

```
$ AZURE_VNET_NAME=$(az network vnet list --resource-group
${AZURE_RESOURCE_GROUP} --query "[].{Name:name}" --output tsv)
```

This value is used to retrieve the Azure subnet ID.

- b. Retrieve and record the Azure subnet ID:

```
$ AZURE_SUBNET_ID=$(az network vnet subnet list --resource-group
${AZURE_RESOURCE_GROUP} --vnet-name $AZURE_VNET_NAME --query "[].{Id:id}
```

```
| [? contains(Id, 'worker')]" --output tsv) && echo "AZURE_SUBNET_ID:
\"$AZURE_SUBNET_ID\""
```

- c. Retrieve and record the Azure network security group (NSG) ID:

```
$ AZURE_NS_G_ID=$(az network nsg list --resource-group
${AZURE_RESOURCE_GROUP} --query "[].{Id:id}" --output tsv) && echo
"AZURE_NS_G_ID: \"$AZURE_NS_G_ID\""
```

- d. Retrieve and record the Azure resource group:

```
$ AZURE_RESOURCE_GROUP=$(oc get infrastructure/cluster -o
jsonpath='{.status.platformStatus.azure.resourceGroupName}') && echo
"AZURE_RESOURCE_GROUP: \"$AZURE_RESOURCE_GROUP\""
```

- e. Retrieve and record the Azure region:

```
$ AZURE_REGION=$(az group show --resource-group
${AZURE_RESOURCE_GROUP} --query "{Location:location}" --output tsv) && echo
"AZURE_REGION: \"$AZURE_REGION\""
```

2. Create a **peer-pods-cm.yaml** manifest according to the following example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: peer-pods-cm
  namespace: openshift-sandboxed-containers-operator
data:
  CLOUD_PROVIDER: "azure"
  VXLAN_PORT: "9000"
  AZURE_INSTANCE_SIZE: "Standard_B2als_v2" ❶
  AZURE_INSTANCE_SIZES:
"Standard_B2als_v2,Standard_D2as_v5,Standard_D4as_v5,Standard_D2ads_v5" ❷
  AZURE_SUBNET_ID: "<azure_subnet_id>" ❸
  AZURE_NS_G_ID: "<azure_nsg_id>" ❹
  PROXY_TIMEOUT: "5m"
  DISABLECVM: "true"
  AZURE_IMAGE_ID: "<azure_image_id>" ❺
  AZURE_REGION: "<azure_region>" ❻
  AZURE_RESOURCE_GROUP: "<azure_resource_group>" ❼
```

- ❶ Defines the default instance size that is used when a type is not defined in the workload.
- ❷ Lists all of the instance sizes you can specify when creating the pod. This allows you to define smaller instance sizes for workloads that need less memory and fewer CPUs or larger instance sizes for larger workloads.
- ❸ Specify the **AZURE_SUBNET_ID** value that you retrieved.
- ❹ Specify the **AZURE_NS_G_ID** value that you retrieved.
- ❺ Optional: By default, this value is populated when you run the **KataConfig** CR, using an

- 6 Specify the **AZURE_REGION** value you retrieved.
- 7 Specify the **AZURE_RESOURCE_GROUP** value you retrieved.

3. Apply the manifest to create a config map:

```
$ oc apply -f peer-pods-cm.yaml
```

A config map is created for your cloud provider.



NOTE

If you update the peer pods config map, you must restart the **peerpodconfig-ctrl-caa-daemon** daemonset to apply the changes.

After you update the config map, apply the manifest. Then restart the **cloud-api-adaptor** pods by running the following command:

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

Restarting the daemonset recreates the peer pods. It does not update the existing pods.

3.2.3.3. Creating an SSH key secret

You must create an SSH key **secret** object for Azure.

Procedure

1. Log in to your OpenShift Container Platform cluster.
2. Generate an SSH key pair by running the following command:

```
$ ssh-keygen -f ./id_rsa -N ""
```

3. Create the **Secret** object by running the following command:

```
$ oc create secret generic ssh-key-secret \
  -n openshift-sandboxed-containers-operator \
  --from-file=id_rsa.pub=./id_rsa.pub \
  --from-file=id_rsa=./id_rsa
```

The SSH key secret is created.

4. Delete the SSH keys you created:

```
$ shred -remove id_rsa.pub id_rsa
```

3.2.3.4. Creating a KataConfig custom resource

You must create a **KataConfig** custom resource (CR) to install **kata-remote** as a runtime class on your worker nodes.

Creating the **KataConfig** CR triggers the OpenShift sandboxed containers Operator to do the following:

- Create a **RuntimeClass** CR named **kata-remote** with a default configuration. This enables users to configure workloads to use **kata-remote** as the runtime by referencing the CR in the **RuntimeClassName** field. This CR also specifies the resource overhead for the runtime.

OpenShift sandboxed containers installs **kata-remote** as a *secondary, optional* runtime on the cluster and not as the primary runtime.



IMPORTANT

Creating the **KataConfig** CR automatically reboots the worker nodes. The reboot can take from 10 to more than 60 minutes. Factors that impede reboot time are as follows:

- A larger OpenShift Container Platform deployment with a greater number of worker nodes.
- Activation of the BIOS and Diagnostics utility.
- Deployment on a hard disk drive rather than an SSD.
- Deployment on physical nodes such as bare metal, rather than on virtual nodes.
- A slow CPU and network.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a **cluster-kataconfig.yaml** manifest file according to the following example:

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  enablePeerPods: true
  logLevel: info
```

2. Optional: To install **kata-remote** on selected nodes, specify the node labels according to the following example:

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  kataConfigPoolSelector:
    matchLabels:
      <label_key>: '<label_value>' 1
# ...
```

- 1 Specify the labels of the selected nodes.

3. Create the **KataConfig** CR:

```
$ oc create -f cluster-kataconfig.yaml
```

The new **KataConfig** CR is created and installs **kata-remote** as a runtime class on the worker nodes.

Wait for the **kata-remote** installation to complete and the worker nodes to reboot before verifying the installation.

Verification

- Monitor the installation progress by running the following command:

```
$ watch "oc describe kataconfig | sed -n /^Status:/,/^Events/p"
```

When the status of all workers under **kataNodes** is **installed** and the condition **InProgress** is **False** without specifying a reason, the **kata-remote** is installed on the cluster.

See [KataConfig status messages](#) for details.

3.2.3.4.1. Optional: Verifying the pod VM image

After **kata-remote** is installed on your cluster, the OpenShift sandboxed containers Operator creates a pod VM image, which is used to create peer pods. This process can take a long time because the image is created on the cloud instance. You can verify that the pod VM image was created successfully by checking the config map that you created for the cloud provider.

Procedure

1. Obtain the config map you created for the peer pods:

```
$ oc get configmap peer-pods-cm -n openshift-sandboxed-containers-operator -o yaml
```

2. Check the **status** stanza of the YAML file.
If the **AZURE_IMAGE_ID** parameter is populated, the pod VM image was created successfully.

Troubleshooting

1. Retrieve the events log by running the following command:

```
$ oc get events -n openshift-sandboxed-containers-operator --field-selector involvedObject.name=osc-podvm-image-creation
```

2. Retrieve the job log by running the following command:

```
$ oc logs -n openshift-sandboxed-containers-operator jobs/osc-podvm-image-creation
```

If you cannot resolve the issue, submit a Red Hat Support case and attach the output of both logs.

3.2.3.5. Optional: Modifying the number of peer pod VMs per node

You can change the limit of peer pod virtual machines (VMs) per node by editing the **peerpodConfig** custom resource (CR).

Procedure

1. Check the current limit by running the following command:

```
$ oc get peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-operator \
-o jsonpath='{.spec.limit}'
```

2. Modify the **limit** attribute of the **peerpodConfig** CR by running the following command:

```
$ oc patch peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-operator \
--type merge --patch '{"spec":{"limit":<value>}}' 1
```

- 1 Replace <value> with the limit you want to define.

3.2.3.6. Configuring workload objects

You deploy an OpenShift sandboxed containers workload by configuring **kata-remote** as the runtime class for the following pod-templated objects:

- **Pod** objects
- **ReplicaSet** objects
- **ReplicationController** objects
- **StatefulSet** objects
- **Deployment** objects
- **DeploymentConfig** objects



IMPORTANT

Do not deploy workloads in the **openshift-sandboxed-containers-operator** namespace. Create a dedicated namespace for these resources.

You can define whether the workload should be deployed using the default instance size, which you defined in the config map, by adding an annotation to the YAML file.

If you do not want to define the instance size manually, you can add an annotation to use an automatic instance size, based on the memory available.

Prerequisites

- You have created a secret object for your provider.
- You have created a config map for your provider.

- You have created a **KataConfig** custom resource (CR).

Procedure

- Add **spec.runtimeClassName: kata-remote** to the manifest of each pod-templated workload object as in the following example:

```
apiVersion: v1
kind: <object>
# ...
spec:
  runtimeClassName: kata-remote
# ...
```

- Add an annotation to the pod-templated object to use a manually defined instance size or an automatic instance size:

- To use a manually defined instance size, add the following annotation:

```
apiVersion: v1
kind: <object>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.machine_type: Standard_B2als_v2 1
# ...
```

- Specify the instance size that you defined in the config map.

- To use an automatic instance size, add the following annotations:

```
apiVersion: v1
kind: <Pod>
metadata:
  annotations:
    io.katacontainers.config.hypervisor.default_vcpus: <vcpus>
    io.katacontainers.config.hypervisor.default_memory: <memory>
# ...
```

Define the amount of memory available for the workload to use. The workload will run on an automatic instance size based on the amount of memory available.

- Apply the changes to the workload object by running the following command:

```
$ oc apply -f <object.yaml>
```

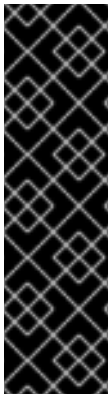
OpenShift Container Platform creates the workload object and begins scheduling it.

Verification

- Inspect the **spec.runtimeClassName** field of a pod-templated object. If the value is **kata-remote**, then the workload is running on OpenShift sandboxed containers, using peer pods.

CHAPTER 4. DEPLOYING WORKLOADS ON IBM

You can deploy OpenShift sandboxed containers workloads on IBM Z® and IBM® LinuxONE.



IMPORTANT

Deploying OpenShift sandboxed containers workloads on IBM Z® and IBM® LinuxONE is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

For more information about the support scope of Red Hat Technology Preview features, see [Technology Preview Features Support Scope](#).

Cluster prerequisites

- You have installed Red Hat OpenShift Container Platform 4.14 or later.
- Your cluster has three control nodes and two worker nodes.

Deployment flow

While this document refers only to IBM Z®, all procedures also apply to IBM® LinuxONE.

You deploy OpenShift sandboxed containers workloads by performing the following steps:

1. Configure a libvirt volume on your KVM host.
2. Create a KVM guest image and upload it to the libvirt volume.
3. Create a peer pod VM image and upload it to the libvirt volume.
4. Create a secret for the libvirt provider.
5. Create a config map for the libvirt provider.
6. Create an SSH key secret for your KVM host.
7. Create a **KataConfig** CR.
8. Optional: Modify the peer pod VM limit per node.
9. Configure your workload objects to use the **kata-remote** runtime class.



NOTE

- Cluster nodes and peer pods must be in the same IBM Z® KVM host logical partition (LPAR).
- Cluster nodes and peer pods must be connected to the same subnet.

4.1. PREPARING YOUR ENVIRONMENT

Perform the following steps to prepare your environment:

1. Ensure that your cluster has sufficient resources.
2. Install the OpenShift sandboxed containers Operator.

4.1.1. Resource requirements

Peer pod virtual machines (VMs) require resources in two locations:

- The worker node. The worker node stores metadata, Kata shim resources (**containerd-shim-kata-v2**), remote-hypervisor resources (**cloud-api-adaptor**), and the tunnel setup between the worker nodes and the peer pod VM.
- The cloud instance. This is the actual peer pod VM running in the cloud.

The CPU and memory resources used in the Kubernetes worker node are handled by the [pod overhead](#) included in the RuntimeClass (**kata-remote**) definition used for creating peer pods.

The total number of peer pod VMs running in the cloud is defined as Kubernetes Node extended resources. This limit is per node and is set by the **limit** attribute in the **peerpodConfig** custom resource (CR).

The **peerpodConfig** CR, named **peerpodconfig-openshift**, is created when you create the **kataConfig** CR and enable peer pods, and is located in the **openshift-sandboxed-containers-operator** namespace.

The following **peerpodConfig** CR example displays the default **spec** values:

```
apiVersion: confidentialcontainers.org/v1alpha1
kind: PeerPodConfig
metadata:
  name: peerpodconfig-openshift
  namespace: openshift-sandboxed-containers-operator
spec:
  cloudSecretName: peer-pods-secret
  configMapName: peer-pods-cm
  limit: "10" 1
  nodeSelector:
    node-role.kubernetes.io/kata-oc: ""
```

- 1** The default limit is 10 VMs per node.

The extended resource is named **kata.peerpods.io/vm**, and enables the Kubernetes scheduler to handle capacity tracking and accounting.

You can edit the limit per node based on the requirements for your environment. See "Modifying the VM limit per node in peer pods" for more information.

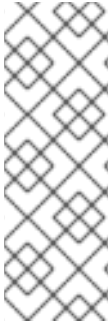
A [mutating webhook](#) adds the extended resource **kata.peerpods.io/vm** to the pod specification. It also removes any resource-specific entries from the pod specification, if present. This enables the Kubernetes scheduler to account for these extended resources, ensuring the peer pod is only scheduled when resources are available.

The mutating webhook modifies a Kubernetes pod as follows:

- The mutating webhook checks the pod for the expected **RuntimeClassName** value, specified in the **TARGET_RUNTIME_CLASS** environment variable. If the value in the pod specification

does not match the value in the **TARGET_RUNTIME_CLASS**, the webhook exits without modifying the pod.

- If the **RuntimeClassName** values match, the webhook makes the following changes to the pod spec:
 1. The webhook removes every resource specification from the **resources** field of all containers and init containers in the pod.
 2. The webhook adds the extended resource (**kata.peerpods.io/vm**) to the spec by modifying the resources field of the first container in the pod. The extended resource **kata.peerpods.io/vm** is used by the Kubernetes scheduler for accounting purposes.



NOTE

The mutating webhook excludes specific system namespaces in OpenShift Container Platform from mutation. If a peer pod is created in those system namespaces, then resource accounting using Kubernetes extended resources does not work unless the pod spec includes the extended resource.

As a best practice, define a cluster-wide policy to only allow peer pod creation in specific namespaces.

4.1.2. Installing the OpenShift sandboxed containers Operator

You can install the OpenShift sandboxed containers Operator by using the OpenShift Container Platform web console or command line interface (CLI).

4.1.2.1. Installing the Operator by using the web console

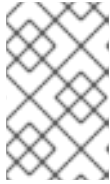
You can install the OpenShift sandboxed containers Operator by using the Red Hat OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators** → **OperatorHub**.
2. In the **Filter by keyword** field, type **OpenShift sandboxed containers**.
3. Select the **OpenShift sandboxed containers Operator** tile and click **Install**.
4. On the **Install Operator** page, select **stable** from the list of available **Update Channel** options.
5. Verify that **Operator recommended Namespace** is selected for **Installed Namespace**. This installs the Operator in the mandatory **openshift-sandboxed-containers-operator** namespace. If this namespace does not yet exist, it is automatically created.

**NOTE**

Attempting to install the OpenShift sandboxed containers Operator in a namespace other than **openshift-sandboxed-containers-operator** causes the installation to fail.

6. Verify that **Automatic** is selected for **Approval Strategy**. **Automatic** is the default value, and enables automatic updates to OpenShift sandboxed containers when a new z-stream release is available.
7. Click **Install**.

The OpenShift sandboxed containers Operator is now installed on your cluster.

Verification

1. Navigate to **Operators → Installed Operators**.
2. Verify that the OpenShift sandboxed containers Operator is displayed.

Additional resources

- [Using Operator Lifecycle Manager on restricted networks](#) .
- [Configuring proxy support in Operator Lifecycle Manager](#) for disconnected environments.

4.1.2.2. Installing the Operator by using the CLI

You can install the OpenShift sandboxed containers Operator by using the CLI.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a **Namespace.yaml** manifest file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-sandboxed-containers-operator
```

2. Create the namespace by running the following command:

```
$ oc create -f Namespace.yaml
```

3. Create an **OperatorGroup.yaml** manifest file:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
```

```

name: openshift-sandboxed-containers-operator
namespace: openshift-sandboxed-containers-operator
spec:
  targetNamespaces:
    - openshift-sandboxed-containers-operator

```

4. Create the operator group by running the following command:

```
$ oc create -f OperatorGroup.yaml
```

5. Create a **Subscription.yaml** manifest file:

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-sandboxed-containers-operator
  namespace: openshift-sandboxed-containers-operator
spec:
  channel: stable
  installPlanApproval: Automatic
  name: sandboxed-containers-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: sandboxed-containers-operator.v1.6.0

```

6. Create the subscription by running the following command:

```
$ oc create -f Subscription.yaml
```

The OpenShift sandboxed containers Operator is now installed on your cluster.

Verification

- Ensure that the Operator is correctly installed by running the following command:

```
$ oc get csv -n openshift-sandboxed-containers-operator
```

Example output

```

NAME                                DISPLAY                                VERSION  REPLACES
PHASE
openshift-sandboxed-containers  openshift-sandboxed-containers-operator  1.6.0    1.5.3
Succeeded

```

4.1.2.3. Additional resources

- [Using Operator Lifecycle Manager on restricted networks](#)
- [Configuring proxy support in Operator Lifecycle Manager](#) for disconnected environments

4.2. DEPLOYING WORKLOADS BY USING THE COMMAND LINE


```

--name $LIBVIRT_VOL_NAME \
--capacity 20G \
--allocation 2G \
--prealloc-metadata \
--format qcow2

```

4.2.2. Creating a KVM guest image

You must create a KVM guest image and upload it to the libvirt volume.

Prerequisites

- IBM z15 or later, or IBM® LinuxONE III or later.
- At least one LPAR running on RHEL 9 or later with KVM.

Procedure

1. Log in to your OpenShift Container Platform cluster.
2. If you have a RHEL subscription, set the subscription environment variables for Red Hat Subscription Management:

- Set the organization ID by running the following command:

```
$ export ORG_ID=$(cat ~/.rh_subscription/orgid)
```

- Set the activation key by running the following command:

```
$ export ACTIVATION_KEY=$(cat ~/.rh_subscription/activation_key)
```

3. If you do not have a RHEL subscription, set the subscription values for RHEL:

- Set the organization ID by running the following command:

```
$ export ORG_ID=<RHEL_ORGID_VALUE> 1
```

- 1 Specify your RHEL organization ID.

- Set the activation key by running the following command:

```
$ export ACTIVATION_KEY=<RHEL_ACTIVATION_KEY> 1
```

- 1 Specify your RHEL activation key.

4. Log in to your IBM Z® system.
5. Download the **s390x** RHEL KVM guest image from the [Red Hat Customer Portal](#) to your libvirt storage directory to grant libvirt correct access.
The default directory is **/var/lib/libvirt/images**. This image is used to generate the peer pod VM image, which includes the relevant binaries.

- Set the **IMAGE_URL** for the downloaded image by running the following command:

```
$ export IMAGE_URL=<path/to/image> 1
```

- Specify the path of the KVM guest image.

- Register the guest KVM image by running the following command:

```
$ export REGISTER_CMD="subscription-manager register --org=${ORG_ID} \
--activationkey=${ACTIVATION_KEY}"
```

- Customize the guest KVM image by running the following command:

```
$ virt-customize -v -x -a ${IMAGE_URL} --run-command "${REGISTER_CMD}"
```

- Set the checksum of the image by running the following command:

```
$ export IMAGE_CHECKSUM=$(sha256sum ${IMAGE_URL} | awk '{ print $1 }')
```

4.2.3. Building a peer pod VM image

You must build a peer pod virtual machine (VM) image and upload it to your libvirt volume.

Procedure

- Log in to your OpenShift Container Platform cluster.
- Clone the [cloud-api-adaptor](#) repository by running the following command:

```
$ git clone --single-branch https://github.com/confidential-containers/cloud-api-adaptor.git
```

- Change into the **podvm** directory by running the following command:

```
$ cd cloud-api-adaptor && git checkout 8577093
```

- Create a builder image from which the final QCOW2 image is generated.

- If you have a subscribed RHEL system, run the following command:

```
$ podman build -t podvm_builder_rhel_s390x \
--build-arg ARCH="s390x" \
--build-arg GO_VERSION="1.21.3" \
--build-arg PROTOC_VERSION="25.1" \
--build-arg PACKER_VERSION="v1.9.4" \
--build-arg RUST_VERSION="1.72.0" \
--build-arg YQ_VERSION="v4.35.1" \
--build-arg
YQ_CHECKSUM="sha256:4e6324d08630e7df733894a11830412a43703682d65a76f1fc9
25aac08268a45" \
-f podvm/Dockerfile.podvm_builder.rhel .
```

- If you have an unsubscribed RHEL system, run the following command:

```
$ podman build -t podvm_builder_rhel_s390x \
--build-arg ORG_ID=$ORG_ID \
--build-arg ACTIVATION_KEY=$ACTIVATION_KEY \
--build-arg ARCH="s390x" \
--build-arg GO_VERSION="1.21.3" \
--build-arg PROTOC_VERSION="25.1" \
--build-arg PACKER_VERSION="v1.9.4" \
--build-arg RUST_VERSION="1.72.0" \
--build-arg YQ_VERSION="v4.35.1" \
--build-arg
YQ_CHECKSUM="sha256:4e6324d08630e7df733894a11830412a43703682d65a76f1fc9
25aac08268a45" \
-f podvm/Dockerfile.podvm_builder.rhel .
```

5. Generate an intermediate image package with the required binaries for running peer pods by running the following command:

```
$ podman build -t podvm_binaries_rhel_s390x \
--build-arg BUILDER_IMG="podvm_builder_rhel_s390x:latest" \
--build-arg ARCH=s390x \
-f podvm/Dockerfile.podvm_binaries.rhel .
```

This process takes a significant length of time.

6. Extract the binaries and build the peer pod QCOW2 image by running the following command:

```
$ podman build -t podvm_rhel_s390x \
--build-arg ARCH=s390x \
--build-arg CLOUD_PROVIDER=libvirt \
--build-arg BUILDER_IMG="localhost/podvm_builder_rhel_s390x:latest" \
--build-arg BINARIES_IMG="localhost/podvm_binaries_rhel_s390x:latest" \
-v ${IMAGE_URL}:/tmp/rhel.qcow2:Z \
--build-arg IMAGE_URL="/tmp/rhel.qcow2" \
--build-arg IMAGE_CHECKSUM=${IMAGE_CHECKSUM} \
-f podvm/Dockerfile.podvm.rhel .
```

7. Create an image directory environment variable by running the following command:

```
$ export IMAGE_OUTPUT_DIR=<image_output_directory> 1
```

1 Specify a directory for the image.

8. Create the image directory by running the following command:

```
$ mkdir -p $IMAGE_OUTPUT_DIR
```

9. Save the extracted peer pod QCOW2 image by running the following command:

```
$ podman save podvm_rhel_s390x | tar -xO --no-wildcards-match-slash '*.tar' | tar -x -C
${IMAGE_OUTPUT_DIR}
```

10. Upload the peer pod QCOW2 image to your libvirt volume:

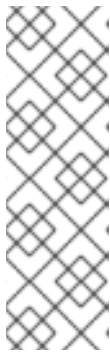
```
$ virsh -c qemu:///system vol-upload \
  --vol $LIBVIRT_VOL_NAME \
  $IMAGE_OUTPUT_DIR/podvm-*.qcow2 \
  --pool $LIBVIRT_POOL --sparse
```

4.2.4. Creating a secret

You must create a **Secret** object on your OpenShift Container Platform cluster.

Prerequisites

- **LIBVIRT_POOL**. Use the value you set when you configured libvirt on the KVM host.
- **LIBVIRT_VOL_NAME**. Use the value you set when you configured libvirt on the KVM host.
- **LIBVIRT_URI**. This value is the default gateway IP address of the libvirt network. Check your libvirt network setup to obtain this value.



NOTE

If libvirt uses the default bridge virtual network, you can obtain the **LIBVIRT_URI** by running the following commands:

```
$ virtint=$(bridge_line=$(virsh net-info default | grep Bridge); echo
"${bridge_line//Bridge:}" | tr -d [:blank:])
$ LIBVIRT_URI=$( ip -4 addr show $virtint | grep -oP '(?<=inet\s)\d+(\.\d+){3}')
```

Procedure

1. Create a **peer-pods-secret.yaml** manifest file according to the following example:

```
apiVersion: v1
kind: Secret
metadata:
  name: peer-pods-secret
  namespace: openshift-sandboxed-containers-operator
type: Opaque
stringData:
  CLOUD_PROVIDER: "libvirt"
  LIBVIRT_URI: "<libvirt_gateway_uri>" 1
  LIBVIRT_POOL: "<libvirt_pool>" 2
  LIBVIRT_VOL_NAME: "<libvirt_volume>" 3
```

- 1 Specify the libvirt URI.
- 2 Specify the libvirt pool.
- 3 Specify the libvirt volume name.

2. Create the **secret** object by applying the manifest:

```
$ oc apply -f peer-pods-secret.yaml
```

NOTE

If you update the peer pods secret, you must restart the **peerpodconfig-ctrl-caa-daemon** DaemonSet to apply the changes.

After you update the secret, apply the manifest. Then restart the **cloud-api-adaptor** pods by running the following command:

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

Restarting a daemon set recreates peer pods. It does not update existing pods.

4.2.5. Creating a config map

You must create a config map on your OpenShift Container Platform cluster for your libvirt provider.

Procedure

1. Create a **peer-pods-cm.yaml** manifest according to the following example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: peer-pods-cm
  namespace: openshift-sandboxed-containers-operator
data:
  CLOUD_PROVIDER: "libvirt"
  PROXY_TIMEOUT: "15m"
```

2. Apply the manifest to create a config map:

```
$ oc apply -f peer-pods-cm.yaml
```

A config map is created for your libvirt provider.

NOTE

If you update the peer pods config map, you must restart the **peerpodconfig-ctrl-caa-daemon** daemonset to apply the changes.

After you update the config map, apply the manifest. Then restart the **cloud-api-adaptor** pods by running the following command:

```
$ oc set env ds/peerpodconfig-ctrl-caa-daemon -n openshift-sandboxed-containers-operator REBOOT="$(date)"
```

Restarting the daemonset recreates the peer pods. It does not update the existing pods.

4.2.6. Creating an SSH key secret

You must create an SSH key **secret** object for your KVM host.

Procedure

1. Log in to your OpenShift Container Platform cluster.
2. Generate an SSH key pair by running the following command:

```
$ ssh-keygen -f ./id_rsa -N ""
```

3. Copy the public SSH key to your KVM host:

```
$ ssh-copy-id -i ./id_rsa.pub <KVM_HOST_IP>
```

4. Create the **Secret** object by running the following command:

```
$ oc create secret generic ssh-key-secret \
  -n openshift-sandboxed-containers-operator \
  --from-file=id_rsa.pub=./id_rsa.pub \
  --from-file=id_rsa=./id_rsa
```

The SSH key secret is created.

5. Delete the SSH keys you created:

```
$ shred -remove id_rsa.pub id_rsa
```

4.2.7. Creating a KataConfig custom resource

You must create a **KataConfig** custom resource (CR) to install **kata-remote** as a runtime class on your worker nodes.

Creating the **KataConfig** CR triggers the OpenShift sandboxed containers Operator to do the following:

- Create a **RuntimeClass** CR named **kata-remote** with a default configuration. This enables users to configure workloads to use **kata-remote** as the runtime by referencing the CR in the **RuntimeClassName** field. This CR also specifies the resource overhead for the runtime.

OpenShift sandboxed containers installs **kata-remote** as a *secondary, optional* runtime on the cluster and not as the primary runtime.



IMPORTANT

Creating the **KataConfig** CR automatically reboots the worker nodes. The reboot can take from 10 to more than 60 minutes. Factors that impede reboot time are as follows:

- A larger OpenShift Container Platform deployment with a greater number of worker nodes.
- Activation of the BIOS and Diagnostics utility.
- Deployment on a hard disk drive rather than an SSD.
- Deployment on physical nodes such as bare metal, rather than on virtual nodes.
- A slow CPU and network.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Create a **cluster-kataconfig.yaml** manifest file according to the following example:

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  enablePeerPods: true
  logLevel: info
```

2. Optional: To install **kata-remote** on selected nodes, specify the node labels according to the following example:

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  kataConfigPoolSelector:
    matchLabels:
      <label_key>: '<label_value>' 1
# ...
```

- 1 Specify the labels of the selected nodes.

3. Create the **KataConfig** CR:

```
$ oc create -f cluster-kataconfig.yaml
```

The new **KataConfig** CR is created and installs **kata-remote** as a runtime class on the worker nodes.

Wait for the **kata-remote** installation to complete and the worker nodes to reboot before verifying the installation.

Verification

- Monitor the installation progress by running the following command:

```
$ watch "oc describe kataconfig | sed -n /^Status:./,/^Events/p"
```

When the status of all workers under **kataNodes** is **installed** and the condition **InProgress** is **False** without specifying a reason, the **kata-remote** is installed on the cluster.

See [KataConfig status messages](#) for details.

4.2.8. Optional: Modifying the number of peer pod VMs per node

You can change the limit of peer pod virtual machines (VMs) per node by editing the **peerpodConfig** custom resource (CR).

Procedure

1. Check the current limit by running the following command:

```
$ oc get peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-operator \
-o jsonpath='{.spec.limit}'
```

2. Modify the **limit** attribute of the **peerpodConfig** CR by running the following command:

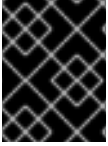
```
$ oc patch peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-containers-operator \
--type merge --patch '{"spec":{"limit":"<value>"}}' 1
```

- 1 Replace <value> with the limit you want to define.

4.2.9. Configuring workload objects

You deploy an OpenShift sandboxed containers workload by configuring **kata-remote** as the runtime class for the following pod-templated objects:

- **Pod** objects
- **ReplicaSet** objects
- **ReplicationController** objects
- **StatefulSet** objects
- **Deployment** objects
- **DeploymentConfig** objects



IMPORTANT

Do not deploy workloads in the **openshift-sandboxed-containers-operator** namespace. Create a dedicated namespace for these resources.

Prerequisites

- You have created a secret object for your provider.
- You have created a config map for your provider.
- You have created a **KataConfig** custom resource (CR).

Procedure

1. Add **spec.runtimeClassName: kata-remote** to the manifest of each pod-templated workload object as in the following example:

```
apiVersion: v1
kind: <object>
# ...
spec:
  runtimeClassName: kata-remote
# ...
```

OpenShift Container Platform creates the workload object and begins scheduling it.

Verification

- Inspect the **spec.runtimeClassName** field of a pod-templated object. If the value is **kata-remote**, then the workload is running on OpenShift sandboxed containers, using peer pods.

CHAPTER 5. MONITORING

You can use the OpenShift Container Platform web console to monitor metrics related to the health status of your sandboxed workloads and nodes.

OpenShift sandboxed containers has a pre-configured dashboard available in the OpenShift Container Platform web console. Administrators can also access and query raw metrics through Prometheus.

5.1. ABOUT METRICS

OpenShift sandboxed containers metrics enable administrators to monitor how their sandboxed containers are running. You can query for these metrics in Metrics UI In the OpenShift Container Platform web console.

OpenShift sandboxed containers metrics are collected for the following categories:

Kata agent metrics

Kata agent metrics display information about the kata agent process running in the VM embedded in your sandboxed containers. These metrics include data from `/proc/<pid>/[io, stat, status]`.

Kata guest operating system metrics

Kata guest operating system metrics display data from the guest operating system running in your sandboxed containers. These metrics include data from `/proc/[stats, diskstats, meminfo, vmstats]` and `/proc/net/dev`.

Hypervisor metrics

Hypervisor metrics display data regarding the hypervisor running the VM embedded in your sandboxed containers. These metrics mainly include data from `/proc/<pid>/[io, stat, status]`.

Kata monitor metrics

Kata monitor is the process that gathers metric data and makes it available to Prometheus. The kata monitor metrics display detailed information about the resource usage of the kata-monitor process itself. These metrics also include counters from Prometheus data collection.

Kata containerd shim v2 metrics

Kata containerd shim v2 metrics display detailed information about the kata shim process. These metrics include data from `/proc/<pid>/[io, stat, status]` and detailed resource usage metrics.

5.2. VIEWING METRICS

You can access the metrics for OpenShift sandboxed containers in the **Metrics** page In the OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role or with view permissions for all projects.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Observe → Metrics**.
2. In the input field, enter the query for the metric you want to observe.
All kata-related metrics begin with **kata**. Typing **kata** displays a list of all available kata metrics.

The metrics from your query are visualized on the page.

Additional resources

- [Querying metrics.](#)
- [Gathering data about your cluster .](#)

CHAPTER 6. UNINSTALLING

You can uninstall OpenShift sandboxed containers by using the OpenShift Container Platform web console or the command line.

Uninstall workflow

1. Delete workload pods.
2. Delete the **KataConfig** custom resource.
3. Uninstall the OpenShift sandboxed containers Operator.
4. Delete the **KataConfig** custom resource definition.

6.1. UNINSTALLING BY USING THE WEB CONSOLE

You can uninstall OpenShift sandboxed containers by using the OpenShift Container Platform web console.

6.1.1. Deleting workload pods


You can delete the OpenShift sandboxed containers workload pods by using the OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have a list of pods that use the OpenShift sandboxed containers runtime class.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Workloads** → **Pods**.
2. Enter the name of the pod that you want to delete in the **Search by name** field.
3. Click the pod name to open it.
4. On the **Details** page, check that **kata** or **kata-remote** is displayed for **Runtime class**.

5. Click the **Options** menu  and select **Delete Pod**.
6. Click **Delete**.

6.1.2. Deleting the KataConfig CR

You can delete the **KataConfig** custom resource (CR) by using the web console. Deleting the **KataConfig** CR removes and uninstalls the **kata** runtime and its related resources from your cluster.



IMPORTANT


Deleting the **KataConfig** CR automatically reboots the worker nodes. The reboot can take from 10 to more than 60 minutes. Factors that impede reboot time are as follows:

- A larger OpenShift Container Platform deployment with a greater number of worker nodes.
- Activation of the BIOS and Diagnostics utility.
- Deployment on a hard drive rather than an SSD.
- Deployment on physical nodes such as bare metal, rather than on virtual nodes.
- A slow CPU and network.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have deleted all running pods that use **kata** as the **runtimeClass**.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators → Installed Operators**.
2. Search for the OpenShift sandboxed containers Operator using the **Search by name** field.
3. Click the Operator to open it, and then select the **KataConfig** tab.
4. Click the **Options** menu  for the **KataConfig** resource, and then select **Delete KataConfig**.
5. Click **Delete** in the confirmation window.

Wait for the **kata** runtime and resources to uninstall and for the worker nodes to reboot before continuing to the next step.

6.1.3. Uninstalling the Operator

You can uninstall the OpenShift sandboxed containers Operator by using OpenShift Container Platform web console. Uninstalling the Operator removes the catalog subscription, Operator group, and cluster service version (CSV) for that Operator. Then, you delete the **openshift-sandboxed-containers-operator** namespace.


Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure


1. In the OpenShift Container Platform web console, navigate to **Operators → Installed Operators**.

2. Enter the OpenShift sandboxed containers Operator name in the **Search by name** field.

3. Click the **Options** menu  for the Operator and select **Uninstall Operator**.

4. Click **Uninstall** in the confirmation window.

5. Enter the **openshift-sandboxed-containers-operator** name in the **Search by name** field.

6. Click the **Options** menu  for the namespace and select **Delete Namespace**.



NOTE

If the **Delete Namespace** option is not available, you do not have permission to delete the namespace.

7. In the **Delete Namespace** window, enter **openshift-sandboxed-containers-operator** and click **Delete**.

8. Click **Delete**.

6.1.4. Deleting the KataConfig CRD

You can delete the **KataConfig** custom resource definition (CRD) by using the OpenShift Container Platform web console.


Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You deleted the **KataConfig** CR.
- You uninstalled the OpenShift sandboxed containers Operator.

Procedure

1. In the web console, navigate to **Administration** → **CustomResourceDefinitions**.

2. Enter the **KataConfig** name in the **Search by name** field.

3. Click the **Options** menu  for the **KataConfig** CRD, and select **Delete CustomResourceDefinition**.

4. Click **Delete** in the confirmation window.

5. Wait for the **KataConfig** CRD to disappear from the list. This can take several minutes.

6.2. UNINSTALLING BY USING THE CLI

You can uninstall OpenShift sandboxed containers by using the command-line interface (CLI).

6.2.1. Deleting workload pods

You can delete the OpenShift sandboxed containers workload pods by using the CLI.

Prerequisites

- You have the JSON processor (**jq**) utility installed.

Procedure

1. Search for the pods by running the following command:

```
$ oc get pods -A -o json | jq -r '.items[] | \
select(.spec.runtimeClassName == "<runtime>").metadata.name'
```

- 1 Specify **kata** for bare metal deployments. Specify **kata-remote** for public cloud, IBM Z®, and IBM® LinuxONE deployments.

2. Delete each pod by running the following command:

```
$ oc delete pod <pod>
```

6.2.2. Deleting the KataConfig CR

You can delete the **KataConfig** custom resource (CR) by using the command line.

Deleting the **KataConfig** CR removes the runtime and its related resources from your cluster.



IMPORTANT

Deleting the **KataConfig** CR automatically reboots the worker nodes. The reboot can take from 10 to more than 60 minutes. Factors that impede reboot time are as follows:

- A larger OpenShift Container Platform deployment with a greater number of worker nodes.
- Activation of the BIOS and Diagnostics utility.
- Deployment on a hard drive rather than an SSD.
- Deployment on physical nodes such as bare metal, rather than on virtual nodes.
- A slow CPU and network.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

- Delete the **KataConfig** CR by running the following command:

–

```
$ oc delete kataconfig <kataconfig>
```

The OpenShift sandboxed containers Operator removes all resources that were initially created to enable the runtime on your cluster.



VERIFICATION

When you delete the **KataConfig** CR, the CLI stops responding until all worker nodes reboot. You must for the deletion process to complete before performing the verification.

- To verify that the **KataConfig** custom resource is deleted, run the following command:

```
$ oc get kataconfig <kataconfig>
```

Example output

```
No KataConfig instances exist
```

6.2.3. Uninstalling the Operator

You can uninstall the OpenShift sandboxed containers Operator by using the CLI. You uninstall the Operator by deleting the Operator subscription, Operator group, cluster service version (CSV), and namespace.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have installed the command-line JSON processor (**jq**).
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Obtain the cluster service version (CSV) name for OpenShift sandboxed containers from the subscription by running the following command:

```
CSV_NAME=$(oc get csv -n openshift-sandboxed-containers-operator -o=custom-columns=:metadata.name)
```

2. Delete the Operatorsubscription from Operator Lifecycle Manager (OLM) by running the following command:

```
$ oc delete subscription sandboxed-containers-operator -n openshift-sandboxed-containers-operator
```

3. Delete the CSV name for OpenShift sandboxed containers by running the following command:

```
$ oc delete csv ${CSV_NAME} -n openshift-sandboxed-containers-operator
```

4. Obtain the Operator group name by running the following command:

```
$ OG_NAME=$(oc get operatorgroup -n openshift-sandboxed-containers-operator -o=jsonpath={..name})
```

5. Delete the Operator group name by running the following command:

```
$ oc delete operatorgroup ${OG_NAME} -n openshift-sandboxed-containers-operator
```

6. Delete the Operator namespace by running the following command:

```
$ oc delete namespace openshift-sandboxed-containers-operator
```

6.2.4. Deleting the KataConfig CRD

You can delete the **KataConfig** custom resource definition (CRD) by using the command line.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You deleted the **KataConfig** CR.
- You uninstalled the OpenShift sandboxed containers Operator.

Procedure

1. Delete the **KataConfig** CRD by running the following command:

```
$ oc delete crd kataconfigs.kataconfiguration.openshift.io
```

Verification

- To verify that the **KataConfig** CRD is deleted, run the following command:

```
$ oc get crd kataconfigs.kataconfiguration.openshift.io
```

Example output

```
Unknown CR KataConfig
```

CHAPTER 7. UPGRADING

The upgrade of the OpenShift sandboxed containers components consists of the following three steps:

1. Upgrade OpenShift Container Platform to update the **Kata** runtime and its dependencies.
2. Upgrade the OpenShift sandboxed containers Operator to update the Operator subscription.

You can upgrade OpenShift Container Platform before or after the OpenShift sandboxed containers Operator upgrade, with the one exception noted below. Always apply the **KataConfig** patch immediately after upgrading OpenShift sandboxed containers Operator.

7.1. UPGRADING RESOURCES

The OpenShift sandboxed containers resources are deployed onto the cluster using Red Hat Enterprise Linux CoreOS (RHCOS) extensions.

The RHCOS extension **sandboxed containers** contains the required components to run OpenShift sandboxed containers, such as the Kata containers runtime, the hypervisor QEMU, and other dependencies. You upgrade the extension by upgrading the cluster to a new release of OpenShift Container Platform.

For more information about upgrading OpenShift Container Platform, see [Updating Clusters](#).

7.2. UPGRADING THE OPERATOR

Use Operator Lifecycle Manager (OLM) to upgrade the OpenShift sandboxed containers Operator either manually or automatically. Selecting between manual or automatic upgrade during the initial deployment determines the future upgrade mode. For manual upgrades, the OpenShift Container Platform web console shows the available updates that can be installed by the cluster administrator.

For more information about upgrading the OpenShift sandboxed containers Operator in Operator Lifecycle Manager (OLM), see [Updating installed Operators](#).

CHAPTER 8. TROUBLESHOOTING

When troubleshooting OpenShift sandboxed containers, you can open a support case and provide debugging information using the **must-gather** tool.

If you are a cluster administrator, you can also review logs on your own, enabling a more detailed level of logs.

8.1. COLLECTING DATA FOR RED HAT SUPPORT

When opening a support case, it is helpful to provide debugging information about your cluster to Red Hat Support.

The **must-gather** tool enables you to collect diagnostic information about your OpenShift Container Platform cluster, including virtual machines and other data related to OpenShift sandboxed containers.

For prompt support, supply diagnostic information for both OpenShift Container Platform and OpenShift sandboxed containers.

Using the must-gather tool

The **oc adm must-gather** CLI command collects the information from your cluster that is most likely needed for debugging issues, including:

- Resource definitions
- Service logs

By default, the **oc adm must-gather** command uses the default plugin image and writes into **./must-gather.local**.

Alternatively, you can collect specific information by running the command with the appropriate arguments as described in the following sections:

- To collect data related to one or more specific features, use the **--image** argument with an image, as listed in a following section.

For example:

```
$ oc adm must-gather --image=registry.redhat.io/openshift-sandboxed-containers/osc-must-gather-rhel9:1.6.0
```

- To collect the audit logs, use the **-- /usr/bin/gather_audit_logs** argument, as described in a following section.

For example:

```
$ oc adm must-gather -- /usr/bin/gather_audit_logs
```



NOTE

Audit logs are not collected as part of the default set of information to reduce the size of the files.

When you run **oc adm must-gather**, a new pod with a random name is created in a new project on the cluster. The data is collected on that pod and saved in a new directory that starts with **must-gather.local**. This directory is created in the current working directory.

For example:

```

NAMESPACE          NAME          READY STATUS  RESTARTS  AGE
...
openshift-must-gather-5drcj  must-gather-bklx4  2/2  Running  0         72s
openshift-must-gather-5drcj  must-gather-s8sdh  2/2  Running  0         72s
...

```

Optionally, you can run the **oc adm must-gather** command in a specific namespace by using the **--run-namespace** option.

For example:

```
$ oc adm must-gather --run-namespace <namespace> --image=registry.redhat.io/openshift-sandboxed-containers/osc-must-gather-rhel9:1.6.0
```

8.2. COLLECTING LOG DATA

The following features and objects are associated with OpenShift sandboxed containers:

- All namespaces and their child objects that belong to OpenShift sandboxed containers resources
- All OpenShift sandboxed containers custom resource definitions (CRDs)

You can collect the following component logs for each pod running with the **kata** runtime:

- Kata agent logs
- Kata runtime logs
- QEMU logs
- Audit logs
- CRI-O logs

8.2.1. Enabling debug logs for CRI-O runtime

You can enable debug logs by updating the **logLevel** field in the **KataConfig** CR. This changes the log level in the CRI-O runtime for the worker nodes running OpenShift sandboxed containers.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Change the **logLevel** field in your existing **KataConfig** CR to **debug**:

```
$ oc patch kataconfig <kataconfig> --type merge --patch '{"spec":{"logLevel":"debug"}}'
```

2. Monitor the **kata-oc** machine config pool until the value of **UPDATED** is **True**, indicating that all worker nodes are updated:

```
$ oc get mcp kata-oc
```

Example output

```
NAME      CONFIG          UPDATED  UPDATING  DEGRADED  MACHINECOUNT
READYMACHINECOUNT  UPDATEDMACHINECOUNT  DEGRADEDMACHINECOUNT
AGE
kata-oc  rendered-kata-oc-169  False   True     False    3          1          1
0          9h
```

Verification

1. Start a debug session with a node in the machine config pool:

```
$ oc debug node/<node_name>
```

2. Change the root directory to **/host**:

```
# chroot /host
```

3. Verify the changes in the **crio.conf** file:

```
# crio config | egrep 'log_level'
```

Example output

```
log_level = "debug"
```

8.2.2. Viewing debug logs for components

Cluster administrators can use the debug logs to troubleshoot issues. The logs for each node are printed to the node journal.

You can review the logs for the following OpenShift sandboxed containers components:

- Kata agent
- Kata runtime (**containerd-shim-kata-v2**)
- **virtiofsd**

QEMU only generates warning and error logs. These warnings and errors print to the node journal in both the Kata runtime logs and the CRI-O logs with an extra **qemuPid** field.

Example of QEMU logs

```
Mar 11 11:57:28 openshift-worker-0 kata[2241647]: time="2023-03-11T11:57:28.587116986Z"
level=info msg="Start logging QEMU (qemuPid=2241693)" name=containerd-shim-v2 pid=2241647
sandbox=d1d4d68efc35e5ccb4331af73da459c13f46269b512774aa6bde7da34db48987
source=virtcontainers/hypervisor subsystem=qemu
```

```
Mar 11 11:57:28 openshift-worker-0 kata[2241647]: time="2023-03-11T11:57:28.607339014Z"
level=error msg="qemu-kvm: -machine q35,accel=kvm,kernel_irqchip=split,foo: Expected '=' after
parameter 'foo'" name=containerd-shim-v2 pid=2241647 qemuPid=2241693
sandbox=d1d4d68efc35e5ccb4331af73da459c13f46269b512774aa6bde7da34db48987
source=virtcontainers/hypervisor subsystem=qemu
```

```
Mar 11 11:57:28 openshift-worker-0 kata[2241647]: time="2023-03-11T11:57:28.60890737Z"
level=info msg="Stop logging QEMU (qemuPid=2241693)" name=containerd-shim-v2 pid=2241647
sandbox=d1d4d68efc35e5ccb4331af73da459c13f46269b512774aa6bde7da34db48987
source=virtcontainers/hypervisor subsystem=qemu
```

The Kata runtime prints **Start logging QEMU** when QEMU starts, and **Stop Logging QEMU** when QEMU stops. The error appears in between these two log messages with the **qemuPid** field. The actual error message from QEMU appears in red.

The console of the QEMU guest is printed to the node journal as well. You can view the guest console logs together with the Kata agent logs.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

- To review the Kata agent logs and guest console logs, run the following command:

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t kata -g "reading guest
console"
```

- To review the Kata runtime logs, run the following command:

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t kata
```

- To review the **virtiofsd** logs, run the following command:

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t virtiofsd
```

- To review the QEMU logs, run the following command:

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t kata -g "qemuPid=\d+"
```

Additional resources

- [Gathering data about your cluster](#) in the OpenShift Container Platform documentation

APPENDIX A. KATACONFIG STATUS MESSAGES

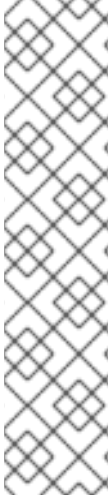
The following table displays the status messages for the **KataConfig** custom resource (CR) for a cluster with two worker nodes.

Table A.1. **KataConfig** status messages

Status	Description
<p>Initial installation</p> <p>When a KataConfig CR is created and starts installing kata on both workers, the following status is displayed for a few seconds.</p>	<pre> conditions: message: Performing initial installation of kata on cluster reason: Installing status: 'True' type: InProgress kataNodes: nodeCount: 0 readyNodeCount: 0 </pre>
<p>Installing</p> <p>Within a few seconds the status changes.</p>	<pre> kataNodes: nodeCount: 2 readyNodeCount: 0 waitingToInstall: - worker-0 - worker-1 </pre>
<p>Installing (Worker-1 installation starting)</p> <p>For a short period of time, the status changes, signifying that one node has initiated the installation of kata, while the other is in a waiting state. This is because only one node can be unavailable at any given time. The nodeCount remains at 2 because both nodes will eventually receive kata, but the readyNodeCount is currently 0 as neither of them has reached that state yet.</p>	<pre> kataNodes: installing: - worker-1 nodeCount: 2 readyNodeCount: 0 waitingToInstall: - worker-0 </pre>
<p>Installing (Worker-1 installed, worker-0 installation started)</p> <p>After some time, worker-1 will complete its installation, causing a change in the status. The readyNodeCount is updated to 1, indicating that worker-1 is now prepared to execute kata workloads. You cannot schedule or run kata workloads until the runtime class is created at the end of the installation process.</p>	<pre> kataNodes: installed: - worker-1 installing: - worker-0 nodeCount: 2 readyNodeCount: 1 </pre>

Status	Description
<p>Installed</p> <p>When installed, both workers are listed as installed, and the InProgress condition transitions to False without specifying a reason, indicating the successful installation of kata on the cluster.</p>	<pre> conditions: message: "" reason: "" status: 'False' type: InProgress kataNodes: installed: - worker-0 - worker-1 nodeCount: 2 readyNodeCount: 2 </pre>

Status	Description
<p>Initial uninstall</p> <p>If kata is installed on both workers, and you delete the KataConfig to remove kata from the cluster, both workers briefly enter a waiting state for a few seconds.</p>	<pre> conditions: message: Removing kata from cluster reason: Uninstalling status: 'True' type: InProgress kataNodes: nodeCount: 0 readyNodeCount: 0 waitingToUninstall: - worker-0 - worker-1 </pre>
<p>Uninstalling</p> <p>After a few seconds, one of the workers starts uninstalling.</p>	<pre> kataNodes: nodeCount: 0 readyNodeCount: 0 uninstalling: - worker-1 waitingToUninstall: - worker-0 </pre>
<p>Uninstalling</p> <p>Worker-1 finishes and worker-0 starts uninstalling.</p>	<pre> kataNodes: nodeCount: 0 readyNodeCount: 0 uninstalling: - worker-0 </pre>



NOTE

The **reason** field can also report the following causes:

- **Failed:** This is reported if the node cannot finish its transition. The **status** reports **True** and the **message** is **Node <node_name> Degraded: <error_message_from_the_node>**.
- **BlockedByExistingKataPods:** This is reported if there are pods running on a cluster that use the **kata** runtime while **kata** is being uninstalled. The **status** field is **False** and the **message** is **Existing pods using "kata" RuntimeClass found. Please delete the pods manually for KataConfig deletion to proceed**. There could also be a technical error message reported like **Failed to list kata pods: <error_message>** if communication with the cluster control plane fails.