Abstract

Information about OpenShift sandboxed containers operators and their usage
Table of Contents

PREFACE .................................................................................................................. 4

CHAPTER 1. UNDERSTANDING OPENSHIFT SANDBOXED CONTAINERS .......... 5
  1.1. OPENSHIFT SANDBOXED CONTAINERS SUPPORTED PLATFORMS .......... 6
  1.2. OPENSHIFT SANDBOXED CONTAINERS COMMON TERMS ....................... 6
  1.3. OPENSHIFT SANDBOXED CONTAINERS WORKLOAD MANAGEMENT ........ 7
    1.3.1. OpenShift sandboxed containers building blocks ............................... 7
    1.3.2. RHCOS extensions ........................................................................ 7
    1.3.3. Virtualization and OpenShift sandboxed containers ......................... 7
  1.4. UNDERSTANDING COMPLIANCE AND RISK MANAGEMENT ................... 8

CHAPTER 2. DEPLOYING OPENSHIFT SANDBOXED CONTAINERS WORKLOADS .... 9
  2.1. PREREQUISITES ......................................................................................... 9
    2.1.1. Resource requirements for OpenShift sandboxed containers .............. 9
    2.1.2. Checking whether cluster nodes are eligible to run OpenShift sandboxed containers 11
  2.2. DEPLOYING OPENSHIFT SANDBOXED CONTAINERS WORKLOADS USING THE WEB CONSOLE .......................................................... 13
    2.2.1. Installing the OpenShift sandboxed containers Operator using the web console 13
    2.2.2. Creating the KataConfig custom resource in the web console ............ 14
    2.2.3. Deploying a workload in a sandboxed container using the web console 16
  2.3. DEPLOYING OPENSHIFT SANDBOXED CONTAINERS WORKLOADS USING THE CLI ............................................................. 17
    2.3.1. Installing the OpenShift sandboxed containers Operator using the CLI .... 17
    2.3.2. Creating the KataConfig custom resource using the CLI .................... 19
    2.3.3. Deploying a workload in a sandboxed container using the CLI ............ 21
  2.4. ADDITIONAL RESOURCES ........................................................................ 23

CHAPTER 3. DEPLOYING OPENSHIFT SANDBOXED CONTAINERS WORKLOADS USING PEER PODS ........................................................ 24
  3.1. PREREQUISITES ......................................................................................... 24
    3.1.1. Resource requirements for peer-pods in OpenShift sandboxed containers 24
    3.1.2. Prerequisites for peer-pods using AWS ............................................. 25
        3.1.2.1. Enable ports 15150 and 9000 for AWS .................................. 25
    3.1.3. Prerequisites for peer-pods using Azure ........................................... 26
        3.1.3.1. Enable ports 15150 and 9000 for Azure ................................ 26
  3.2. DEPLOYING OPENSHIFT SANDBOXED CONTAINERS WORKLOADS USING PEER PODS WITH THE WEB CONSOLE ...................................... 28
    3.2.1. Installing the OpenShift sandboxed containers Operator using the web console 28
    3.2.2. Configuring peer-pod parameters for AWS using the web console ....... 29
        3.2.2.1. Creating a secret object for AWS using the web console ............ 29
        3.2.2.2. Creating an AWS VM image (AMI) using the web console ..... 30
        3.2.2.3. Creating a peer-pod ConfigMap for AWS using the web console 31
    3.2.3. Configuring peer-pod parameters for Azure using the web console ....... 31
        3.2.3.1. Creating a secret object for Azure using the web console .......... 32
        3.2.3.2. Creating an Azure VM image using the web console ............... 33
        3.2.3.3. Creating a peer-pod ConfigMap for Azure using the web console 33
        3.2.3.4. Creating an SSH key secret object for Azure using the web console 34
    3.2.4. Creating the KataConfig custom resource in the web console .......... 35
    3.2.5. Deploying a workload with peer pods in a sandboxed container using the web console 36
  3.3. DEPLOYING OPENSHIFT SANDBOXED CONTAINERS WORKLOADS USING PEER PODS WITH THE CLI ...................................................... 38
    3.3.1. Installing the OpenShift sandboxed containers Operator using the CLI .... 38
    3.3.2. Setting up peer pods for AWS using the CLI ...................................... 40
        3.3.2.1. Creating a secret object for AWS using the CLI ....................... 40
        3.3.2.2. Creating an AWS VM image (AMI) using the CLI ................. 42
3.3.2.3. Creating a peer-pod ConfigMap for AWS using the CLI
3.3.3. Setting up peer pods for Azure using the CLI
3.3.3.1. Creating a secret object for Azure using the CLI
3.3.3.2. Creating an Azure VM image using the CLI
3.3.3.3. Creating a peer-pod ConfigMap for Azure using the CLI
3.3.3.4. Creating an SSH key secret object for Azure using the CLI
3.3.4. Creating the KataConfig custom resource using the CLI
3.3.5. Deploying a workload with peer pods in a sandboxed container using the CLI

3.4. ADDITIONAL RESOURCES

CHAPTER 4. MONITORING OPENSHIFT SANDBOXED CONTAINERS .......................... 51
4.1. ABOUT OPENSHIFT SANDBOXED CONTAINERS METRICS 51
4.2. VIEWING METRICS FOR OPENSHIFT SANDBOXED CONTAINERS 51
4.3. VIEWING THE OPENSHIFT SANDBOXED CONTAINERS DASHBOARD 52
4.4. ADDITIONAL RESOURCES 53

CHAPTER 5. UNINSTALLING OPENSHIFT SANDBOXED CONTAINERS .......................... 54
5.1. UNINSTALLING OPENSHIFT SANDBOXED CONTAINERS USING THE WEB CONSOLE 54
5.1.1. Deleting OpenShift sandboxed containers pods using the web console 54
5.1.2. Deleting the KataConfig custom resource using the web console 54
5.1.3. Deleting the OpenShift sandboxed containers Operator using the web console 55
5.1.4. Deleting the OpenShift sandboxed containers namespace using the web console 56
5.1.5. Deleting the KataConfig custom resource definition using the web console 56
5.2. UNINSTALLING OPENSHIFT SANDBOXED CONTAINERS USING THE CLI 57
5.2.1. Deleting OpenShift sandboxed containers pods using the CLI 57
5.2.2. Deleting the KataConfig custom resource using the CLI 57
5.2.3. Deleting the OpenShift sandboxed containers Operator using the CLI 58
5.2.4. Deleting the KataConfig custom resource definition using the CLI 59

CHAPTER 6. UPGRADING OPENSHIFT SANDBOXED CONTAINERS .......................... 61
6.1. UPGRADING THE OPENSHIFT SANDBOXED CONTAINERS RESOURCES 61
6.2. UPGRADING THE OPENSHIFT SANDBOXED CONTAINERS OPERATOR 61
6.3. UPGRADING THE OPENSHIFT SANDBOXED CONTAINERS MONITOR PODS 61
6.3.1. Upgrading the monitor pods using the web console 62
6.3.2. Upgrading the monitor pods using the CLI 62

CHAPTER 7. COLLECTING OPENSHIFT SANDBOXED CONTAINERS DATA ......................... 63
7.1. COLLECTING OPENSHIFT SANDBOXED CONTAINERS DATA FOR RED HAT SUPPORT 63
7.1.1. About the must-gather tool 63
7.2. ABOUT OPENSHIFT SANDBOXED CONTAINERS LOG DATA 64
7.3. ENABLING DEBUG LOGS FOR OPENSHIFT SANDBOXED CONTAINERS 64
7.3.1. Viewing debug logs for OpenShift sandboxed containers 65
7.4. ADDITIONAL RESOURCES 67
PREFACE
CHAPTER 1. UNDERSTANDING OPENSHIFT SANDBOXED CONTAINERS

OpenShift sandboxed containers support for Red Hat OpenShift provides you with built-in support for running Kata Containers as an additional optional runtime. The new runtime supports containers in dedicated virtual machines (VMs), providing improved workload isolation. This is particularly useful for performing the following tasks:

Run privileged or untrusted workloads
OpenShift sandboxed containers (OSC) makes it possible to safely run workloads that require specific privileges, without having to risk 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 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 may 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 kernel isolation for each workload
OpenShift sandboxed containers supports 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
OpenShift sandboxed containers enables you to support multiple users (tenants) from different organizations sharing the same OpenShift cluster. The system also lets you run 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 use OpenShift sandboxed containers to run a containerized workload with known vulnerabilities or to handle an issue in a legacy application. This isolation also 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. Kernel packet filtering 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, resources such as CPU, memory, storage, or networking are managed in a more robust and secure way in OpenShift sandboxed containers. Since OpenShift sandboxed containers are deployed 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 allocate 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.

1.1. OPENSHIFT SANDBOXED CONTAINERS SUPPORTED PLATFORMS

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

Red Hat Enterprise Linux CoreOS (RHCOS) is the only supported operating system for OpenShift sandboxed containers. OpenShift sandboxed containers 1.4 runs on Red Hat Enterprise Linux CoreOS (RHCOS) 8.6.

OpenShift sandboxed containers 1.4 is compatible with Red Hat OpenShift 4.11.

1.2. OPENSHIFT SANDBOXED CONTAINERS 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 Red Hat OpenShift. 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 Red Hat OpenShift.

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.

### 1.3. OPENSHIFT SANDBOXED CONTAINERS WORKLOAD MANAGEMENT

OpenShift sandboxed containers provides the following features for enhancing workload management and allocation:

#### 1.3.1. OpenShift sandboxed containers building blocks

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.

#### 1.3.2. RHCOS extensions

The OpenShift sandboxed containers Operator is based on the Red Hat Enterprise Linux CoreOS (RHCOS) extensions concept. Red Hat Enterprise Linux CoreOS (RHCOS) extensions are a mechanism to install optional Red Hat OpenShift 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

#### 1.3.3. Virtualization and OpenShift sandboxed containers

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

To run OpenShift Virtualization and OpenShift sandboxed containers at the same time, you must enable VMs to migrate, so that they do not block node reboots. Configure the following parameters on your VM:

- Use `ocs-storagecluster-ceph-rbd` as the storage class.

- Set the `evictionStrategy` parameter to `LiveMigrate` in the VM.
Additional resources

- Configuring local storage for virtual machines
- Configuring virtual machine eviction strategy

1.4. UNDERSTANDING COMPLIANCE AND RISK MANAGEMENT

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 new peer pods runtime class kata-remote-cc 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 Red Hat OpenShift deployments on the x86_64 architecture.

You can install the OpenShift sandboxed containers Operator using either the web console or OpenShift CLI (`oc`). Before installing the OpenShift sandboxed containers Operator, you must prepare your Red Hat OpenShift cluster.

### 2.1. PREREQUISITES

Before you install OpenShift sandboxed containers, ensure that your Red Hat OpenShift cluster meets the following requirements:

- Your cluster must be installed on on-premise bare-metal infrastructure with Red Hat Enterprise Linux CoreOS (RHCOS) workers. You can use any installation method including user-provisioned, installer-provisioned, or assisted installer to deploy your cluster.

**NOTE**

- OpenShift sandboxed containers only supports RHCOS worker nodes. RHEL nodes are not supported.
- Nested virtualization is not supported.

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

### 2.1.1. Resource requirements for OpenShift sandboxed containers

OpenShift sandboxed containers lets users run workloads on their Red Hat OpenShift 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.
<table>
<thead>
<tr>
<th>Resource</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory allocated by default to a virtual machine</td>
<td>2Gi</td>
</tr>
<tr>
<td>Guest Linux kernel memory usage at boot</td>
<td>~110Mi</td>
</tr>
<tr>
<td>Memory used by the QEMU process (excluding VM memory)</td>
<td>~30Mi</td>
</tr>
<tr>
<td>Memory used by the <code>virtiofsd</code> process (excluding VM I/O buffers)</td>
<td>~10Mi</td>
</tr>
<tr>
<td>Memory used by the <code>containerd-shim-kata-v2</code> process</td>
<td>~20Mi</td>
</tr>
<tr>
<td>File buffer cache data after running <code>dnf install</code> on Fedora</td>
<td>~300Mi* [1]</td>
</tr>
</tbody>
</table>

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 being used in all three cases. In other words, the total memory usage is only 300Mi, mapped in three different places. This is correctly accounted for when reporting the memory utilization metrics.

Additional resources

- Installing a user-provisioned cluster on bare metal

2.1.2. Checking whether cluster nodes are eligible to run OpenShift sandboxed containers

Before running OpenShift sandboxed containers, you can check whether the nodes in your cluster are eligible to run Kata containers. Some cluster nodes might not comply with sandboxed containers’ minimum requirements. The most common reason for node ineligibility is the lack of virtualization support on the node. If you attempt to run sandboxed workloads on ineligible nodes, you will experience errors. You can use the Node Feature Discovery (NFD) Operator and a NodeFeatureDiscovery resource to automatically check node eligibility.

NOTE

If you want 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 resource.

Alternatively, to install the Kata runtime on all worker nodes, set checkNodeEligibility: false in the KataConfig resource.

In both these scenarios, you do not need to create the NodeFeatureDiscovery resource. 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 Kata 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

- Install the OpenShift CLI (oc).
- Log in as a user with cluster-admin privileges.
- Install the Node Feature Discovery (NFD) Operator.
Procedure

1. Create a **NodeFeatureDiscovery** resource to detect node capabilities suitable for running Kata containers:
   a. Save the following YAML in the **nfd.yaml** file:

   ```yaml
   apiVersion: nfd.openshift.io/v1
   kind: NodeFeatureDiscovery
   metadata:
     name: nfd-kata
     namespace: openshift-nfd
   spec:
     operand:
       image: quay.io/openshift/origin-node-feature-discovery:4.10
       imagePullPolicy: Always
       servicePort: 12000
     workerConfig:
       configData: |
       sources:
         custom:
           - name: "feature.node.kubernetes.io/runtime.kata"
             matchOn:
               cpuId: ["SSE4", "VMX"]
               loadedKMod: ["kvm", "kvm_intel"]
               - cpuId: ["SSE4", "SVM"]
               loadedKMod: ["kvm", "kvm_amd"]
   
   b. Create the **NodeFeatureDiscovery** custom resource (CR):

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

   **Example output**

   ```
   nodefeaturediscovery.nfd.openshift.io/nfd-kata created
   ```

   A **feature.node.kubernetes.io/runtime.kata=true** label is applied to all qualifying worker nodes.

2. Set the **checkNodeEligibility** field to **true** in the **KataConfig** resource to enable the feature, for example:
   a. Save the following YAML in the **kata-config.yaml** file:

   ```yaml
   apiVersion: kataconfiguration.openshift.io/v1
   kind: KataConfig
   metadata:
     name: example-kataconfig
   spec:
     checkNodeEligibility: true
   
   b. Create the **KataConfig** CR:

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

```
kataconfig.kataconfiguration.openshift.io/example-kataconfig created
```

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

```
<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
<th>ROLES</th>
<th>AGE</th>
<th>VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>compute-3.example.com</td>
<td>Ready</td>
<td>worker</td>
<td>4h38m</td>
<td>v1.25.0</td>
</tr>
<tr>
<td>compute-2.example.com</td>
<td>Ready</td>
<td>worker</td>
<td>4h35m</td>
<td>v1.25.0</td>
</tr>
</tbody>
</table>
```

Additional resources

- For more information about installing the Node Feature Discovery (NFD) Operator, see Installing NFD.

### 2.2. DEPLOYING OPENSIFT SANDBOXED CONTAINERS WORKLOADS USING THE WEB CONSOLE

You can deploy OpenShift sandboxed containers workloads from the web console. First, you must install the OpenShift sandboxed containers Operator, then create the *KataConfig* custom resource (CR). Once you are ready to deploy a workload in a sandboxed container, you must manually add *kata* as the *runtimeClassName* to the workload YAML file.

#### 2.2.1. Installing the OpenShift sandboxed containers Operator using the web console

You can install the OpenShift sandboxed containers Operator from the Red Hat OpenShift web console.

**Prerequisites**

- You have Red Hat OpenShift 4.12 installed.
- You have access to the cluster as a user with the *cluster-admin* role.

**Procedure**

1. From the **Administrator** perspective in the web console, navigate to **Operators** → **OperatorHub**.
2. In the **Filter by keyword** field, type **OpenShift sandboxed containers**.
3. Select the **OpenShift sandboxed containers** tile.
4. Read the information about the Operator and click **Install**.
5. On the **Install Operator** page:
a. Select **stable** from the list of available **Update Channel** options.

b. 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.

c. 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.

6. Click **Install**.

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

**Verification**

1. From the **Administrator** perspective in the web console, navigate to **Operators → Installed Operators**.

2. Verify that the OpenShift sandboxed containers Operator is listed in the in operators list.

### 2.2.2. Creating the KataConfig custom resource in the web console

You must create one **KataConfig** custom resource (CR) to enable installing **kata** as a **RuntimeClass** on your cluster nodes.

**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 Red Hat OpenShift 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 installed Red Hat OpenShift 4.12 on your cluster.
- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift sandboxed containers Operator.
NOTE

Kata is installed on all worker nodes by default. If you want to install kata as a RuntimeClass only on specific nodes, you can add labels to those nodes, then define the label in the KataConfig CR when you create it.

Procedure

1. From the Administrator perspective in the web console, navigate to Operators → Installed Operators.

2. Select the OpenShift sandboxed containers Operator from the list of operators.

3. In the KataConfig tab, click Create KataConfig.

4. In the Create KataConfig page, enter the following details:
   - **Name**: Enter a name for the KataConfig resource. By default, the name is defined as example-kataconfig.
   - **Labels** (Optional): Enter any relevant, identifying attributes to the KataConfig resource. Each label represents a key-value pair.
   - **checkNodeEligibility** (Optional, not relevant for peer pods): Select this checkbox to use the Node Feature Discovery Operator (NFD) to detect node eligibility to run kata as a RuntimeClass. For more information, see “Checking whether cluster nodes are eligible to run OpenShift sandboxed containers”.
   - **enablePeerPods** (For peer pods): Select this checkbox to enable peer pods and use OpenShift sandboxed containers in a public cloud environment.
   - **kataConfigPoolSelector**: By default, kata is installed as a RuntimeClass on all nodes. If you want to install kata as a RuntimeClass only on selected nodes, you must add a matchExpression:
     a. Expand the kataConfigPoolSelector area.
     b. In the kataConfigPoolSelector, expand matchExpressions. This is a list of label selector requirements.
     c. Click Add matchExpressions.
     d. In the key field, add the label key the selector applies to.
     e. In the operator field, add 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 running kata as a RuntimeClass. For more information, see “Collecting OpenShift sandboxed containers data”.

5. Click Create.
The new KataConfig CR is created and begins to install kata as a RuntimeClass on the worker nodes. Wait for the kata installation to complete and the worker nodes to reboot before continuing to the next step.

**IMPORTANT**

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

**Verification**

1. In the KataConfig tab, select the new KataConfig CR.
2. In the KataConfig page, select the YAML tab.
3. Monitor the installationStatus field in the status.
   A message appears each time there is an update. Click Reload to view the updated KataConfig CR.

   Once the value of Completed nodes equals the number of worker or labeled nodes, the installation is complete. The status also contains a list of nodes where the installation is completed.

**2.2.3. Deploying a workload in a sandboxed container using the web console**

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

To deploy a pod-templated workload in a sandboxed container, you must manually add kata as the runtimeClassName to the workload YAML file.

**Prerequisites**

- You have installed Red Hat OpenShift 4.12 on your cluster.
- You have access to the cluster as a user with the cluster-admin role.
- You have installed the OpenShift sandboxed containers Operator.
- You have created a KataConfig custom resource (CR).

**Procedure**

1. From the Administrator perspective in the web console, expand Workloads and select the type of workload you want to create.
2. In the workload page, click to create the workload.
3. In the YAML file for the workload, in the spec field where the container is listed, add runtimeClassName: kata.

**Example for Pod object**

```yaml
apiVersion: v1
custom: Pod
```
You can deploy OpenShift sandboxed containers workloads using the CLI. First, you must install the OpenShift sandboxed containers Operator, then create the `KataConfig` custom resource. Once you are ready to deploy a workload in a sandboxed container, you must add `kata` as the `runtimeClassName` to the workload YAML file.

### 2.3.1. Installing the OpenShift sandboxed containers Operator using the CLI

You can install the OpenShift sandboxed containers Operator using the Red Hat OpenShift CLI.
Prerequisites

- You have Red Hat OpenShift 4.12 installed on your cluster.
- You have installed the OpenShift CLI (oc).
- You have access to the cluster as a user with the cluster-admin role.
- You have subscribed to the OpenShift sandboxed containers catalog.

NOTE

Subscribing to the OpenShift sandboxed containers catalog provides openshift-sandboxed-containers-operator namespace access to the OpenShift sandboxed containers Operator.

Procedure

1. Create the Namespace object for the OpenShift sandboxed containers Operator.
   a. Create a Namespace object YAML file that contains the following manifest:

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

   b. Create the Namespace object:

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

2. Create the OperatorGroup object for the OpenShift sandboxed containers Operator.
   a. Create an OperatorGroup object YAML file that contains the following manifest:

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

   b. Create the OperatorGroup object:

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

3. Create the Subscription object to subscribe the Namespace to the OpenShift sandboxed containers Operator.
   a. Create a Subscription object YAML file that contains the following manifest:

```yaml
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
```
Create the Subscription object:

$ oc create -f Subscription.yaml

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

NOTE

All the object file names listed above are suggestions. You can create the object YAML files using other names.

Verification

- Ensure that the Operator is correctly installed:

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

Example output

<table>
<thead>
<tr>
<th>NAME</th>
<th>DISPLAY</th>
<th>VERSION</th>
<th>REPLACES</th>
<th>PHASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>openshift-sandboxed-containers</td>
<td>openshift-sandboxed-containers-operator</td>
<td>1.4.0</td>
<td>1.3.3</td>
<td>Succeeded</td>
</tr>
</tbody>
</table>

Additional resources

- Installing from OperatorHub using the CLI

2.3.2. Creating the KataConfig custom resource using the CLI

You must create one KataConfig custom resource (CR) to install kata as a RuntimeClass on your nodes. Creating the KataConfig CR triggers the OpenShift sandboxed containers Operator to do the following:

- Install the needed 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.
NOTE
Kata is installed on all worker nodes by default. If you want to install kata as a RuntimeClass only on specific nodes, you can add labels to those nodes, and then define the label in the KataConfig CR when you create it.

Prerequisites
- You have installed Red Hat OpenShift 4.12 on your cluster.
- You have installed the OpenShift CLI (oc).
- You have access to the cluster as a user with the cluster-admin role.
- You have installed the OpenShift sandboxed containers Operator.

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 Red Hat OpenShift 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.

Procedure
1. Create a YAML file with the following manifest:

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

Set `checkNodeEligibility` to true to detect node eligibility to run kata as a RuntimeClass. For more information, see "Checking whether cluster nodes are eligible to run OpenShift sandboxed containers".

2. (Optional) If you want to install kata as a RuntimeClass only on selected nodes, create a YAML file that includes the label in the manifest:

```yaml
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
```
Labels in `kataConfigPoolSelector` only support single values; `nodeSelector` syntax is not supported.

3. Create the `KataConfig` resource:

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

   The new `KataConfig` CR is created and begins to install `kata` as a `RuntimeClass` on the worker nodes. Wait for the `kata` installation to complete and the worker nodes to reboot before continuing to the next step.

   **IMPORTANT**

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

   **Verification**

   - Monitor the installation progress:

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

   Once the value of `Is In Progress` appears as `false`, the installation is complete.

   **Additional resources**

   - Understanding how to update labels on nodes

2.3.3. Deploying a workload in a sandboxed container using the CLI

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

To deploy a pod-templated workload in a sandboxed container, you must add `kata` as the `runtimeClassName` to the workload YAML file.

**Prerequisites**

- You have installed Red Hat OpenShift 4.12 on your cluster.
- You have installed the OpenShift CLI (`oc`).
- You have access to the cluster as a user with the `cluster-admin` role.
- You have installed the OpenShift sandboxed containers Operator.
- You have created a `KataConfig` custom resource (CR).
Procedure

- Add **runtimeClassName: kata** to any pod-templated object:
  - **Pod** objects
  - **ReplicaSet** objects
  - **ReplicationController** objects
  - **StatefulSet** objects
  - **Deployment** objects
  - **DeploymentConfig** objects

Example for Pod object

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
runtimeClassName: kata
```

Example for Deployment object

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mypod
labels:
  app: mypod
spec:
  replicas: 3
selector:
  matchLabels:
    app: mypod
template:
  metadata:
    labels:
      app: mypod
spec:
  runtimeClassName: kata
  containers:
    - name: mypod
      image: myImage
```

Red Hat OpenShift creates the workload and begins scheduling it.

Verification

- Inspect the **runtimeClassName** field on a pod-templated object. If the **runtimeClassName** is **kata**, then the workload is running on OpenShift sandboxed containers.
2.4. ADDITIONAL RESOURCES

- The OpenShift sandboxed containers Operator is supported in a restricted network environment. For more information, see Using Operator Lifecycle Manager on restricted networks.

- When using a disconnected cluster on a restricted network, you must configure proxy support in Operator Lifecycle Manager to access the OperatorHub. Using a proxy allows the cluster to fetch the OpenShift sandboxed containers Operator.
CHAPTER 3. DEPLOYING OPENSHIFT SANBOXED CONTAINERS WORKLOADS USING PEER PODS

You can install the OpenShift sandboxed containers Operator using either the web console or OpenShift CLI (oc). Before installing the OpenShift sandboxed containers Operator, you must prepare your Red Hat OpenShift cluster.

**IMPORTANT**

Deploying OpenShift sandboxed containers workloads using peer pods 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.

### 3.1. PREREQUISITES

Before you install OpenShift sandboxed containers and enable peer pods, you must meet the following requirements:

- You have Red Hat OpenShift 4.12 installed on AWS or Azure.
- You have installed the OpenShift CLI (oc).
- You have access to the cluster as a user with the cluster-admin role.

#### 3.1.1. Resource requirements for peer-pods in OpenShift sandboxed containers

A peer pod uses resources in two locations:

- The Kubernetes worker node. The worker node stores metadata, Kata shim resources (containerd-shim-kata-v2), remote-hypervisor (cloud-api-adaptor) resources, 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 a pod overhead included in the RuntimeClass (kata-remote-cc) definition used for creating peer pods.

The total capacity of the peer-pod VMs running in the cloud is defined as Kubernetes Node extended resources. This enables the Kubernetes scheduler to handle capacity tracking and accounting. The extended resource is named kata.peerpods.io/vm, and defines the maximum number of peer pods that you can create in the cluster.

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.

3. The webhook adds an annotation to the pod that specifies the instance type that is used when the VM is created in the cloud environment.

**NOTE**

The mutating webhook excludes specific system namespaces in Red Hat OpenShift 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.

### 3.1.2. Prerequisites for peer-pods using AWS

If you are using AWS to create peer pods, you must ensure the following requirements:

- Your Red Hat OpenShift cluster must be installed on AWS, with at least one worker node.

- You have access to `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` credentials. These are used to create additional cloud instances in the same virtual private cloud (VPC) of the cluster.

- You must have the AWS CLI tool installed and configured.

- You must enable internal cluster communication on ports 15150 and 9000. You can enable these ports in either the AWS web console or using the CLI.

#### 3.1.2.1. Enable ports 15150 and 9000 for AWS

**Procedure**

1. Retrieve the Instance ID:

   ```sh
   $ 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.azure.region}')

3. Retrieve the security groups:

$ SG=$(aws ec2 describe-instances --instance-ids ${INSTANCE_ID} --query 'Reservations[0].Instances[0].SecurityGroups[*].GroupId' --output text --region ${AWS_REGION})

4. Authorize the peer-pods shim and to access kata-agent communication. Run the following command:

$ aws ec2 authorize-security-group-ingress --group-id ${SG} --protocol tcp --port 15150 --source-group ${SG} --region ${AWS_REGION}

$ aws ec2 authorize-security-group-ingress --group-id ${SG} --protocol tcp --port 9000 --source-group ${SG} --region ${AWS_REGION}

5. Set up the peer-pods tunnel. Run the following command:

The ports are now enabled.

3.1.3. Prerequisites for peer-pods using Azure

If you are using Microsoft Azure to create peer pods, you must ensure the following requirements:

- Your Red Hat OpenShift cluster must be installed on Azure, with at least one worker node.
- You have access to the following credentials and subscription details:
  - AZURE_SUBSCRIPTION_ID
  - AZURE_CLIENT_ID
  - AZURE_CLIENT_SECRET
  - AZURE_TENANT_ID

These are used to create additional cloud instances in the same virtual private cloud (VPC) of the cluster.

- You must have the Azure CLI tool installed and configured.
- You must enable cluster communication on ports 15150 and 9000. In Azure, internal communication is allowed on these ports. However, if communication is blocked, you can enable the ports in either the Azure web console or using the CLI.

3.1.3.1. Enable ports 15150 and 9000 for Azure

Procedure

1. Retrieve the Instance ID:
2. Retrieve the Azure resource group:

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

3. Retrieve the Azure network security group (NSG) name:

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

4. Retrieve the Azure VNet name:

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

5. Retrieve the Azure subnet name:

```bash
$ AZURE_SUBNET_NAME=$(az network vnet subnet list --resource-group ${AZURE_RESOURCE_GROUP} --vnet-name ${AZURE_VNET_NAME} --query "[].{Name:name} | [% contains(Name, 'worker')]") --output tsv)
```

6. Retrieve the Azure subnet prefix:

```bash
$ AZURE_SUBNET_PREFIX=$(az network vnet subnet show --name ${AZURE_SUBNET_NAME} --vnet-name ${AZURE_VNET_NAME} --resource-group ${AZURE_RESOURCE_GROUP} --query "addressPrefix" --output tsv)
```

7. Authorize the peer-pods shim to access kata-agent communication. Run the following command:

```bash
$ az network nsg rule create
--resource-group $AZURE_RESOURCE_GROUP
--nsg-name $AZURE_NSG_NAME
--name Allow-Kata-Agent-Internal
--access Allow
--protocol Tcp
--direction Inbound
--priority 112
--source-address-prefixes $AZURE_SUBNET_PREFIX
--source-port-range ***
--destination-address-prefixes $AZURE_SUBNET_PREFIX
--destination-port-range 15150
```

8. Set up the peer-pods tunnel. Run the following command:

```bash
$ az network nsg rule create
--resource-group $AZURE_RESOURCE_GROUP
--nsg-name $AZURE_NSG_NAME
--name Allow-VXLAN-Internal
--access Allow
```
3.2. DEPLOYING OPENSHIFT SANDBOXED CONTAINERS WORKLOADS USING PEER PODS WITH THE WEB CONSOLE

You can deploy OpenShift sandboxed containers workloads from the web console. First, you must install the OpenShift sandboxed containers Operator, then create a secret object, the VM image, and a peer-pod ConfigMap. The secret object and ConfigMap are unique, depending on your cloud provider. Finally, you must create the KataConfig custom resource (CR). Once you are ready to deploy a workload in a sandboxed container, you must manually add kata-remote-cc as the runtimeClassName to the workload YAML file.

3.2.1. Installing the OpenShift sandboxed containers Operator using the web console

You can install the OpenShift sandboxed containers Operator from the Red Hat OpenShift web console.

Prerequisites

- You have Red Hat OpenShift 4.12 installed.
- You have access to the cluster as a user with the cluster-admin role.

Procedure

1. From the Administrator perspective in the web console, navigate to Operators → OperatorHub.
2. In the Filter by keyword field, type OpenShift sandboxed containers.
3. Select the OpenShift sandboxed containers tile.
4. Read the information about the Operator and click Install.
5. On the Install Operator page:
   a. Select stable from the list of available Update Channel options.
   b. 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.

The ports are now enabled.
NOTE

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

c. 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.

6. Click Install.

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

Verification

1. From the Administrator perspective in the web console, navigate to Operators → Installed Operators.

2. Verify that the OpenShift sandboxed containers Operator is listed in the in operators list.

3.2.2. Configuring peer-pod parameters for AWS using the web console

You must create a secret object and a ConfigMap to deploy OpenShift sandboxed containers using peer pods on AWS.

After creating the secret object, you must still create the KataConfig custom resource (CR) to deploy OpenShift sandboxed containers using peer pods.

Prerequisites

- You have installed Red Hat OpenShift 4.12 on your cluster.
- You have access to the cluster as a user with the cluster-admin role.
- You have installed the OpenShift sandboxed containers Operator.

3.2.2.1. Creating a secret object for AWS using the web console

Set your AWS access keys and configure your network in a secret object. The secret object is used to create the pod VM image and used by peer pods.

When creating a secret object for AWS, you must set specific environment values. You can retrieve some of these values before creating the secret object. Retrieving these values must be done using the CLI. For more information, see Creating a secret object for AWS using the CLI.

In addition, in the AWS web console, you must locate and prepare the following values:

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`

Procedure
1. From the Administrator perspective in the web console, navigate to Operators → Installed Operators.

2. Select the OpenShift sandboxed containers Operator from the list of operators.

3. Click the Import icon (+) in the top right corner.

4. In the Import YAML window, paste the following YAML manifest:

   ```yaml
   apiVersion: v1
   kind: Secret
   metadata:
     name: peer-pods-secret
     namespace: openshift-sandboxed-containers-operator
     type: Opaque
   stringData:
     AWS_ACCESS_KEY_ID: "<enter value>"  # 1
     AWS_SECRET_ACCESS_KEY: "<enter value>"  # 2
     AWS_REGION: "<enter value>"  # 3
     AWS_SUBNET_ID: "<enter value>"  # 4
     AWS_VPC_ID: "<enter value>"  # 5
     AWS_SG_IDS: "<enter value>"  # 6
   
   1. Enter the AWS_ACCESS_KEY_ID value you prepared before you began.
   2. Enter the AWS_SECRET_ACCESS_KEY value you prepared before you began.
   3. Enter the AWS_REGION value you retrieved.
   4. Enter the AWS_SUBNET_ID value you retrieved.
   5. Enter the AWS_VPC_ID value you retrieved.
   6. Enter the AWS_SG_IDS value you retrieved.

5. Click Create.

The secret object is created. You can see it listed under Workloads → Secrets.

3.2.2.2. Creating an AWS VM image (AMI) using the web console

To run OpenShift sandboxed containers using peer pods on AWS, you must first create a RHEL AMI using your AWS account and resources.

Procedure

1. From the Administrator perspective in the web console, navigate to Workloads → Jobs.

2. In the Jobs window, in the top left corner, verify that you are in the openshift-sandboxed-containers-operator project.

3. Click Create Job.

4. In the Create Job window, paste this complete YAML manifest.
5. Click **Create**.

The image is created.

**NOTE**

This image is not managed by OpenShift sandboxed containers. You can delete it if necessary using the AWS web console or AWS CLI tool.

Once the image is created, you must set the image using a peer-pod ConfigMap.

### 3.2.2.3. Creating a peer-pod ConfigMap for AWS using the web console

When creating a ConfigMap for AWS, you must set the AMI ID. You can retrieve this value before you create the ConfigMap. Retrieving this value must be done using the CLI. For more information, see [Creating a peer-pod ConfigMap for AWS using the CLI](#).

**Procedure**

1. From the *Administrator* perspective in the web console, navigate to **Operators** → **Installed Operators**.

2. Select the OpenShift sandboxed containers Operator from the list of operators.

3. Click the Import icon (+) in the top right corner.

4. In the **Import YAML** window, paste the following YAML manifest:

   ```yaml
   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.small"
     PROXY_TIMEOUT: "5m"
     PODVM_AMI_ID: "<enter value>" ①
   
   ① Enter the `PODVM_AMI_ID` value you retrieved.
   
5. Click **Create**.

The ConfigMap object is created. You can see it listed under **Workloads** → **ConfigMaps**.

Once you create the **KataConfig** CR, you can run OpenShift sandboxed containers using peer pods on AWS.

### 3.2.3. Configuring peer-pod parameters for Azure using the web console

You must create a secret object and a ConfigMap in order to deploy OpenShift sandboxed containers using peer pods on Microsoft Azure.
After creating the secret object, you must still create the KataConfig custom resource (CR) to deploy OpenShift sandboxed containers using peer pods.

Prerequisites

- You have installed Red Hat OpenShift 4.12 on your cluster.
- You have access to the cluster as a user with the cluster-admin role.
- You have installed the OpenShift sandboxed containers Operator.

3.2.3.1. Creating a secret object for Azure using the web console

Set your Azure access keys and configure your network in a secret object. The secret object is used to create the pod VM image and used by peer pods.

When creating a secret object for Azure, you must set specific environment values. You can retrieve some of these values before creating the secret object. Retrieving these values must be done using the CLI. For more information, see Creating a secret object for Azure using the CLI.

In addition, in the Azure web console, you must locate and prepare the following values:

- **AZURE_CLIENT_ID**
- **AZURE_CLIENT_SECRET**
- **AZURE_TENANT_ID**
- **AZURE_SUBSCRIPTION_ID**
- **AZURE_REGION**
- **AZURE_RESOURCE_GROUP**

Procedure

1. From the Administrator perspective in the web console, navigate to Operators → Installed Operators.
2. Select the OpenShift sandboxed containers Operator from the list of operators.
3. Click the Import icon (+) in the top right corner.
4. In the Import YAML window, paste the following YAML manifest:

```yaml
apiVersion: v1
class: Secret
metadata:
  name: peer-pods-secret
  namespace: openshift-sandboxed-containers-operator
type: Opaque
stringData:
  AZURE_CLIENT_ID: "<enter value>" 1
  AZURE_CLIENT_SECRET: "<enter value>" 2
  AZURE_TENANT_ID: "<enter value>" 3
  AZURE_SUBSCRIPTION_ID: "<enter value>" 4
  AZURE_REGION: "<enter value>" 5
  AZURE_RESOURCE_GROUP: "<enter value>" 6
```

1. Enter the AZURE_CLIENT_ID value you prepared before you began.
Enter the `AZURE_CLIENT_SECRET` value you prepared before you began.

Enter the `AZURE_TENANT_ID` value you prepared before you began.

Enter the `AZURE_SUBSCRIPTION_ID` value you retrieved.

Enter the `AZURE_REGION` value you retrieved.

Enter the `AZURE_RESOURCE_GROUP` value you retrieved.

5. Click Create.

The secret object is created. You can see it listed under Workloads → Secrets.

### 3.2.3.2. Creating an Azure VM image using the web console

To run OpenShift sandboxed containers using peer pods on Azure, you must first create a RHEL image for Azure using your Azure account and resources.

**Procedure**

1. From the Administrator perspective in the web console, navigate to Workloads → Jobs.

2. In the Jobs window, in the top left corner, verify that you are in the openshift-sandboxed-containers-operator project.

3. Click Create Job.

4. In the Create Job window, paste this complete YAML manifest.

5. Click Create.

The image is created.

**NOTE**

This image is not managed by OpenShift sandboxed containers. You can delete it if necessary using the Azure web console or Azure CLI tool.

Once the image is created, you must set the image using a peer-pod ConfigMap.

### 3.2.3.3. Creating a peer-pod ConfigMap for Azure using the web console

When creating a ConfigMap for Azure, you must set specific configuration values. Retrieving these values must be done using the CLI. For more information, see Creating a peer-pod ConfigMap for Azure using the CLI.

**Procedure**

1. From the Administrator perspective in the web console, navigate to Operators → Installed Operators.

2. Select the OpenShift sandboxed containers Operator from the list of operators.
3. Click the import icon (+) in the top right corner.

4. In the Import YAML window, paste the following YAML manifest:

   ```yaml
   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_D8as_v5"
     AZURE_SUBNET_ID: "{enter value}"  
     AZURE_NSG_ID: "{enter value}"  
     AZURE_IMAGE_ID: "{enter value}"  
     PROXY_TIMEOUT: "5m"
     DISABLECVM: "true"
   ```

   1. Enter the **AZURE_SUBNET_ID** value you retrieved.
   2. Enter the **AZURE_NSG_ID** value you retrieved.
   3. Enter the **AZURE_IMAGE_ID** value you retrieved.

5. Click **Create**.

The ConfigMap object is created. You can see it listed under **Workloads → ConfigMaps**.

### 3.2.3.4. Creating an SSH key secret object for Azure using the web console

You must create an SSH key secret object to use peer pods with Azure. If you do not already have an SSH key to create the object, you must generate one using the CLI. For more information, see **Procedure**

1. From the Administrator perspective in the web console, navigate to **Workloads → Secrets**.

2. In the **Secrets** window, in the top left corner, verify that you are in the **openshift-sandboxed-containers-operator** project.

3. Click **Create** and select **Key/value secret** from the list.

4. In the **Secret name** field, enter **ssh-key-secret**.

5. In the **Key** field, enter **id_rsa.pub**.

6. In the **Value** field, paste your public SSH key.

7. Click **Create**.

The SSH key secret object is created. You can see it listed under **Workloads → Secrets**.

Once you create the **KataConfig** CR, you can run OpenShift sandboxed containers using peer pods on Azure.
3.2.4. Creating the KataConfig custom resource in the web console

You must create one KataConfig custom resource (CR) to enable installing kata-remote-cc as a RuntimeClass on your cluster nodes.

**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 Red Hat OpenShift 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 installed Red Hat OpenShift 4.12 on your cluster.
- You have access to the cluster as a user with the cluster-admin role.
- You have installed the OpenShift sandboxed containers Operator.

**NOTE**

Kata for peer pods is installed on all worker nodes by default. If you want to install kata-remote-cc as a RuntimeClass only on specific nodes, you can add labels to those nodes, then define the label in the KataConfig CR when you create it.

**Procedure**

1. From the Administrator perspective in the web console, navigate to Operators → Installed Operators.
2. Select the OpenShift sandboxed containers Operator from the list of operators.
3. In the KataConfig tab, click Create KataConfig.
4. In the Create KataConfig page, enter the following details:
   - **Name**: Enter a name for the KataConfig resource. By default, the name is defined as example-kataconfig.
   - **Labels** (Optional): Enter any relevant, identifying attributes to the KataConfig resource. Each label represents a key-value pair.
   - **checkNodeEligibility** (Optional, not relevant for peer pods): Select this checkbox to use the Node Feature Discovery Operator (NFD) to detect node eligibility to run kata as a RuntimeClass. For more information, see "Checking whether cluster nodes are eligible to run OpenShift sandboxed containers".
• **enablePeerPods** (For peer pods): Select this checkbox to enable peer pods and use OpenShift sandboxed containers in a public cloud environment.

• **kataConfigPoolSelector**: By default, kata-remote-cc is installed as a **RuntimeClass** on all nodes. If you want to install kata-remote-cc as a **RuntimeClass** only on selected nodes, you must add a **matchExpression**:

  a. Expand the **kataConfigPoolSelector** area.

  b. In the **kataConfigPoolSelector**, expand **matchExpressions**. This is a list of label selector requirements.

  c. Click **Add matchExpressions**.

  d. In the **key** field, add the label key the selector applies to.

  e. In the **operator** field, add 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 running kata-remote-cc as a **RuntimeClass**. For more information, see "Collecting OpenShift sandboxed containers data".

5. Click **Create**.

The new **KataConfig** CR is created and begins to install kata-remote-cc as a **RuntimeClass** on the worker nodes. Wait for the installation to complete and the worker nodes to reboot before continuing to the next step.

**IMPORTANT**

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

**Verification**

1. In the **KataConfig** tab, select the new **KataConfig** CR.

2. In the **KataConfig** page, select the **YAML** tab.

3. Monitor the **installationStatus** field in the status.

   A message appears each time there is an update. Click **Reload** to view the updated **KataConfig** CR.

   Once the value of **Completed nodes** equals the number of worker or labeled nodes, the installation is complete. The status also contains a list of nodes where the installation is completed.

**3.2.5. Deploying a workload with peer pods in a sandboxed container using the web console**
OpenShift sandboxed containers installs Kata as a secondary, optional runtime on your cluster, and not as the primary runtime.

To deploy a pod-templated workload using peer pods in a sandboxed container, you must manually add `kata-remote-cc` as the `runtimeClassName` to the workload YAML file.

**Prerequisites**

- You have installed Red Hat OpenShift 4.12 on your cluster.
- You have access to the cluster as a user with the `cluster-admin` role.
- You have installed the OpenShift sandboxed containers Operator.
- You have created a secret object and peer-pod config map unique to your cloud provider.
- You have created a `KataConfig` custom resource (CR).

**Procedure**

1. From the Administrator perspective in the web console, expand **Workloads** and select the type of workload you want to create.

2. In the workload page, click to create the workload.

3. In the YAML file for the workload, in the `spec` field where the container is listed, add `runtimeClassName: kata-remote-cc`.

   **Example for Pod object**

   ```yaml
   apiVersion: v1
   kind: Pod
   metadata:
     name: example
   labels:
     app: httpd
   namespace: openshift-sandboxed-containers-operator
   spec:
     runtimeClassName: kata-remote-cc
     containers:
     - name: httpd
       image: 'image-registry.openshift-image-registry.svc:5000/openshift/httpd:latest'
       ports:
         - name: null
         - containerPort: 8080
   ```

   **Example for Deployment object**

   ```yaml
   apiVersion: apps/v1
   kind: Deployment
   metadata:
     name: example
   namespace: openshift-sandboxed-containers-operator
   spec:
     selector:
       matchLabels:
   ```
Red Hat OpenShift creates the workload and begins scheduling it.

### 3.3. DEPLOYING OPENSHIFT SANBOXED CONTAINERS WORKLOADS USING PEER PODS WITH THE CLI

You can deploy OpenShift sandboxed containers workloads using the CLI. First, you must install the OpenShift sandboxed containers Operator, then create the `KataConfig` custom resource. Once you are ready to deploy a workload in a sandboxed container, you must add `kata-remote-cc` as the `runtimeClassName` to the workload YAML file.

#### 3.3.1. Installing the OpenShift sandboxed containers Operator using the CLI

You can install the OpenShift sandboxed containers Operator using the Red Hat OpenShift CLI.

**Prerequisites**

- You have Red Hat OpenShift 4.12 installed on your cluster.
- You have installed the OpenShift CLI (`oc`).
- You have access to the cluster as a user with the `cluster-admin` role.
- You have subscribed to the OpenShift sandboxed containers catalog.

**NOTE**

Subscribing to the OpenShift sandboxed containers catalog provides `openshift-sandboxed-containers-operator` namespace access to the OpenShift sandboxed containers Operator.

**Procedure**

1. Create the **Namespace** object for the OpenShift sandboxed containers Operator.
   a. Create a **Namespace** object YAML file that contains the following manifest:

   ```yaml
   apiVersion: v1
   ```
Create the **Namespace** object:

```bash
cat Namespace.yaml
```

2. Create the **OperatorGroup** object for the OpenShift sandboxed containers Operator.

   a. Create an **OperatorGroup** object YAML file that contains the following manifest:

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

   b. Create the **OperatorGroup** object:

```bash
cat OperatorGroup.yaml
```

3. Create the **Subscription** object to subscribe the **Namespace** to the OpenShift sandboxed containers Operator.

   a. Create a **Subscription** object YAML file that contains the following manifest:

```
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.v1.4.0
```

   b. Create the **Subscription** object:

```bash
cat Subscription.yaml
```

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

**NOTE**

All the object file names listed above are suggestions. You can create the object YAML files using other names.
Verification

- Ensure that the Operator is correctly installed:

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

Example output

<table>
<thead>
<tr>
<th>NAME</th>
<th>DISPLAY</th>
<th>VERSION</th>
<th>REPLACES</th>
<th>PHASE</th>
</tr>
</thead>
<tbody>
<tr>
<td>openshift-sandboxed-containers</td>
<td>openshift-sandboxed-containers-operator</td>
<td>1.4.0</td>
<td>1.3.3</td>
<td>Succeeded</td>
</tr>
</tbody>
</table>

Additional resources

- Installing from OperatorHub using the CLI

3.3.2. Setting up peer pods for AWS using the CLI

To set up peer pods for use on AWS, you must create a secret object, an AWS image VM (AMI), and a peer-pod ConfigMap.

After setting up peer pods for AWS, you must still create the KataConfig custom resource (CR) to deploy OpenShift sandboxed containers using peer pods.

Prerequisites

- You have installed Red Hat OpenShift 4.12 on your cluster.
- You have installed the OpenShift CLI (`oc`).
- You have access to the cluster as a user with the `cluster-admin` role.
- You have installed the OpenShift sandboxed containers Operator.

3.3.2.1. Creating a secret object for AWS using the CLI

Set your AWS access keys and configure your network in a secret object. The secret object is used to create the pod VM image and used by peer pods.

When creating a secret object for AWS, you must set specific environment values. You can retrieve some of these values before creating the secret object. However, you must have the following values prepared:

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`

Procedure

1. Collect the necessary parameter values for the secret object. Make sure to write down each value.
   a. Retrieve the instance ID:
$ INSTANCE_ID=$(oc get nodes -l 'node-role.kubernetes.io/worker' -o jsonpath='{.items[0].spec.providerID}' | sed 's#[^ ]*/##g')

This value is not needed for the secret object itself, but is used to retrieve other values for the secret object.

b. Retrieve the AWS region:

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

c. Retrieve 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 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 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 YAML file with the following manifest:

apiVersion: v1
kind: Secret
metadata:
  name: peer-pods-secret
  namespace: openshift-sandboxed-containers-operator
type: Opaque
stringData:
  AWS_ACCESS_KEY_ID: "<enter value>" ①
  AWS_SECRET_ACCESS_KEY: "<enter value>" ②
  AWS_REGION: "<enter value>" ③
  AWS_SUBNET_ID: "<enter value>" ④
  AWS_VPC_ID: "<enter value>" ⑤
  AWS_SG_IDS: "<enter value>" ⑥

① Enter the AWS_ACCESS_KEY_ID value you prepared before you began.
② Enter the AWS_SECRET_ACCESS_KEY value you prepared before you began.
③ Enter the AWS_REGION value you retrieved.
Enter the AWS_SUBNET_ID value you retrieved.

Enter the AWS_VPC_ID value you retrieved.

Enter the AWS_SG_IDS value you retrieved.

3. Apply the secret object:

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

The secret object is applied.

3.3.2.2. Creating an AWS VM image (AMI) using the CLI

To run OpenShift sandboxed containers using peer pods on AWS, you must first create a RHEL AMI using your AWS account and resources.

Procedure

1. Run the following K8s job to create the image:


NOTE

This image is not managed by OpenShift sandboxed containers. You can delete it if necessary using the AWS web console or AWS CLI tool.

2. Wait for the job to complete:

$ oc wait --for=condition=complete job.batch/aws-image-creation --timeout=7m -n openshift-sandboxed-containers-operator

Once the image is created, you must set the image using a peer-pod ConfigMap.

3.3.2.3. Creating a peer-pod ConfigMap for AWS using the CLI

When creating a ConfigMap for AWS, you must set the AMI ID. You can retrieve this value before you create the ConfigMap.

Procedure

1. Retrieve the AMI ID. Make sure to save the value for later.

   $ PODVM_AMI_ID=$(aws ec2 describe-images --query "Images[].[ImageId]" --filters "Name=name,Values=peer-pod-ami" --region ${AWS_REGION} --output text) && echo "PODVM_AMI_ID: "$PODVM_AMI_ID"

2. Create a YAML file with the following manifest:

   apiVersion: v1
Enter the AMI ID value you retrieved.

3. Deploy the ConfigMap:

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

The ConfigMap is deployed. Once you create the **KataConfig** CR, you can run OpenShift sandboxed containers using peer pods on AWS.

### 3.3.3. Setting up peer pods for Azure using the CLI

To set up peer pods for use on Microsoft Azure, you must create a secret object, an Azure image VM, a peer-pod ConfigMap, and an SSH key secret object.

After setting up peer pods for Azure, you must still create the **KataConfig** custom resource (CR) to deploy OpenShift sandboxed containers using peer pods.

**Prerequisites**

- You have installed Red Hat OpenShift 4.12 on your cluster.
- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the OpenShift sandboxed containers Operator.

#### 3.3.3.1. Creating a secret object for Azure using the CLI

Set your Azure access keys and configure your network in a secret object. The secret object is used to create the pod VM image and used by peer pods.

When creating a secret object for Azure, you must set specific environment values. You can retrieve some of these values before creating the secret object. However, you must have the following values prepared:

- **AZURE_CLIENT_ID**
- **AZURE_CLIENT_SECRET**
- **AZURE_TENANT_ID**

**Procedure**
1. Collect additional parameter values for the secret object. Make sure to write down each value.
   
   a. Retrieve the subscription ID:
      
      ```bash
      $ AZURE_SUBSCRIPTION_ID=$(az account list --query "[?isDefault].id" -o tsv) && echo "AZURE_SUBSCRIPTION_ID: "$AZURE_SUBSCRIPTION_ID"
      ```

   b. Retrieve the resource group:
      
      ```bash
      $ AZURE_RESOURCE_GROUP=$(oc get infrastructure/cluster -o jsonpath='{.status.platformStatus.azure.resourceGroupName}') && echo "AZURE_RESOURCE_GROUP: "$AZURE_RESOURCE_GROUP"
      ```

   c. Retrieve the Azure region:
      
      ```bash
      $ AZURE_REGION=$(az group show --resource-group $(AZURE_RESOURCE_GROUP) --query "{Location:location}" --output tsv) && echo "AZURE_REGION: "$AZURE_REGION"
      ```

2. Create a YAML file with the following manifest:

   ```yaml
   apiVersion: v1
   kind: Secret
   metadata:
     name: peer-pods-secret
   namespace: openshift-sandboxed-containers-operator
   type: Opaque
   stringData:
     AZURE_CLIENT_ID: "<enter value>"  
     AZURE_CLIENT_SECRET: "<enter value>"  
     AZURE_TENANT_ID: "<enter value>"  
     AZURE_SUBSCRIPTION_ID: "<enter value>"  
     AZURE_REGION: "<enter value>"  
     AZURE_RESOURCE_GROUP: "<enter value>"  
   ```

   1. Enter the AZURE_CLIENT_ID value you prepared before you began.
   2. Enter the AZURE_CLIENT_SECRET value you prepared before you began.
   3. Enter the AZURE_TENANT_ID value you prepared before you began.
   4. Enter the AZURE_SUBSCRIPTION_ID value you retrieved.
   5. Enter the AZURE_REGION value you retrieved.
   6. Enter the AZURE_RESOURCE_GROUP value you retrieved.

3. Apply the secret object:

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

   The secret object is applied.
3.3.3.2. Creating an Azure VM image using the CLI

To run OpenShift sandboxed containers using peer pods on Azure, you must first create a RHEL image for Azure using your Azure account and resources.

**Procedure**

1. Run the following K8s job to create the image:

   ```bash
   
   NOTE
   This image is not managed by OpenShift sandboxed containers. You can delete it if necessary using the Azure web console or Azure CLI tool.
   
2. Wait for the job to complete:

   ```bash
   $ oc wait --for=condition=complete job.batch/azure-image-creation --timeout=7m -n openshift-sandboxed-containers-operator
   
   Once the image is created, you must set the image using a peer-pod ConfigMap.

3.3.3.3. Creating a peer-pod ConfigMap for Azure using the CLI

When creating a ConfigMap for Azure, you must set specific configuration values. You can retrieve these values before you create the ConfigMap.

**Procedure**

1. Collect the configuration values for the Azure peer-pod ConfigMap. Make sure to write down each value.
   a. Retrieve the Azure image ID:

      ```bash
      $ AZURE_IMAGE_ID=$(az image list --resource-group ${AZURE_RESOURCE_GROUP} --query "[].{Id: id} | [? contains(Id, 'peer-pod-vmimage')]") --output tsv) && echo "AZURE_IMAGE_ID: "$AZURE_IMAGE_ID"
      
      b. Retrieve the Azure VNet name:

      ```bash
      $ AZURE_VNET_NAME=$(az network vnet list --resource-group ${AZURE_RESOURCE_GROUP} --query "][.{Name:name}"
      --output tsv) && echo "AZURE_VNET_NAME: "$AZURE_VNET_NAME"
      
      This value is not needed for the ConfigMap, but is used to retrieve the Azure subnet ID.

   c. Retrieve the Azure subnet ID:

      ```bash
      $ 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"
      ```
d. Retrieve the Azure network security group (NSG) ID:

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

2. Create a YAML file with the following manifest:

```yaml
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_D8as_v5"
  AZURE_SUBNET_ID: "<enter value>"  
  AZURE_NSG_ID: "<enter value>"  
  AZURE_IMAGE_ID: "<enter value>"  
  PROXY_TIMEOUT: "5m"
  DISABLECVM: "true"
```

1. Enter the `AZURE_SUBNET_ID` value you retrieved.
2. Enter the `AZURE_NSG_ID` value you retrieved.
3. Enter the `AZURE_IMAGE_ID` value you retrieved.

3. Deploy the ConfigMap:

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

The ConfigMap is deployed.

3.3.3.4. Creating an SSH key secret object for Azure using the CLI

You must generate an SSH key and create an SSH key secret object to use peer pods with Azure.

**Procedure**

1. Generate the SSH key:

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

2. Create the SHH secret object:

   ```bash
   $ 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 is created and the SSH key secret object is created. Once you create the **KataConfig** CR, you can run OpenShift sandboxed containers using peer pods on Azure.
3.3.4. Creating the KataConfig custom resource using the CLI

You must create one KataConfig custom resource (CR) to install kata-remote-cc as a RuntimeClass on your nodes. Creating the KataConfig CR triggers the OpenShift sandboxed containers Operator to do the following:

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

**NOTE**

Kata for peer pods is installed on all worker nodes by default. If you want to install kata-remote-cc as a RuntimeClass only on specific nodes, you can add labels to those nodes, and then define the label in the KataConfig CR when you create it.

Prerequisites

- You have installed Red Hat OpenShift 4.12 on your cluster.
- You have installed the OpenShift CLI (oc).
- You have access to the cluster as a user with the cluster-admin role.
- You have installed the OpenShift sandboxed containers Operator.

**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 Red Hat OpenShift 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.

Procedure

1. Create a YAML file with the following manifest:

```yaml
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
```
2. (Optional) If you want to install kata-remote-cc as a RuntimeClass only on selected nodes, create a YAML file that includes the label in the manifest:

```yaml
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
spec:
  enablePeerPods: true
  logLevel: info
kataConfigPoolSelector:
  matchLabels:
    <label_key>: '<label_value>'
```

Labels in kataConfigPoolSelector only support single values; nodeSelector syntax is not supported.

3. Create the KataConfig resource:

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

The new KataConfig CR is created and begins to install kata-remote-cc as a RuntimeClass on the worker nodes. Wait for the kata-remote-cc installation to complete and the worker nodes to reboot before continuing to the next step.

**IMPORTANT**

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

**Verification**

- Monitor the installation progress:

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

Once the value of Is In Progress appears as false, the installation is complete.

**Additional resources**

- Understanding how to update labels on nodes

**3.3.5. Deploying a workload with peer pods in a sandboxed container using the CLI**

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

To deploy a pod-templated workload using peer pods in a sandboxed container, you must add kata-remote-cc as the runtimeClassName to the workload YAML file.
Prerequisites

- You have installed Red Hat OpenShift 4.12 on your cluster.
- You have installed the OpenShift CLI (oc).
- You have access to the cluster as a user with the `cluster-admin` role.
- You have installed the OpenShift sandboxed containers Operator.
- You have created a secret object and peer-pod config map unique to your cloud provider.
- You have created a `KataConfig` custom resource (CR).

Procedure

- Add `runtimeClassName: kata-remote-cc` to any pod-templated object:
  - **Pod** objects
  - **ReplicaSet** objects
  - **ReplicationController** objects
  - **StatefulSet** objects
  - **Deployment** objects
  - **DeploymentConfig** objects

Example for Pod object

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mypod
spec:
runtimeClassName: kata-remote-cc
```

Example for Deployment object

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mypod
labels:
  app: mypod
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mypod
  template:
    metadata:
      labels:
        app: mypod
```
Red Hat OpenShift creates the workload and begins scheduling it.

**Verification**

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

### 3.4. ADDITIONAL RESOURCES

- The OpenShift sandboxed containers Operator is supported in a restricted network environment. For more information, see [Using Operator Lifecycle Manager on restricted networks](#).

- When using a disconnected cluster on a restricted network, you must configure proxy support in [Operator Lifecycle Manager](#) to access the OperatorHub. Using a proxy allows the cluster to fetch the OpenShift sandboxed containers Operator.
CHAPTER 4. MONITORING OPENSHIFT SANDBOXED CONTAINERS

You can use the Red Hat OpenShift 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 web console, and administrators can also access and query raw metrics through Prometheus.

4.1. ABOUT OPENSHIFT SANDBOXED CONTAINERS 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 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 OS metrics
Kata guest OS metrics display data from the guest OS 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.

4.2. VIEWING METRICS FOR OPENSHIFT SANDBOXED CONTAINERS

You can access the metrics for OpenShift sandboxed containers in the Metrics page in the web console.

Prerequisites
- You have Red Hat OpenShift 4.12 installed.
- You have OpenShift sandboxed containers installed.
- You have access to the cluster as a user with the cluster-admin role or with view permissions for all projects.

Procedure
1. From the Administrator perspective in the 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 will display a list with all of the available kata metrics.

The metrics from your query are visualized on the page.

Additional resources

- For more information about creating PromQL queries to view metrics, see Querying metrics.

4.3. VIEWING THE OPENSHIFT SANDBOXED CONTAINERS DASHBOARD

You can access the OpenShift sandboxed containers dashboard in the Dashboards page in the web console.

Prerequisites

- You have Red Hat OpenShift 4.12 installed.
- You have OpenShift sandboxed containers installed.
- You have access to the cluster as a user with the cluster-admin role or with view permissions for all projects.

Procedure

1. From the Administrator perspective in the web console, navigate to Observe → Dashboards.

2. From the Dashboard drop-down list, select the Sandboxed Containers dashboard.

3. Optional: Select a time range for the graphs in the Time Range list.
   - Select a pre-defined time period.
   - Set a custom time range by selecting Custom time range in the Time Range list.
     a. Define the date and time range for the data you want to view.
     b. Click Save to save the custom time range.

4. Optional: Select a Refresh Interval

The dashboard appears on the page with the following metrics from the Kata guest OS category:

**Number of running VMs**
Displays the total number of sandboxed containers running on your cluster.

**CPU Usage (per VM)**
Displays the CPU usage for each individual sandboxed container.

**Memory Usage (per VM)**
Displays the memory usage for each individual sandboxed container.

Hover over each of the graphs within a dashboard to display detailed information about specific items.
4.4. ADDITIONAL RESOURCES

- For more information about gathering data for support, see Gathering data about your cluster.
CHAPTER 5. UNINSTALLING OPENSHIFT SANDBOXED CONTAINERS

You can uninstall OpenShift sandboxed containers by using either the Red Hat OpenShift web console or OpenShift CLI (oc). Both procedures are explained below.

5.1. UNINSTALLING OPENSHIFT SANDBOXED CONTAINERS USING THE WEB CONSOLE

Use the Red Hat OpenShift web console to delete the relevant OpenShift sandboxed containers pods, resources, and namespace.

5.1.1. Deleting OpenShift sandboxed containers pods using the web console

To uninstall OpenShift sandboxed containers, you must first delete all running pods that use kata as the runtimeClass.

Prerequisites

- You have Red Hat OpenShift 4.12 installed on your cluster.
- You have access to the cluster as a user with the cluster-admin role.
- You have a list of the pods that use kata as the runtimeClass.

Procedure

1. From the Administrator perspective, navigate to Workloads → Pods.
2. Search for the pod that you want to delete using the Search by name field.
3. Click the pod name to open it.
4. On the Details page, check that kata is displayed for Runtime class.
5. Click the Actions menu and select Delete Pod.
6. Click Delete in the confirmation window.

Additional resources

You can retrieve a list of running pods that use kata as the runtimeClass from the OpenShift CLI. For details, see Deleting OpenShift sandboxed containers pods.

5.1.2. Deleting the KataConfig custom resource using the web console

 Deleting the KataConfig custom resource (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 Red Hat OpenShift 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 Red Hat OpenShift 4.12 installed on your cluster.
- You have access to the cluster as a user with the `cluster-admin` role.
- You have no running pods that use kata as the `runtimeClass`.

**Procedure**

1. From the Administrator perspective, 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.

### 5.1.3. Deleting the OpenShift sandboxed containers Operator using the web console

Deleting the OpenShift sandboxed containers Operator removes the catalog subscription, Operator group, and cluster service version (CSV) for that Operator.

**Prerequisites**

- You have Red Hat OpenShift 4.12 installed on your cluster.
- You have access to the cluster as a user with the `cluster-admin` role.

**Procedure**

1. From the Administrator perspective, navigate to Operators → Installed Operators.
2. Search for the OpenShift sandboxed containers Operator using the Search by name field.

3. Click the Options menu for the Operator and select Uninstall Operator.

4. Click Uninstall in the confirmation window.

5.1.4. Deleting the OpenShift sandboxed containers namespace using the web console

After you run the preceding commands, your cluster is restored to the state that it was prior to the installation process. You can now revoke namespace access to the Operator by deleting the openshift-sandboxed-containers-operator namespace.

Prerequisites

- You have Red Hat OpenShift 4.12 installed on your cluster.
- You have access to the cluster as a user with the cluster-admin role.

Procedure

1. From the Administrator perspective, navigate to Administration → Namespaces.

2. Search for the openshift-sandboxed-containers-operator namespace using the Search by name field.

3. 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.

4. In the Delete Namespace pane, enter openshift-sandboxed-containers-operator and click Delete.

5. Click Delete.

5.1.5. Deleting the KataConfig custom resource definition using the web console

The KataConfig custom resource definition (CRD) lets you define the KataConfig CR. To complete the uninstall process, delete the KataConfig CRD from your cluster.

Prerequisites

- You have Red Hat OpenShift 4.12 installed on your cluster.
- You have access to the cluster as a user with the cluster-admin role.
- You have removed the KataConfig CR from your cluster.
You have removed the OpenShift sandboxed containers Operator from your cluster.

Procedure

1. From the Administrator perspective, navigate to Administration → CustomResourceDefinitions.

2. Search for KataConfig using the Search by name field.

3. Click the Options menu for the KataConfig CRD, and then 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.

5.2. UNINSTALLING OPENSHIFT SANDBOXED CONTAINERS USING THE CLI

You can uninstall OpenShift sandboxed containers by using the Red Hat OpenShift command-line interface (CLI). Follow the steps below in the order that they are presented.

5.2.1. Deleting OpenShift sandboxed containers pods using the CLI

To uninstall OpenShift sandboxed containers, you must first delete all running pods that use kata as the runtimeClass.

Prerequisites

- You have installed the OpenShift CLI (oc).
- You have the command-line JSON processor (jq) installed.

Procedure

1. Search for pods that use kata as the runtimeClass by running the following command:

   ```
   $ oc get pods -A -o json | jq -r '.items[] | select(.spec.runtimeClassName == "kata").metadata.name'
   ```

2. To delete each pod, run the following command:

   ```
   $ oc delete pod <pod-name>
   ```

5.2.2. Deleting the KataConfig custom resource using the CLI

Remove and uninstall the kata runtime and all its related resources, such as CRI-O config and RuntimeClass, from your cluster.

Prerequisites
You have Red Hat OpenShift 4.12 installed on your cluster.

You have installed the OpenShift CLI (oc).

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

**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 Red Hat OpenShift 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.

**Procedure**

1. Delete the KataConfig custom resource by running the following command:

   ```sh
   $ oc delete kataconfig <KataConfig_CR_Name>
   ```

   The OpenShift sandboxed containers Operator removes all resources that were initially created to enable the runtime on your cluster.

   **IMPORTANT**

   During deletion, the CLI stops responding until all worker nodes reboot. Wait for the process to complete before performing the verification or continuing to the next procedure.

**Verification**

- To verify that the KataConfig custom resource is deleted, run the following command:

  ```sh
  $ oc get kataconfig <KataConfig_CR_Name>
  ```

**Example output**

```
No KataConfig instances exist
```

5.2.3. Deleting the OpenShift sandboxed containers Operator using the CLI

Remove the OpenShift sandboxed containers Operator from your cluster by deleting the Operator subscription, Operator group, cluster service version (CSV), and namespace.

**Prerequisites**
You have Red Hat OpenShift 4.10 installed on your cluster.

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. Fetch the cluster service version (CSV) name for OpenShift sandboxed containers from the subscription by running the following command:

   \[
   \text{CSV\_NAME}=$(\text{oc get csv -n openshift-sandboxed-containers-operator -o=custom-columns=:metadata.name})
   \]

2. Delete the OpenShift sandboxed containers Operator subscription from Operator Lifecycle Manager (OLM) by running the following command:

   \[
   $ \text{oc delete subscription sandboxed-containers-operator -n openshift-sandboxed-containers-operator}
   \]

3. Delete the CSV name for OpenShift sandboxed containers by running the following command:

   \[
   $ \text{oc delete csv $\{\text{CSV\_NAME}\} -n openshift-sandboxed-containers-operator}
   \]

4. Fetch the OpenShift sandboxed containers Operator group name by running the following command:

   \[
   $ \text{OG\_NAME}=$(\text{oc get operatorgroup -n openshift-sandboxed-containers-operator -o=jsonpath={..name}})
   \]

5. Delete the OpenShift sandboxed containers Operator group name by running the following command:

   \[
   $ \text{oc delete operatorgroup $\{\text{OG\_NAME}\} -n openshift-sandboxed-containers-operator}
   \]

6. Delete the OpenShift sandboxed containers namespace by running the following command:

   \[
   $ \text{oc delete namespace openshift-sandboxed-containers-operator}
   \]

5.2.4. Deleting the KataConfig custom resource definition using the CLI

The KataConfig custom resource definition (CRD) lets you define the KataConfig CR. Delete the KataConfig CRD from your cluster.

Prerequisites

- You have installed the OpenShift CLI (oc).
- You have access to the cluster as a user with the cluster-admin role.
- You have removed the KataConfig CR from your cluster.
• You have removed the OpenShift sandboxed containers Operator from your cluster.

**Procedure**

1. Delete the **KataConfig** CRD by running the following command:

   ```bash
   $ oc delete crd kataconfigs.kataconfiguration.openshift.io
   ```

**Verification**

• To verify that the **KataConfig** CRD is deleted, run the following command:

   ```bash
   $ oc get crd kataconfigs.kataconfiguration.openshift.io
   ```

**Example output**

```
Unknown CR KataConfig
```
CHAPTER 6. UPGRADING OPENSHIFT SANDBOXED CONTAINERS

The upgrade of the OpenShift sandboxed containers components consists of the following three steps:

- Upgrading Red Hat OpenShift to update the Kata runtime and its dependencies.
- Upgrading the OpenShift sandboxed containers Operator to update the Operator subscription.
- Manually patching the KataConfig custom resource (CR) to update the monitor pods.

You can upgrade Red Hat OpenShift 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.

IMPORTANT

If you are upgrading to Red Hat OpenShift 4.11 with OpenShift sandboxed containers 1.3, the recommended order is to first upgrade OpenShift sandboxed containers from 1.2 to 1.3, and then upgrade Red Hat OpenShift from 4.10 to 4.11.

6.1. UPGRADING THE OPENSHIFT SANDBOXED CONTAINERS 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 Kata 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 Red Hat OpenShift.

For more information about upgrading Red Hat OpenShift, see Updating Clusters.

6.2. UPGRADING THE OPENSHIFT SANDBOXED CONTAINERS 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 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.

6.3. UPGRADING THE OPENSHIFT SANDBOXED CONTAINERS MONITOR PODS

After upgrading OpenShift sandboxed containers, you need to update the monitor image in the KataConfig CR to upgrade the monitor pods. Otherwise, the monitor pods will continue running images from the previous version.

You can perform the update using the web console or the CLI.
6.3.1. Upgrading the monitor pods using the web console

The KataConfig YAML file in the Red Hat OpenShift contains the version number for the monitor image. Update the version number with the correct version.

**Prerequisites**

- You have Red Hat OpenShift 4.12 installed on your cluster.
- You have access to the cluster as a user with the cluster-admin role.

**Procedure**

1. From the Administrator perspective of Red Hat OpenShift, navigate to Operators → Installed Operators.
2. Select the OpenShift sandboxed containers Operator and go to the KataConfig tab.
3. Search for the KataConfig resource using the Search by name field. The default name for the KataConfig resource is example-kataconfig.
4. Select the KataConfig resource and go to the KataConfig tab.
5. Modify the version number for kataMonitorImage:

```
checkNodeEligibility: false
kataConfigPoolSelector: null
kataMonitorImage: 'registry.redhat.io/openshift-sandboxed-containers/osc-monitor-rhel8:1.3.0'
```

6. Click Save.

6.3.2. Upgrading the monitor pods using the CLI

You can manually patch the monitor image in the KataConfig CR to update the monitor pods.

**Prerequisites**

- You have Red Hat OpenShift 4.12 installed on your cluster.
- You have installed the OpenShift CLI (oc).
- You have access to the cluster as a user with the cluster-admin role.

**Procedure**

- In the Red Hat OpenShift CLI, run the following command:

  ```
  $ oc patch kataconfig <kataconfig_name> --type merge --patch 
  '{"spec":{"kataMonitorImage":"registry.redhat.io/openshift-sandboxed-containers/osc-monitor-rhel8:1.3.0"}}'
  ```

  where: `<kataconfig_name>` specifies the name of your Kata configuration file, such as example-kataconfig.
CHAPTER 7. COLLECTING OPENSIFT SANDBOXED CONTAINERS DATA

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.

7.1. COLLECTING OPENSIFT SANDBOXED CONTAINERS 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 Red Hat OpenShift cluster, including virtual machines and other data related to OpenShift sandboxed containers.

For prompt support, supply diagnostic information for both Red Hat OpenShift and OpenShift sandboxed containers.

7.1.1. About 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/container-native-virtualization/cnv-must-gather-rhel8:v4.12.0

- 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/container-native-virtualization/cnv-must-gather-rhel8:v4.12.0
```

To collect OpenShift sandboxed containers data with `must-gather`, you must specify the OpenShift sandboxed containers image:

```
--image=registry.redhat.io/openshift-sandboxed-containers/osc-must-gather-rhel8:1.4.0
```

### 7.2. ABOUT OPENSHIFT SANDBOXED CONTAINERS LOG DATA

When you collect log data about your cluster, the following features and objects are associated with OpenShift sandboxed containers:

- All namespaces and their child objects that belong to any OpenShift sandboxed containers resources
- All OpenShift sandboxed containers custom resource definitions (CRDs)

The following OpenShift sandboxed containers component logs are collected for each pod running with the `kata` runtime:

- Kata agent logs
- Kata runtime logs
- QEMU logs
- Audit logs
- CRI-O logs

### 7.3. ENABLING DEBUG LOGS FOR OPENSHIFT SANDBOXED CONTAINERS

As a cluster administrator, you can collect a more detailed level of logs for OpenShift sandboxed containers. You can also enhance logging by changing 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.
containers.

**Procedure**

1. Change the `logLevel` field in your existing KataConfig CR to `debug`:

   ```
   $ oc patch kataconfig <name_of_kataconfig_file> --type merge --patch '{"spec":{"logLevel":"debug"}}'
   ```

   **NOTE**

   When running this command, reference the name of your KataConfig CR. This is the name you used to create the CR when setting up OpenShift sandboxed containers.

**Verification**

1. Monitor the kata-oc machine config pool until the **UPDATED** field appears as **True**, meaning 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
   ```

2. Verify that the `log_level` was updated in CRI-O:
   
   a. Open an `oc debug` session to a node in the machine config pool and run `chroot /host`:

      ```
      $ oc debug node/<node_name>
      sh-4.4# chroot /host
      ```

   b. Verify the changes in the `crio.conf` file:

      ```
      sh-4.4# crio config | grep 'log_level'
      ```

   **Example output**

   ```
   log_level = "debug"
   ```

**7.3.1. Viewing debug logs for OpenShift sandboxed containers**

Cluster administrators can use the enhanced debug logs for OpenShift sandboxed containers 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="2023-03-11T11:57:28.60890737Z" 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:
  ```
  $ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t kata -g "reading guest console"
  ```

- To review the kata runtime logs, run:
  ```
  $ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t kata
  ```

- To review the virtiofsd logs, run:
  ```
  $ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t virtiofsd
  ```

- To review the QEMU logs, run:
7.4. ADDITIONAL RESOURCES

- For more information about gathering data for support, see Gathering data about your cluster.