OpenShift sandboxed containers 1.7

User guide

Deploying sandboxed containers in OpenShift Container Platform

Last Updated: 2024-09-25

OpenShift sandboxed containers 1.7 User guide

Deploying sandboxed containers in OpenShift Container Platform

Abstract

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

Table of Contents

PREFACE

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

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

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

CHAPTER 1. ABOUT OPENSHIFT SANDBOXED CONTAINERS

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

1.1. FEATURES

OpenShift sandboxed containers provides the following features:

Run privileged or untrusted workloads

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

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

Ensure isolation for sensitive workloads

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

Ensure kernel isolation for each workload

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

Share the same workload across tenants

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

Ensure proper isolation and sandboxing for testing software

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

Ensure default resource containment through VM boundaries

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

1.2. COMPATIBILITY WITH OPENSHIFT CONTAINER PLATFORM

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

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

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

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

NOTE

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

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

Table 1.1. Supported architectures

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

There are two ways to deploy Kata containers runtime:

- Bare metal
- Peer pods

Peer pods technology for the deployment of OpenShift sandboxed containers in public clouds was available as Developer Preview in OpenShift sandboxed containers 1.5 and OpenShift Container Platform 4.14.

With the release of OpenShift sandboxed containers 1.7, the Operator requires OpenShift Container Platform version 4.15 or later.

Table 1.2. Feature availability by OpenShift version

Feature	Deployment method	OpenShift Container Platform 4.15	OpenShift Container Platform 4.16
Confidential Containers	Bare metal		
	Peer pods	Technology Preview	Technology Preview [1]
GPU support [2]	Bare metal		
	Peer pods	Developer Preview	Developer Preview

1. Technology Preview of Confidential Containers has been available since OpenShift sandboxed containers 1.7.0.
2. GPU functionality is not available on IBM Z.

Table 1.3. Supported cloud platforms for OpenShift sandboxed containers

Platform	GPU	Confidential Containers
AWS Cloud Computing Services	Developer Preview	
Microsoft Azure Cloud Computing Services	Developer Preview	Technology Preview [1]

1. Technology Preview of Confidential Containers has been available since OpenShift sandboxed containers 1.7.0.

Additional resources

- Developer Preview Support Scope
- Technology Preview Features - Scope of Support

1.3. NODE ELIGIBILITY CHECKS

You can verify that your bare-metal cluster nodes support OpenShift sandboxed containers by running a node eligibility check. The most common reason for node ineligibility is lack of virtualization support. If you run sandboxed workloads on ineligible nodes, you will experience errors.

High-level workflow

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

Additional resources

- Installing the Node Feature Discovery Operator

1.4. COMMON TERMS

The following terms are used throughout the documentation.

Sandbox

A sandbox is an isolated environment where programs can run. In a sandbox, you can run untested or untrusted programs without risking harm to the host machine or the operating system.
In the context of OpenShift sandboxed containers, sandboxing is achieved by running workloads in a different kernel using virtualization, providing enhanced control over the interactions between multiple workloads that run on the same host.

Pod

A pod is a construct that is inherited from Kubernetes and OpenShift Container Platform. It represents resources where containers can be deployed. Containers run inside of pods, and pods are used to specify resources that can be shared between multiple containers.
In the context of OpenShift sandboxed containers, a pod is implemented as a virtual machine. Several containers can run in the same pod on the same virtual machine.

OpenShift sandboxed containers Operator

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

Kata Containers

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

KataConfig

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

Runtime class

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

Peer pod

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

IBM Secure Execution

IBM Secure Execution for Linux is an advanced security feature introduced with IBM z15® and LinuxONE III. This feature extends the protection provided by pervasive encryption. IBM Secure Execution safeguards data at rest, in transit, and in use. It enables secure deployment of workloads and ensures data protection throughout its lifecycle. For more information, see Introducing IBM Secure Execution for Linux.

Confidential Containers

Confidential Containers protects containers and data by verifying that your workload is running in a Trusted Execution Environment (TEE). You can deploy this feature to safeguard the privacy of big data analytics and machine learning inferences.
Trustee is a component of Confidential Containers. Trustee is an attestation service that verifies the trustworthiness of the location where you plan to run your workload or where you plan to send confidential information. Trustee includes components deployed on a trusted side and used to verify whether the remote workload is running in a Trusted Execution Environment (TEE). Trustee is flexible and can be deployed in several different configurations to support a wide variety of applications and hardware platforms.

Confidential compute attestation Operator

The Confidential compute attestation Operator manages the installation, lifecycle, and configuration of Confidential Containers.

1.5. OPENSHIFT SANDBOXED CONTAINERS OPERATOR

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

The OpenShift sandboxed containers Operator is packaged in the Operator bundle format as two container images:

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

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

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

Additional resources

- Adding extensions to RHCOS

1.6. ABOUT CONFIDENTIAL CONTAINERS

Confidential Containers provides a confidential computing environment to protect containers and data by leveraging Trusted Execution Environments (TEE).

IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

For more information, see Exploring the OpenShift confidential containers solution.

1.7. OPENSHIFT VIRTUALIZATION

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

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

1.8. STORAGE CONSIDERATIONS

1.8.1. Block volume support

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

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

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

You can provision raw block volumes for OpenShift sandboxed containers by specifying **volumeMode: Block** in the PV specification.

Block volume example

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

1. Set **volumeMode** to **Block** to indicate that this PV is a raw block volume.

2. Replace this value with the filepath to your **LocalVolume** resource **by-id**. PVs are created for these local disks when the provisioner is deployed successfully. You must also use this path to label the node that uses the block device when deploying OpenShift sandboxed containers.

1.9. FIPS COMPLIANCE

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

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

OpenShift sandboxed containers can be used on FIPS enabled clusters.

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

NOTE

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

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

IMPORTANT

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

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

CHAPTER 2. DEPLOYING ON BARE METAL

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

NOTE

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

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

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

Cluster requirements

- You have installed Red Hat OpenShift Container Platform 4.14 or later on the cluster where you are installing the OpenShift sandboxed containers Operator.
- Your cluster has at least one worker node.

2.1. OPENSHIFT SANDBOXED CONTAINERS RESOURCE REQUIREMENTS

You must ensure that your cluster has sufficient resources.

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

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

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

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

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

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

Default memory sizes

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

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

File buffers appear and are accounted for in multiple locations:

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

NOTE

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

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

Example

```
$ oc describe runtimeclass kata
```

Example output

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

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

NOTE

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

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

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

2.2. DEPLOYING OPENSHIFT SANDBOXED CONTAINERS BY USING THE WEB CONSOLE

You can deploy OpenShift sandboxed containers on bare metal by using the OpenShift Container Platform web console to perform the following tasks:

1. Install the OpenShift sandboxed containers Operator.
2. Optional: Install the Node Feature Discovery (NFD) Operator to configure node eligibility checks. For more information, see node eligibility checks and the NFD Operator documentation.
3. Create the **KataConfig** custom resource.
4. Configure the OpenShift sandboxed containers workload objects.

2.2.1. Installing the OpenShift sandboxed containers Operator

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

Prerequisites

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

Procedure

1. 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 Operator** tile and click **Install**.
4. On the **Install Operator** page, select **stable** from the list of available **Update Channel** options.
5. Verify that **Operator recommended Namespace** is selected for **Installed Namespace**. This installs the Operator in the mandatory **openshift-sandboxed-containers-operator** namespace. If this namespace does not yet exist, it is automatically created.

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

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

7. Click **Install**.

8. Navigate to **Operators** → **Installed Operators** to verify that the Operator is installed.

Additional resources

- Using Operator Lifecycle Manager on restricted networks .
- Configuring proxy support in Operator Lifecycle Manager for disconnected environments.

2.2.2. Creating the KataConfig custom resource

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

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

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

> **IMPORTANT**
>
> Creating the **KataConfig** CR automatically reboots the worker nodes. The reboot can take from 10 to more than 60 minutes. The following factors might increase the reboot time:
>
> - A larger OpenShift Container Platform deployment with a greater number of worker nodes.
> - Activation of the BIOS and Diagnostics utility.
> - Deployment on a hard disk drive rather than an SSD.
> - Deployment on physical nodes such as bare metal, rather than on virtual nodes.
> - A slow CPU and network.

Prerequisites

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

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators** → **Installed Operators**.

2. Select the OpenShift sandboxed containers Operator.

3. On the **KataConfig** tab, click **Create KataConfig**.

4. Enter the following details:

    - **Name**: Optional: The default name is **example-kataconfig**.

    - **Labels**: Optional: Enter any relevant, identifying attributes to the **KataConfig** resource. Each label represents a key-value pair.

    - **checkNodeEligibility**: Optional: Select to use the Node Feature Discovery Operator (NFD) to detect node eligibility.

    - **kataConfigPoolSelector**. Optional: To install **kata** on selected nodes, add a match expression for the labels on the selected nodes:

        a. Expand the **kataConfigPoolSelector** area.

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

        c. Click **Add matchExpressions**.

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

        e. In the **Operator** field, enter the key's relationship to the label values. Valid operators are **In**, **NotIn**, **Exists**, and **DoesNotExist**.

        f. Expand the **Values** area and then click **Add value**.

        g. In the **Value** field, enter **true** or **false** for **key** label value.

    - **logLevel**: Define the level of log data retrieved for nodes with the **kata** runtime class.

5. Click **Create**. The **KataConfig** CR is created and installs the **kata** runtime class on the worker nodes.
   Wait for the **kata** installation to complete and the worker nodes to reboot before verifying the installation.

Verification

1. On the **KataConfig** tab, click the **KataConfig** CR to view its details.

2. Click the **YAML** tab to view the **status** stanza.
   The **status** stanza contains the **conditions** and **kataNodes** keys. The value of **status.kataNodes** is an array of nodes, each of which lists nodes in a particular state of **kata** installation. A message appears each time there is an update.

3. Click **Reload** to refresh the YAML.
   When all workers in the **status.kataNodes** array display the values **installed** and **conditions.InProgress: False** with no specified reason, the **kata** is installed on the cluster.

Additional resources

- KataConfig status messages

2.2.3. Configuring workload objects

You must configure OpenShift sandboxed containers workload objects by setting **kata** as the runtime class for the following pod-templated objects:

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

IMPORTANT

Do not deploy workloads in an Operator namespace. Create a dedicated namespace for these resources.

Prerequisites

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

Procedure

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

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

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

Verification

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

2.3. DEPLOYING OPENSHIFT SANDBOXED CONTAINERS BY USING THE COMMAND LINE

You can deploy OpenShift sandboxed containers on bare metal by using the command line interface (CLI) to perform the following tasks:

1. Install the OpenShift sandboxed containers Operator.
2. After installing the Operator, you can configure the following options:
   - Configure a block storage device.
   - Install the Node Feature Discovery (NFD) Operator to configure node eligibility checks. For more information, see node eligibility checks and the NFD Operator documentation.
     - Create a **NodeFeatureDiscovery** custom resource.
3. Create the **KataConfig** custom resource.
4. Optional: Modify the pod overhead.
5. Configure the OpenShift sandboxed containers workload objects.

2.3.1. Installing the OpenShift sandboxed containers Operator

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

Prerequisites

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

Procedure

1. Create an **osc-namespace.yaml** manifest file:

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

2. Create the namespace by running the following command:

```
$ oc apply -f osc-namespace.yaml
```

3. Create an **osc-operatorgroup.yaml** manifest file:

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

```
spec:
  targetNamespaces:
  - openshift-sandboxed-containers-operator
```

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

   ```
   $ oc apply -f osc-operatorgroup.yaml
   ```

5. Create an **osc-subscription.yaml** manifest file:

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: 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.v1.7.0
   ```

6. Create the subscription by running the following command:

   ```
   $ oc apply -f osc-subscription.yaml
   ```

7. Verify that the Operator is correctly installed by running the following command:

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

   This command can take several minutes to complete.

8. Watch the process by running the following command:

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

   Example output

   ```
   NAME                     DISPLAY                          VERSION       REPLACES
   PHASE
   openshift-sandboxed-containers   openshift-sandboxed-containers-operator  1.7.0    1.6.0
   Succeeded
   ```

Additional resources

- Using Operator Lifecycle Manager on restricted networks .
- Configuring proxy support in Operator Lifecycle Manager for disconnected environments.

2.3.2. Optional configurations

You can configure the following options after you install the OpenShift sandboxed containers Operator.

2.3.2.1. Provisioning local block volumes

You can use local block volumes with OpenShift sandboxed containers. You must first provision the local block volumes by using the Local Storage Operator (LSO). Then you must enable the nodes with the local block volumes to run OpenShift sandboxed containers workloads.

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

Prerequisites

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

Procedure

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

   NOTE

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

   Example: Block

```
apiVersion: "local.storage.openshift.io/v1"
kind: "LocalVolume"
metadata:
  name: "local-disks"
  namespace: "openshift-local-storage" (1)
spec:
  nodeSelector: (2)
    nodeSelectorTerms:
    - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - ip-10-0-136-143
          - ip-10-0-140-255
          - ip-10-0-144-180
  storageClassDevices:
    - storageClassName: "local-sc" (3)
      forceWipeDevicesAndDestroyAllData: false (4)
    volumeMode: Block
    devicePaths: 5
      - /path/to/device 6
```

1. The namespace where the Local Storage Operator is installed.

2. Optional: A node selector containing a list of nodes where the local storage volumes are attached. This example uses the node hostnames, obtained from **oc get node**. If a value is not defined, then the Local Storage Operator will attempt to find matching disks on all available nodes.

3. The name of the storage class to use when creating persistent volume objects.

4. This setting defines whether or not to call **wipefs**, which removes partition table signatures (magic strings) making the disk ready to use for Local Storage Operator provisioning. No other data besides signatures is erased. The default is "false" (**wipefs** is not invoked). Setting **forceWipeDevicesAndDestroyAllData** to "true" can be useful in scenarios where previous data can remain on disks that need to be re-used. In these scenarios, setting this field to true eliminates the need for administrators to erase the disks manually.

5. The path containing a list of local storage devices to choose from. You must use this path when enabling a node with a local block device to run OpenShift sandboxed containers workloads.

6. Replace this value with the filepath to your **LocalVolume** resource **by-id**, such as **/dev/disk/by-id/wwn**. PVs are created for these local disks when the provisioner is deployed successfully.

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

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

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

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

Example output

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

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

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

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

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

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

4. Verify that the persistent volumes were created:

```
$ oc get pv
```

Example output

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

IMPORTANT

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

2.3.2.2. Enabling nodes to use a local block device

You can configure nodes with a local block device to run OpenShift sandboxed containers workloads at the paths specified in the defined volume resource.

Prerequisites

- You provisioned a block device using the Local Storage Operator (LSO).

Procedure

- Enable each node with a local block device to run OpenShift sandboxed containers workloads by running the following command:

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

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

Example output

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

2.3.2.3. Creating a NodeFeatureDiscovery custom resource

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

NOTE

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

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

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

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

Prerequisites

- You have installed the NFD Operator.

Procedure

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

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

2. Create the **NodeFeatureDiscovery** CR:

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

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

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

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

2. Create the **KataConfig** CR:

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

Verification

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

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

Example output

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

2.3.3. Creating the KataConfig custom resource

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

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

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

IMPORTANT

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

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

Prerequisites

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

Procedure

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

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: example-kataconfig
spec:
  checkNodeEligibility: false (1)
  logLevel: info
#  kataConfigPoolSelector:
#    matchLabels:
#      <label_key>: '<label_value>' (2)
```

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

(2) Optional: If you have applied node labels to install OpenShift sandboxed containers on specific nodes, specify the key and value.

2. Create the **KataConfig** CR by running the following command:

```
$ oc apply -f example-kataconfig.yaml
```

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

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

3. Monitor the installation progress by running the following command:

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

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

2.3.4. Modifying pod overhead

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

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

For example, if you use 300Mi of file buffer cache in the guest, both QEMU and **virtiofsd** appear to use

300Mi additional memory. However, the same memory is being used in all three cases. Therefore, the total memory usage is only 300Mi, mapped in three different places. This is correctly accounted for when reporting the memory utilization metrics.

NOTE

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

Procedure

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

```
$ oc describe runtimeclass kata
```

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

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

3. Apply the changes by running the following command:

```
$ oc apply -f runtimeclass.yaml
```

2.3.5. Configuring workload objects

You must configure OpenShift sandboxed containers workload objects by setting **kata** as the runtime class for the following pod-templated objects:

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

IMPORTANT

Do not deploy workloads in an Operator namespace. Create a dedicated namespace for these resources.

Prerequisites

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

Procedure

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

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

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

Verification

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

CHAPTER 3. DEPLOYING ON AWS

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

Cluster requirements

- You have installed Red Hat OpenShift Container Platform 4.14 or later on the cluster where you are installing the OpenShift sandboxed containers Operator.
- Your cluster has at least one worker node.

3.1. PEER POD RESOURCE REQUIREMENTS

You must ensure that your cluster has sufficient resources.

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

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

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

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

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

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

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

(1) The default limit is 10 VMs per node.

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

You can edit the limit per node based on the requirements for your environment after you install the OpenShift sandboxed containers Operator.

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

The mutating webhook modifies a Kubernetes pod as follows:

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

NOTE

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

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

3.2. DEPLOYING OPENSHIFT SANDBOXED CONTAINERS BY USING THE WEB CONSOLE

You can deploy OpenShift sandboxed containers on AWS by using the OpenShift Container Platform web console to perform the following tasks:

1. Install the OpenShift sandboxed containers Operator.
2. Enable ports 15150 and 9000 to allow internal communication with peer pods.
3. Create the peer pods secret.
4. Create the peer pods config map.
5. Create the **KataConfig** custom resource.
6. Configure the OpenShift sandboxed containers workload objects.

3.2.1. Installing the OpenShift sandboxed containers Operator

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

Prerequisites

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

Procedure

1. 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 Operator** tile and click **Install**.
4. On the **Install Operator** page, select **stable** from the list of available **Update Channel** options.
5. Verify that **Operator recommended Namespace** is selected for **Installed Namespace**. This installs the Operator in the mandatory **openshift-sandboxed-containers-operator** namespace. If this namespace does not yet exist, it is automatically created.

   NOTE

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

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

Additional resources

- Using Operator Lifecycle Manager on restricted networks .
- Configuring proxy support in Operator Lifecycle Manager for disconnected environments.

3.2.2. Enabling ports for AWS

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

Prerequisites

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

Procedure

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

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

2. Retrieve the AWS region:

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

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

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

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

```
$ for AWS_SG_ID in "${AWS_SG_IDS[@]}"; do \
  aws ec2 authorize-security-group-ingress --group-id $AWS_SG_ID --protocol tcp --port
15150 --source-group $AWS_SG_ID --region $AWS_REGION \
  aws ec2 authorize-security-group-ingress --group-id $AWS_SG_ID --protocol tcp --port
9000 --source-group $AWS_SG_ID --region $AWS_REGION \
done
```

The ports are now enabled.

3.2.3. Creating the peer pods secret

You must create the peer pods secret for OpenShift sandboxed containers.

The secret stores credentials for creating the pod virtual machine (VM) image and peer pod instances.

By default, the OpenShift sandboxed containers Operator creates the secret based on the credentials used to create the cluster. However, you can manually create a secret that uses different credentials.

Prerequisites

- You have the following values generated by using the AWS console:
  - **AWS_ACCESS_KEY_ID**
  - **AWS_SECRET_ACCESS_KEY**

Procedure

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

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

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

1 Specify the **AWS_ACCESS_KEY_ID** value.

2 Specify the **AWS_SECRET_ACCESS_KEY** value.

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

6. Navigate to **Workloads** → **Secrets** to verify the peer pods secret.

3.2.4. Creating the peer pods config map

You must create the peer pods config map for OpenShift sandboxed containers.

Prerequisites

- You have your Amazon Machine Image (AMI) ID if you are not using the default AMI ID based on your cluster credentials.

Procedure

1. Obtain the following values from your AWS instance:

   a. Retrieve and record the instance ID:

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

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

   b. Retrieve and record the AWS region:

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

   c. Retrieve and record the AWS subnet ID:

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

   d. Retrieve and record the AWS VPC ID:

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

e. Retrieve and record the AWS security group IDs:

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

2. In the OpenShift Container Platform web console, navigate to **Operators → Installed Operators**.

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

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

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

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

1 Defines the default instance type that is used when a type is not defined in the workload.

2 Lists all of the instance types you can specify when creating the pod. This allows you to define smaller instance types for workloads that need less memory and fewer CPUs or larger instance types for larger workloads.

3 Optional: By default, this value is populated when you run the **KataConfig** CR, using an AMI ID based on your cluster credentials. If you create your own AMI, specify the correct AMI ID.

4 Specify the **AWS_REGION** value you retrieved.

5 Specify the **AWS_SUBNET_ID** value you retrieved.

6 Specify the **AWS_VPC_ID** value you retrieved.

7 Specify the **AWS_SG_IDS** value you retrieved.

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

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

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

3.2.5. Creating the KataConfig custom resource

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

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

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

IMPORTANT

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

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

Prerequisites

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

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators** → **Installed Operators**.
2. Select the OpenShift sandboxed containers Operator.
3. On the **KataConfig** tab, click **Create KataConfig**.
4. Enter the following details:
   - **Name**: Optional: The default name is **example-kataconfig**.

- **Labels**: Optional: Enter any relevant, identifying attributes to the **KataConfig** resource. Each label represents a key-value pair.
- **enablePeerPods**: Select for public cloud, IBM Z®, and IBM® LinuxONE deployments.
- **kataConfigPoolSelector**. Optional: To install **kata-remote** on selected nodes, add a match expression for the labels on the selected nodes:
  a. Expand the **kataConfigPoolSelector** area.
  b. In the **kataConfigPoolSelector** area, expand **matchExpressions**. This is a list of label selector requirements.
  c. Click **Add matchExpressions**.
  d. In the **Key** field, enter the label key the selector applies to.
  e. In the **Operator** field, enter the key's relationship to the label values. Valid operators are **In**, **NotIn**, **Exists**, and **DoesNotExist**.
  f. Expand the **Values** area and then click **Add value**.
  g. In the **Value** field, enter **true** or **false** for **key** label value.
- **logLevel**: Define the level of log data retrieved for nodes with the **kata-remote** runtime class.

5. Click **Create**. The **KataConfig** CR is created and installs the **kata-remote** runtime class on the worker nodes.
   Wait for the **kata-remote** installation to complete and the worker nodes to reboot before verifying the installation.

Verification

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

Additional resources

- KataConfig status messages

Verifying the pod VM image

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

Procedure

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

Troubleshooting

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

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

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

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

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

3.2.6. Configuring workload objects

You must configure OpenShift sandboxed containers workload objects by setting **kata-remote** as the runtime class for the following pod-templated objects:

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

IMPORTANT

Do not deploy workloads in an Operator namespace. Create a dedicated namespace for these resources.

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

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

Prerequisites

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

Procedure

1. In the OpenShift Container Platform web console, navigate to **Workloads** → workload type, for example, **Pods**.

2. On the workload type page, click an object to view its details.

3. Click the **YAML** tab.

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

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

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

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

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

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

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

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

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

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

Verification

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

3.3. DEPLOYING OPENSHIFT SANDBOXED CONTAINERS BY USING THE COMMAND LINE

You can deploy OpenShift sandboxed containers on AWS by using the command line interface (CLI) to perform the following tasks:

1. Install the OpenShift sandboxed containers Operator.
2. Optional: Change the number of virtual machines running on each worker node.
3. Enable ports 15150 and 9000 to allow internal communication with peer pods.
4. Create the peer pods secret.
5. Create the peer pods config map.
6. Create the **KataConfig** custom resource.
7. Configure the OpenShift sandboxed containers workload objects.

3.3.1. Installing the OpenShift sandboxed containers Operator

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

Prerequisites

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

Procedure

1. Create an **osc-namespace.yaml** manifest file:

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

2. Create the namespace by running the following command:

```
$ oc apply -f osc-namespace.yaml
```

3. Create an **osc-operatorgroup.yaml** manifest file:

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

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

   ```
   $ oc apply -f osc-operatorgroup.yaml
   ```

5. Create an **osc-subscription.yaml** manifest file:

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: 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.v1.7.0
   ```

6. Create the subscription by running the following command:

   ```
   $ oc apply -f osc-subscription.yaml
   ```

7. Verify that the Operator is correctly installed by running the following command:

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

   This command can take several minutes to complete.

8. Watch the process by running the following command:

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

   Example output

   ```
   NAME                          DISPLAY                              VERSION        REPLACES
   PHASE
   openshift-sandboxed-containers   openshift-sandboxed-containers-operator  1.7.0    1.6.0
   Succeeded
   ```

Additional resources

- Using Operator Lifecycle Manager on restricted networks .
- Configuring proxy support in Operator Lifecycle Manager for disconnected environments.

3.3.2. Modifying the number of peer pod VMs per node

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

Procedure

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

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

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

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

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

3.3.3. Enabling ports for AWS

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

Prerequisites

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

Procedure

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

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

2. Retrieve the AWS region:

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

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

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

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

```
$ for AWS_SG_ID in "${AWS_SG_IDS[@]}"; do \
  aws ec2 authorize-security-group-ingress --group-id $AWS_SG_ID --protocol tcp --port
15150 --source-group $AWS_SG_ID --region $AWS_REGION \
  aws ec2 authorize-security-group-ingress --group-id $AWS_SG_ID --protocol tcp --port
9000 --source-group $AWS_SG_ID --region $AWS_REGION \
done
```

The ports are now enabled.

3.3.4. Creating the peer pods secret

You must create the peer pods secret for OpenShift sandboxed containers.

The secret stores credentials for creating the pod virtual machine (VM) image and peer pod instances.

By default, the OpenShift sandboxed containers Operator creates the secret based on the credentials used to create the cluster. However, you can manually create a secret that uses different credentials.

Prerequisites

- You have the following values generated by using the AWS console:
  - **AWS_ACCESS_KEY_ID**
  - **AWS_SECRET_ACCESS_KEY**

Procedure

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

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

   (1) Specify the **AWS_ACCESS_KEY_ID** value.

   (2) Specify the **AWS_SECRET_ACCESS_KEY** value.

2. Create the secret by running the following command:

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

3. Optional: To update an existing peer pods config map, restart the **peerpodconfig-ctrl-caa-daemon** daemon set by running the following command:

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

3.3.5. Creating the peer pods config map

You must create the peer pods config map for OpenShift sandboxed containers.

Prerequisites

- You have your Amazon Machine Image (AMI) ID if you are not using the default AMI ID based on your cluster credentials.

Procedure

1. Obtain the following values from your AWS instance:

   a. Retrieve and record the instance ID:

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

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

   b. Retrieve and record the AWS region:

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

   c. Retrieve and record the AWS subnet ID:

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

   d. Retrieve and record the AWS VPC ID:

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

   e. Retrieve and record the AWS security group IDs:

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

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

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: peer-pods-cm
     namespace: openshift-sandboxed-containers-operator
   data:
     CLOUD_PROVIDER: "aws"
     VXLAN_PORT: "9000"
     PODVM_INSTANCE_TYPE: "t3.medium" 1
     PODVM_INSTANCE_TYPES: "t2.small,t2.medium,t3.large" 2
   ```

```
PROXY_TIMEOUT: "5m"
PODVM_AMI_ID: "<podvm_ami_id>" 3
AWS_REGION: "<aws_region>" 4
AWS_SUBNET_ID: "<aws_subnet_id>" 5
AWS_VPC_ID: "<aws_vpc_id>" 6
AWS_SG_IDS: "<aws_sg_ids>" 7
DISABLECVM: "true"
```

1 Defines the default instance type that is used when a type is not defined in the workload.

2 Lists all of the instance types you can specify when creating the pod. This allows you to define smaller instance types for workloads that need less memory and fewer CPUs or larger instance types for larger workloads.

3 Optional: By default, this value is populated when you run the **KataConfig** CR, using an AMI ID based on your cluster credentials. If you create your own AMI, specify the correct AMI ID.

4 Specify the **AWS_REGION** value you retrieved.

5 Specify the **AWS_SUBNET_ID** value you retrieved.

6 Specify the **AWS_VPC_ID** value you retrieved.

7 Specify the **AWS_SG_IDS** value you retrieved.

3. Create the config map by running the following command:

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

4. Optional: To update an existing peer pods config map, restart the **peerpodconfig-ctrl-caa-daemon** daemon set by running the following command:

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

3.3.6. Creating the KataConfig custom resource

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

Creating the **KataConfig** CR triggers the OpenShift sandboxed containers Operator to do the following: * Create a **RuntimeClass** CR named **kata-remote** with a default configuration. This enables users to configure workloads to use **kata-remote** as the runtime by referencing the CR in the **RuntimeClassName** field. This CR also specifies the resource overhead for the runtime.

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

IMPORTANT

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

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

Prerequisites

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

Procedure

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

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

1 Optional: If you have applied node labels to install **kata-remote** on specific nodes, specify the key and value, for example, **osc: 'true'**.

2. Create the **KataConfig** CR by running the following command:

```
$ oc apply -f example-kataconfig.yaml
```

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

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

3. Monitor the installation progress by running the following command:

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

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

4. Verify the daemon set by running the following command:

```
$ oc get -n openshift-sandboxed-containers-operator ds/peerpodconfig-ctrl-caa-daemon
```

5. Verify the runtime classes by running the following command:

```
$ oc get runtimeclass
```

Example output

```
NAME          HANDLER        AGE
kata          kata           152m
kata-remote   kata-remote    152m
```

Verifying the pod VM image

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

Procedure

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

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

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

Troubleshooting

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

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

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

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

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

3.3.7. Configuring workload objects

You must configure OpenShift sandboxed containers workload objects by setting **kata-remote** as the runtime class for the following pod-templated objects:

- **Pod** objects
- **ReplicaSet** objects
- **ReplicationController** objects

- **StatefulSet** objects
- **Deployment** objects
- **DeploymentConfig** objects

IMPORTANT

Do not deploy workloads in an Operator namespace. Create a dedicated namespace for these resources.

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

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

Prerequisites

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

Procedure

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

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

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

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

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

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

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

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

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

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

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

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

Verification

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

CHAPTER 4. DEPLOYING ON AZURE

You can deploy OpenShift sandboxed containers and Confidential Containers on Microsoft Azure Cloud Computing Services.

Cluster requirements

- You have installed Red Hat OpenShift Container Platform 4.14 or later on the cluster where you are installing the OpenShift sandboxed containers Operator.
- Your cluster has at least one worker node.

4.1. PEER POD RESOURCE REQUIREMENTS

You must ensure that your cluster has sufficient resources.

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

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

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

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

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

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

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

(1) The default limit is 10 VMs per node.

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

You can edit the limit per node based on the requirements for your environment after you install the OpenShift sandboxed containers Operator.

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

The mutating webhook modifies a Kubernetes pod as follows:

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

NOTE

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

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

4.2. DEPLOYING OPENSHIFT SANDBOXED CONTAINERS BY USING THE WEB CONSOLE

You can deploy OpenShift sandboxed containers on Azure by using the OpenShift Container Platform web console to perform the following tasks:

1. Install the OpenShift sandboxed containers Operator.
2. Create the peer pods secret.
3. Create the peer pods config map.
4. Create the Azure secret.
5. Create the **KataConfig** custom resource.
6. Configure the OpenShift sandboxed containers workload objects.

4.2.1. Installing the OpenShift sandboxed containers Operator

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

Prerequisites

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

Procedure

1. 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 Operator** tile and click **Install**.
4. On the **Install Operator** page, select **stable** from the list of available **Update Channel** options.
5. Verify that **Operator recommended Namespace** is selected for **Installed Namespace**. This installs the Operator in the mandatory **openshift-sandboxed-containers-operator** namespace. If this namespace does not yet exist, it is automatically created.

   NOTE

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

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

Additional resources

- Using Operator Lifecycle Manager on restricted networks .
- Configuring proxy support in Operator Lifecycle Manager for disconnected environments.

4.2.2. Creating the peer pods secret

You must create the peer pods secret for OpenShift sandboxed containers.

The secret stores credentials for creating the pod virtual machine (VM) image and peer pod instances.

By default, the OpenShift sandboxed containers Operator creates the secret based on the credentials used to create the cluster. However, you can manually create a secret that uses different credentials.

Prerequisites

- You have installed and configured the Azure CLI tool.

Procedure

1. Retrieve the Azure subscription ID by running the following command:

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

2. Generate the RBAC content by running the following command:

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

Example output

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

3. Record the RBAC output to use in the **secret** object.

4. In the OpenShift Container Platform web console, navigate to **Operators** → **Installed Operators**.

5. Click the OpenShift sandboxed containers Operator tile.

6. Click the Import icon (**+**) on the top right corner.

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

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

1 Specify the **AZURE_CLIENT_ID** value.

2 Specify the **AZURE_CLIENT_SECRET** value.

3 Specify the **AZURE_TENANT_ID** value.

4 Specify the **AZURE_SUBSCRIPTION_ID** value.

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

9. Navigate to **Workloads** → **Secrets** to verify the peer pods secret.

4.2.3. Creating the peer pods config map

You must create the peer pods config map for OpenShift sandboxed containers.

Procedure

1. Obtain the following values from your Azure instance:

   a. Retrieve and record the Azure resource group:

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

   b. Retrieve and record the Azure VNet name:

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

   This value is used to retrieve the Azure subnet ID.

   c. Retrieve and record the Azure subnet ID:

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

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

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

   e. Retrieve and record the Azure region:

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

2. In the OpenShift Container Platform web console, navigate to **Operators** → **Installed Operators**.

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

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

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

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: peer-pods-cm
  namespace: openshift-sandboxed-containers-operator
```

```
data:
  CLOUD_PROVIDER: "azure"
  VXLAN_PORT: "9000"
  AZURE_INSTANCE_SIZE: "Standard_B2als_v2" (1)
  AZURE_INSTANCE_SIZES:
"Standard_B2als_v2,Standard_D2as_v5,Standard_D4as_v5,Standard_D2ads_v5" (2)
  AZURE_SUBNET_ID: "<azure_subnet_id>" (3)
  AZURE_NSG_ID: "<azure_nsg_id>" (4)
  PROXY_TIMEOUT: "5m"
  AZURE_IMAGE_ID: "<azure_image_id>" (5)
  AZURE_REGION: "<azure_region>" (6)
  AZURE_RESOURCE_GROUP: "<azure_resource_group>" (7)
  DISABLECVM: "true"
```

(1) This value is the default if an instance size is not defined in the workload.

(2) Lists all of the instance sizes you can specify when creating the pod. This allows you to define smaller instance sizes for workloads that need less memory and fewer CPUs or larger instance sizes for larger workloads.

(3) Specify the **AZURE_SUBNET_ID** value that you retrieved.

(4) Specify the **AZURE_NSG_ID** value that you retrieved.

(5) Optional: By default, this value is populated when you run the **KataConfig** CR, using an Azure image ID based on your cluster credentials. If you create your own Azure image, specify the correct image ID.

(6) Specify the **AZURE_REGION** value you retrieved.

(7) Specify the **AZURE_RESOURCE_GROUP** value you retrieved.

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

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

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

4.2.4. Creating the Azure secret

You must create the secret for Azure.

Procedure

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

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

3. In the OpenShift Container Platform web console, navigate to **Workloads** → **Secrets**.

4. On the **Secrets** page, verify that you are in the **openshift-sandboxed-containers-operator** project.

5. Click **Create** and select **Key/value secret**

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

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

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

9. Click **Create**.

10. Delete the SSH keys you created:

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

4.2.5. Creating the KataConfig custom resource

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

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

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

IMPORTANT

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

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

Prerequisites

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

Procedure

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

Verification

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

Additional resources

- KataConfig status messages

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

Procedure

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

Troubleshooting

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

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

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

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

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

4.2.6. Configuring workload objects

You must configure OpenShift sandboxed containers workload objects by setting **kata-remote** as the runtime class for the following pod-templated objects:

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

IMPORTANT

Do not deploy workloads in an Operator namespace. Create a dedicated namespace for these resources.

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

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

Prerequisites

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

Procedure

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

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

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

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

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

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

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

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

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

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

Verification

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

4.3. DEPLOYING OPENSHIFT SANDBOXED CONTAINERS BY USING THE COMMAND LINE

You can deploy OpenShift sandboxed containers on Azure by using the command line interface (CLI) to perform the following tasks:

1. Install the OpenShift sandboxed containers Operator.
2. Optional: Change the number of virtual machines running on each worker node.
3. Create the peer pods secret.
4. Create the peer pods config map.
5. Create the Azure secret.
6. Create the **KataConfig** custom resource.
7. Configure the OpenShift sandboxed containers workload objects.

4.3.1. Installing the OpenShift sandboxed containers Operator

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

Prerequisites

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

Procedure

1. Create an **osc-namespace.yaml** manifest file:

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

2. Create the namespace by running the following command:

```
$ oc apply -f osc-namespace.yaml
```

3. Create an **osc-operatorgroup.yaml** manifest file:

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

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

```
$ oc apply -f osc-operatorgroup.yaml
```

5. Create an **osc-subscription.yaml** manifest file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: 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.v1.7.0
```

6. Create the subscription by running the following command:

```
$ oc apply -f osc-subscription.yaml
```

7. Verify that the Operator is correctly installed by running the following command:

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

This command can take several minutes to complete.

8. Watch the process by running the following command:

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

Example output

```
NAME                    DISPLAY                          VERSION         REPLACES
PHASE
openshift-sandboxed-containers   openshift-sandboxed-containers-operator  1.7.0    1.6.0
Succeeded
```

Additional resources

- Using Operator Lifecycle Manager on restricted networks .
- Configuring proxy support in Operator Lifecycle Manager for disconnected environments.

4.3.2. Modifying the number of peer pod VMs per node

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

Procedure

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

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

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

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

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

4.3.3. Creating the peer pods secret

You must create the peer pods secret for OpenShift sandboxed containers.

The secret stores credentials for creating the pod virtual machine (VM) image and peer pod instances.

By default, the OpenShift sandboxed containers Operator creates the secret based on the credentials used to create the cluster. However, you can manually create a secret that uses different credentials.

Prerequisites

- You have installed and configured the Azure CLI tool.

Procedure

1. Retrieve the Azure subscription ID by running the following command:

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

2. Generate the RBAC content by running the following command:

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

Example output

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

3. Record the RBAC output to use in the **secret** object.

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

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

1 Specify the **AZURE_CLIENT_ID** value.

2 Specify the **AZURE_CLIENT_SECRET** value.

3 Specify the **AZURE_TENANT_ID** value.

4 Specify the **AZURE_SUBSCRIPTION_ID** value.

5. Create the secret by running the following command:

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

6. Optional: To update an existing peer pods config map, restart the **peerpodconfig-ctrl-caa-daemon** daemon set by running the following command:

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

4.3.4. Creating the peer pods config map

You must create the peer pods config map for OpenShift sandboxed containers.

Procedure

1. Obtain the following values from your Azure instance:

   a. Retrieve and record the Azure resource group:

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

b. Retrieve and record the Azure VNet name:

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

This value is used to retrieve the Azure subnet ID.

c. Retrieve and record the Azure subnet ID:

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

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

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

e. Retrieve and record the Azure region:

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

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

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

1 This value is the default if an instance size is not defined in the workload.

2 Lists all of the instance sizes you can specify when creating the pod. This allows you to define smaller instance sizes for workloads that need less memory and fewer CPUs or larger instance sizes for larger workloads.

3 Specify the **AZURE_SUBNET_ID** value that you retrieved.

4 Specify the **AZURE_NSG_ID** value that you retrieved.

5 Optional: By default, this value is populated when you run the **KataConfig** CR, using an Azure image ID based on your cluster credentials. If you create your own Azure image, specify the correct image ID.

6 Specify the **AZURE_REGION** value you retrieved.

7 Specify the **AZURE_RESOURCE_GROUP** value you retrieved.

3. Create the config map by running the following command:

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

4. Optional: To update an existing peer pods config map, restart the **peerpodconfig-ctrl-caa-daemon** daemon set by running the following command:

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

4.3.5. Creating the Azure secret

You must create the secret for Azure.

Procedure

1. Log in to your OpenShift Container Platform cluster.

2. Generate an SSH key pair by running the following command:

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

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

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

4. Delete the SSH keys you created:

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

4.3.6. Creating the KataConfig custom resource

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

Creating the **KataConfig** CR triggers the OpenShift sandboxed containers Operator to do the following: * Create a **RuntimeClass** CR named **kata-remote** with a default configuration. This enables users to configure workloads to use **kata-remote** as the runtime by referencing the CR in the

RuntimeClassName field. This CR also specifies the resource overhead for the runtime.

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

IMPORTANT

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

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

Prerequisites

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

Procedure

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

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

1 Optional: If you have applied node labels to install **kata-remote** on specific nodes, specify the key and value, for example, **osc: 'true'**.

2. Create the **KataConfig** CR by running the following command:

```
$ oc apply -f example-kataconfig.yaml
```

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

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

3. Monitor the installation progress by running the following command:

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

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

4. Verify the daemon set by running the following command:

```
$ oc get -n openshift-sandboxed-containers-operator ds/peerpodconfig-ctrl-caa-daemon
```

5. Verify the runtime classes by running the following command:

```
$ oc get runtimeclass
```

Example output

```
NAME          HANDLER        AGE
kata          kata           152m
kata-remote   kata-remote    152m
```

Verifying the pod VM image

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

Procedure

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

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

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

Troubleshooting

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

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

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

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

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

4.3.7. Configuring workload objects

You must configure OpenShift sandboxed containers workload objects by setting **kata-remote** as the runtime class for the following pod-templated objects:

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

IMPORTANT

Do not deploy workloads in an Operator namespace. Create a dedicated namespace for these resources.

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

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

Prerequisites

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

Procedure

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

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

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

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

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

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

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

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

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

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

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

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

Verification

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

4.4. DEPLOYING CONFIDENTIAL CONTAINERS ON AZURE

You can deploy Confidential Containers on Microsoft Azure Cloud Computing Services after you deploy OpenShift sandboxed containers.

IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Cluster requirements

- You have installed Red Hat OpenShift Container Platform 4.15 or later on the cluster where you are installing the Confidential compute attestation Operator.

You deploy Confidential Containers by performing the following steps:

1. Install the Confidential compute attestation Operator.
2. Create the route for Trustee.
3. Enable the Confidential Containers feature gate.
4. Update the peer pods config map.
5. Delete the **KataConfig** custom resource (CR).

6. Re-create the **KataConfig** CR.

7. Create the Trustee authentication secret.

8. Create the Trustee config map.

9. Configure attestation policies:

    a. Create reference values.

    b. Create secrets for attested clients.

    c. Create the resource access policy.

    d. Optional: Create an attestation policy that overrides the default policy.

    e. If your TEE is Intel Trust Domain Extensions, configure the Provisioning Certificate Caching Service.

10. Create the **KbsConfig** CR.

11. Verify the attestation process.

4.4.1. Installing the Confidential compute attestation Operator

You can install the Confidential compute attestation Operator on Azure by using the CLI.

Prerequisites

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

Procedure

1. Create a **trustee-namespace.yaml** manifest file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: trustee-operator-system
```

2. Create the **trustee-operator-system** namespace by running the following command:

```
$ oc apply -f trustee-namespace.yaml
```

3. Create a **trustee-operatorgroup.yaml** manifest file:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: trustee-operator-group
  namespace: trustee-operator-system
```

```
spec:
  targetNamespaces:
  - trustee-operator-system
```

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

```
$ oc apply -f trustee-operatorgroup.yaml
```

5. Create a **trustee-subscription.yaml** manifest file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: trustee-operator
  namespace: trustee-operator-system
spec:
  channel: stable
  installPlanApproval: Automatic
  name: trustee-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: trustee-operator.v0.1.0
```

6. Create the subscription by running the following command:

```
$ oc apply -f trustee-subscription.yaml
```

7. Verify that the Operator is correctly installed by running the following command:

```
$ oc get csv -n trustee-operator-system
```

This command can take several minutes to complete.

8. Watch the process by running the following command:

```
$ watch oc get csv -n trustee-operator-system
```

Example output

```
NAME                    DISPLAY                  PHASE
trustee-operator.v0.1.0   Trustee Operator  0.1.0       Succeeded
```

4.4.2. Creating the route for Trustee

You can create a secure route with edge TLS termination for Trustee. External ingress traffic reaches the router pods as HTTPS and passes on to the Trustee pods as HTTP.

Prerequisites

- You have enabled the Confidential Containers feature gate.
- You have installed the Confidential compute attestation Operator.

Procedure

1. Create an edge route by running the following command:

```
$ oc create route edge --service=kbs-service --port kbs-port \
  -n trustee-operator-system
```

NOTE

Note: Currently, only a route with a valid CA-signed certificate is supported. You cannot use a route with self-signed certificate.

2. Set the **TRUSTEE_HOST** variable by running the following command:

```
$ TRUSTEE_HOST=$(oc get route -n trustee-operator-system kbs-service \
  -o jsonpath={.spec.host})
```

3. Verify the route by running the following command:

```
$ echo $TRUSTEE_HOST
```

Example output

```
kbs-service-trustee-operator-system.apps.memvjias.eastus.aroapp.io
```

Record this value for the peer pods config map.

4.4.3. Enabling the Confidential Containers feature gate

You must enable the Confidential Containers feature gate.

Procedure

1. Create a **cc-feature-gate.yaml** manifest file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: osc-feature-gates
  namespace: openshift-sandboxed-containers-operator
data:
  confidential: "true"
```

2. Create the config map by running the following command:

```
$ oc apply -f cc-feature-gate.yaml
```

4.4.4. Updating the peer pods config map

You must update the peer pods config map for Confidential Containers.

Procedure

1. Obtain the following values from your Azure instance:

   a. Retrieve and record the Azure resource group:

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

   b. Retrieve and record the Azure VNet name:

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

   This value is used to retrieve the Azure subnet ID.

   c. Retrieve and record the Azure subnet ID:

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

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

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

   e. Retrieve and record the Azure region:

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

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

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: peer-pods-cm
  namespace: openshift-sandboxed-containers-operator
data:
  CLOUD_PROVIDER: "azure"
  VXLAN_PORT: "9000"
  AZURE_INSTANCE_SIZE: "Standard_DC2as_v5" 1
  AZURE_INSTANCE_SIZES:
"Standard_DC2as_v5,Standard_DC4as_v5,Standard_DC8as_v5,Standard_DC16as_v5" 2
  AZURE_SUBNET_ID: "<azure_subnet_id>" 3
  AZURE_NSG_ID: "<azure_nsg_id>" 4
  PROXY_TIMEOUT: "5m"
  AZURE_IMAGE_ID: "<azure_image_id>" 5
  AZURE_REGION: "<azure_region>" 6
```

```
AZURE_RESOURCE_GROUP: "<azure_resource_group>" 7
DISABLECVM: "false"
AA_KBC_PARAMS: "cc_kbc::https://${TRUSTEE_HOST}" 8
```

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

3. Create the config map by running the following command:

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

4. Restart the **peerpodconfig-ctrl-caa-daemon** daemon set by running the following command:

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

4.4.5. Deleting the KataConfig custom resource

You can delete the **KataConfig** custom resource (CR) by using the command line.

Deleting the **KataConfig** CR removes the runtime and its related resources from your cluster.

IMPORTANT

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

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

Prerequisites

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

Procedure

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

   ```
   $ oc delete kataconfig example-kataconfig
   ```

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

   IMPORTANT

   When you delete the **KataConfig** CR, the CLI stops responding until all worker nodes reboot. You must for the deletion process to complete before performing the verification.

2. Verify that the custom resource was deleted by running the following command:

   ```
   $ oc get kataconfig example-kataconfig
   ```

   Example output

   ```
   No example-kataconfig instances exist
   ```

4.4.6. Re-creating the KataConfig custom resource

You must re-create the **KataConfig** custom resource (CR) for Confidential Containers.

IMPORTANT

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

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

Prerequisites

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

Procedure

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

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: example-kataconfig
spec:
  enablePeerPods: true
  logLevel: info
#  kataConfigPoolSelector:
#    matchLabels:
#      <label_key>: '<label_value>' (1)
```

(1) Optional: If you have applied node labels to install **kata-remote** on specific nodes, specify the key and value, for example, **cc: 'true'**.

2. Create the **KataConfig** CR by running the following command:

```
$ oc apply -f example-kataconfig.yaml
```

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

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

3. Monitor the installation progress by running the following command:

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

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

4. Verify the daemon set by running the following command:

```
$ oc get -n openshift-sandboxed-containers-operator ds/peerpodconfig-ctrl-caa-daemon
```

5. Verify the runtime classes by running the following command:

```
$ oc get runtimeclass
```

Example output

```
NAME          HANDLER       AGE
kata          kata          152m
kata-remote   kata-remote   152m
```

4.4.7. Creating the Trustee authentication secret

You must create the authentication secret for Trustee.

Prerequisites

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

Procedure

1. Create a private key by running the following command:

```
$ openssl genpkey -algorithm ed25519 > privateKey
```

2. Create a public key by running the following command:

```
$ openssl pkey -in privateKey -pubout -out publicKey
```

3. Create a secret by running the following command:

```
$ oc create secret generic kbs-auth-public-key --from-file=publicKey -n trustee-operator-
system
```

4. Verify the secret by running the following command:

```
$ oc get secret -n trustee-operator-system
```

4.4.8. Creating the Trustee config map

You must create the config map to configure the Trustee server.

Prerequisites

- You have created a route for Trustee.

Procedure

1. Create a **kbs-config-cm.yaml** manifest file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kbs-config-cm
  namespace: trustee-operator-system
data:
  kbs-config.json: |
    {
      "insecure_http" : true,
      "sockets": ["0.0.0.0:8080"],
      "auth_public_key": "/etc/auth-secret/publicKey",
      "attestation_token_config": {
        "attestation_token_type": "CoCo"
      },
      "repository_config": {
        "type": "LocalFs",
      "dir_path": "/opt/confidential-containers/kbs/repository"
    },
    "as_config": {
      "work_dir": "/opt/confidential-containers/attestation-service",
      "policy_engine": "opa",
      "attestation_token_broker": "Simple",
        "attestation_token_config": {
        "duration_min": 5
        },
      "rvps_config": {
        "store_type": "LocalJson",
        "store_config": {
          "file_path": "/opt/confidential-containers/rvps/reference-values/reference-values.json"
        }
       }
    },
    "policy_engine_config": {
      "policy_path": "/opt/confidential-containers/opa/policy.rego"
    }
  }
```

2. Create the config map by running the following command:

```
$ oc apply -f kbs-config-cm.yaml
```

4.4.9. Configuring attestation policies

You can configure the following attestation policy settings:

Reference values

You can configure reference values for the Reference Value Provider Service (RVPS) by specifying the trusted digests of your hardware platform.
The client collects measurements from the running software, the Trusted Execution Environment (TEE) hardware and firmware and it submits a quote with the claims to the Attestation Server. These measurements must match the trusted digests registered to the Trustee. This process ensures that the confidential VM (CVM) is running the expected software stack and has not been tampered with.

Secrets for clients

You must create one or more secrets to share with attested clients.

Resource access policy

You must configure a policy for the Trustee policy engine to determine which resources to access.
Do not confuse the Trustee policy engine with the Attestation Service policy engine, which determines the validity of TEE evidence.

Attestation policy

Optional: You can overwrite the default attestation policy by creating your own attestation policy.

Provisioning Certificate Caching Service for TDX

If your TEE is Intel Trust Domain Extensions (TDX), you must configure the Provisioning Certificate Caching Service (PCCS). The PCCS retrieves Provisioning Certification Key (PCK) certificates and caches them in a local database.

IMPORTANT

Do not use the public Intel PCCS service. Use a local caching service on-premise or on the public cloud.

Procedure

1. Create an **rvps-configmap.yaml** manifest file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: rvps-reference-values
  namespace: trustee-operator-system
data:
  reference-values.json: |
    [ (1)
    ]
```

(1) Specify the trusted digests for your hardware platform if required. Otherwise, leave it empty.

2. Create the RVPS config map by running the following command:

```
$ oc apply -f rvps-configmap.yaml
```

3. Create one or more secrets to share with attested clients according to the following example:

```
$ oc create secret generic kbsres1 --from-literal key1=<res1val1> \
  --from-literal key2=<res1val2> -n trustee-operator-system
```

In this example, the **kbsres1** secret has two entries (**key1**, **key2**), which the Trustee clients retrieve. You can add more secrets according to your requirements.

4. Create a **resourcepolicy-configmap.yaml** manifest file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: resource-policy
  namespace: trustee-operator-system
data:
  policy.rego: | (1)
    package policy (2)
    default allow = false
    allow {
      input["tee"] != "sample"
    }
```

(1) The name of the resource policy, **policy.rego**, must match the resource policy defined in the Trustee config map.

2 The resource policy follows the Open Policy Agent specification. This example allows the retrieval of all resources when the TEE is not the sample attester.

5. Create the resource policy config map by running the following command:

```
$ oc apply -f resourcepolicy-configmap.yaml
```

6. Optional: Create an **attestation-policy.yaml** manifest file according to the following example:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: attestation-policy
  namespace: trustee-operator-system
data:
  default.rego: |
     package policy 1
     import future.keywords.every

     default allow = false

     allow {
        every k, v in input {
            judge_field(k, v)
        }
     }

     judge_field(input_key, input_value) {
        has_key(data.reference, input_key)
        reference_value := data.reference[input_key]
        match_value(reference_value, input_value)
     }

     judge_field(input_key, input_value) {
        not has_key(data.reference, input_key)
     }

     match_value(reference_value, input_value) {
        not is_array(reference_value)
        input_value == reference_value
     }

     match_value(reference_value, input_value) {
        is_array(reference_value)
        array_include(reference_value, input_value)
     }

     array_include(reference_value_array, input_value) {
        reference_value_array == []
     }

     array_include(reference_value_array, input_value) {
        reference_value_array != []
        some i
        reference_value_array[i] == input_value
```

```
      }

      has_key(m, k) {
        _ = m[k]
      }
```

1. The attestation policy follows the Open Policy Agent specification. In this example, the attestation policy compares the claims provided in the attestation report to the reference values registered in the RVPS database. The attestation process is successful only if all the values match.

7. Create the attestation policy config map by running the following command:

```
$ oc apply -f attestation-policy.yaml
```

8. If your TEE is Intel TDX, create a **tdx-config.yaml** manifest file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tdx-config
  namespace: trustee-operator-system
data:
  sgx_default_qcnl.conf: | \
      {
        "collateral_service": "https://api.trustedservices.intel.com/sgx/certification/v4/",
        "pccs_url": "<pccs_url>" 1
      }
```

1. Specify the PCCS URL, for example, **https://localhost:8081/sgx/certification/v4/**.

9. Create the TDX config map by running the following command:

```
$ oc apply -f tdx-config.yaml
```

4.4.10. Creating the KbsConfig custom resource

You must create the **KbsConfig** custom resource (CR) to launch Trustee.

Then, you check the Trustee pods and pod logs to verify the configuration.

Procedure

1. Create a **kbsconfig-cr.yaml** manifest file:

```
apiVersion: confidentialcontainers.org/v1alpha1
kind: KbsConfig
metadata:
  labels:
    app.kubernetes.io/name: kbsconfig
    app.kubernetes.io/instance: kbsconfig
    app.kubernetes.io/part-of: trustee-operator
    app.kubernetes.io/managed-by: kustomize
    app.kubernetes.io/created-by: trustee-operator
  name: kbsconfig
  namespace: trustee-operator-system
spec:
  kbsConfigMapName: kbs-config-cm
  kbsAuthSecretName: kbs-auth-public-key
  kbsDeploymentType: AllInOneDeployment
  kbsRvpsRefValuesConfigMapName: rvps-reference-values
  kbsSecretResources: ["kbsres1"]
  kbsResourcePolicyConfigMapName: resource-policy
```

2. Create the **KbsConfig** CR by running the following command:

```
$ oc apply -f kbsconfig-cr.yaml
```

Verification

1. Set the default project by running the following command:

```
$ oc project trustee-operator-system
```

2. Check the pods by running the following command:

```
$ oc get pods -n trustee-operator-system
```

Example output

```
NAME                                                   READY   STATUS    RESTARTS   AGE
trustee-deployment-8585f98449-9bbgl                    1/1     Running   0          22m
trustee-operator-controller-manager-5fbd44cd97-55dlh   2/2     Running   0          59m
```

3. Set the **POD_NAME** environmental variable by running the following command:

```
$ POD_NAME=$(oc get pods -l app=kbs -o jsonpath='{.items[0].metadata.name}' -n trustee-
operator-system)
```

4. Check the pod logs by running the following command:

```
$ oc logs -n trustee-operator-system $POD_NAME
```

Example output

```
[2024-05-30T13:44:24Z INFO  kbs] Using config file /etc/kbs-config/kbs-config.json
[2024-05-30T13:44:24Z WARN  attestation_service::rvps] No RVPS address provided and
will launch a built-in rvps
[2024-05-30T13:44:24Z INFO  attestation_service::token::simple] No Token Signer key in
config file, create an ephemeral key and without CA pubkey cert
[2024-05-30T13:44:24Z INFO  api_server] Starting HTTPS server at [0.0.0.0:8080]
[2024-05-30T13:44:24Z INFO  actix_server::builder] starting 12 workers
[2024-05-30T13:44:24Z INFO  actix_server::server] Tokio runtime found; starting in existing
Tokio runtime
```

4.4.11. Verifying the attestation process

You can verify the attestation process by creating a test pod and retrieving its secret.

IMPORTANT

This procedure is an example to verify that attestation is working. Do not write sensitive data to standard I/O because the data can be captured by using a memory dump. Only data written to memory is encrypted.

By default, an agent side policy embedded in the pod VM image disables the **exec** and **log** APIs for a Confidential Containers pod. This policy ensures that sensitive data is not written to standard I/O.

In a test scenario, you can override the restriction at runtime by adding a policy annotation to the pod. For Technology Preview, runtime policy annotations are not verified by remote attestation.

Prerequisites

- You have created a route if the Trustee server and the test pod are not running in the same cluster.

Procedure

1. Create a **verification-pod.yaml** manifest file:

```
apiVersion: v1
kind: Pod
metadata:
  name: ocp-cc-pod
  labels:
    app: ocp-cc-pod
  annotations:
    io.katacontainers.config.agent.policy:
cGFja2FnZSBhZ2VudF9wb2xpY3kKCmRlZmF1bHQgQWRkQVJQTmVpZ2hib3JzUmVxdWVz
dCA6PSB0cnVlCmRlZmF1bHQgQWRkU3dhcFJlcXVlc3QgOj0gdHJ1ZQpkZWZhdWx0IENsb3
NlU3RkaW5SZXF1ZXN0IDo9IHRydWUKZGVmYXVsdCBDb3B5RmlsZVJlcXVlc3QgOj0gdHJ1
ZQpkZWZhdWx0IENyZWF0ZUNvbnRhaW5lclJlcXVlc3QgOj0gdHJ1ZQpkZWZhdWx0IENyZW
F0ZVNhbmRib3hSZXF1ZXN0IDo9IHRydWUKZGVmYXVsdCBEZXN0cm95U2FuZGJveFJlcX
Vlc3QgOj0gdHJ1ZQpkZWZhdWx0IEV4ZWNQcm9jZXNzUmVxdWVzdCA6PSB0cnVlCmRlZm
F1bHQgR2V0TWV0cmljc1JlcXVlc3QgOj0gdHJ1ZQpkZWZhdWx0IEdldE9PTUV2ZW50UmVxd
WVzdCA6PSB0cnVlCmRlZmF1bHQgR3Vlc3REZXRhaWxzUmVxdWVzdCA6PSB0cnVlCmRlZ
mF1bHQgTGlzdEludGVyZmFjZXNSZXF1ZXN0IDo9IHRydWUKZGVmYXVsdCBMaXN0Um91
dGVzUmVxdWVzdCA6PSB0cnVlCmRlZmF1bHQgTWVtSG90cGx1Z0J5UHJvYmVSZXF1ZXN
0IDo9IHRydWUKZGVmYXVsdCBPbmxpbmVDUFVNZW1SZXF1ZXN0IDo9IHRydWUKZGVm
YXVsdCBQYXVzZUNvbnRhaW5lclJlcXVlc3QgOj0gdHJ1ZQpkZWZhdWx0IFB1bGxJbWFnZVJl
cXVlc3QgOj0gdHJ1ZQpkZWZhdWx0IFJlYWRTdHJlYW1SZXF1ZXN0IDo9IHRydWUKZGVmY
XVsdCBSZW1vdmVDb250YWluZXJSZXF1ZXN0IDo9IHRydWUKZGVmYXVsdCBSZW1vdmV
TdGFsZVZpcnRpb2ZzU2hhcmVNb3VudHNSZXF1ZXN0IDo9IHRydWUKZGVmYXVsdCBSZX
NlZWRSYW5kb21EZXZSZXF1ZXN0IDo9IHRydWUKZGVmYXVsdCBSZXN1bWVDb250YWlu
ZXJSZXF1ZXN0IDo9IHRydWUKZGVmYXVsdCBTZXRHdWVzdERhdGVUaW1lUmVxdWVzd
CA6PSB0cnVlCmRlZmF1bHQgU2V0UG9saWN5UmVxdWVzdCA6PSB0cnVlCmRlZmF1bHQ
gU2lnbmFsUHJvY2Vzc1JlcXVlc3QgOj0gdHJ1ZQpkZWZhdWx0IFN0YXJ0Q29udGFpbmVyUm
VxdWVzdCA6PSB0cnVlCmRlZmF1bHQgU3RhcnRUcmFjaW5nUmVxdWVzdCA6PSB0cnVlCm
RlZmF1bHQgU3RhdHNDb250YWluZXJSZXF1ZXN0IDo9IHRydWUKZGVmYXVsdCBTdG9w
VHJhY2luZ1JlcXVlc3QgOj0gdHJ1ZQpkZWZhdWx0IFR0eVdpblJlc2l6ZVJlcXVlc3QgOj0gdHJ1Z
```

```
QpkZWZhdWx0IFVwZGF0ZUNvbnRhaW5lclJlcXVlc3QgOj0gdHJ1ZQpkZWZhdWx0IFVwZGF
0ZUVwaGVtZXJhbE1vdW50c1JlcXVlc3QgOj0gdHJ1ZQpkZWZhdWx0IFVwZGF0ZUludGVyZ
mFjZVJlcXVlc3QgOj0gdHJ1ZQpkZWZhdWx0IFVwZGF0ZVJvdXRlc1JlcXVlc3QgOj0gdHJ1ZQ
pkZWZhdWx0IFdhaXRQcm9jZXNzUmVxdWVzdCA6PSB0cnVlCmRlZmF1bHQgV3JpdGVTdH
JlYW1SZXF1ZXN0IDo9IHRydWUK (1)
spec:
  runtimeClassName: kata-remote
  containers:
    - name: skr-openshift
      image: registry.access.redhat.com/ubi9/ubi:9.3
      command:
        - sleep
        - "36000"
      securityContext:
        privileged: false
        seccompProfile:
          type: RuntimeDefault
```

(1) This pod annotation overrides the policy that prevents sensitive data from being written to standard I/O.

2. Create the pod by running the following command:

```
$ oc create -f verification-pod.yaml
```

3. Connect to the Bash shell of the **ocp-cc-pod** by running the following command:

```
$ oc exec -it ocp-cc-pod -- bash
```

4. Fetch the pod secret by running the following command:

```
$ curl http://127.0.0.1:8006/cdh/resource/default/kbsres1/key1
```

Example output

```
res1val1
```

The Trustee server returns the secret only if the attestation is successful.

CHAPTER 5. DEPLOYING ON IBM Z AND IBM LINUXONE

You can deploy OpenShift sandboxed containers or Confidential Containers on IBM Z® and IBM® LinuxONE.

IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Cluster requirements

- You have installed Red Hat OpenShift Container Platform 4.14 or later on the cluster where you are installing the OpenShift sandboxed containers Operator.
- Your cluster has at least one worker node.

5.1. PEER POD RESOURCE REQUIREMENTS

You must ensure that your cluster has sufficient resources.

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

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

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

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

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

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

```
apiVersion: confidentialcontainers.org/v1alpha1
kind: PeerPodConfig
metadata:
  name: peerpodconfig-openshift
  namespace: openshift-sandboxed-containers-operator
spec:
```

```
cloudSecretName: peer-pods-secret
configMapName: peer-pods-cm
limit: "10" (1)
nodeSelector:
  node-role.kubernetes.io/kata-oc: ""
```

1. The default limit is 10 VMs per node.

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

You can edit the limit per node based on the requirements for your environment after you install the OpenShift sandboxed containers Operator.

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

The mutating webhook modifies a Kubernetes pod as follows:

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

NOTE

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

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

5.2. DEPLOYING OPENSHIFT SANDBOXED CONTAINERS ON IBM Z AND IBM LINUXONE

You can deploy OpenShift sandboxed containers on IBM Z® and IBM® LinuxONE by using the command line interface (CLI) to perform the following tasks:

1. Install the OpenShift sandboxed containers Operator.
2. Optional: Change the number of virtual machines running on each worker node.

3. Configure the libvirt volume.
4. Optional: Create a custom peer pod VM image.
5. Create the peer pods secret.
6. Create the peer pods config map.
7. Create the peer pod VM image config map.
8. Create the KVM host secret.
9. Create the **KataConfig** custom resource.
10. Configure the OpenShift sandboxed containers workload objects.

5.2.1. Installing the OpenShift sandboxed containers Operator

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

Prerequisites

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

Procedure

1. Create an **osc-namespace.yaml** manifest file:

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

2. Create the namespace by running the following command:

```
$ oc apply -f osc-namespace.yaml
```

3. Create an **osc-operatorgroup.yaml** manifest file:

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

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

```
$ oc apply -f osc-operatorgroup.yaml
```

5. Create an **osc-subscription.yaml** manifest file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: 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.v1.7.0
```

6. Create the subscription by running the following command:

```
$ oc apply -f osc-subscription.yaml
```

7. Verify that the Operator is correctly installed by running the following command:

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

   This command can take several minutes to complete.

8. Watch the process by running the following command:

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

   Example output

```
NAME                             DISPLAY                                  VERSION         REPLACES
PHASE
openshift-sandboxed-containers   openshift-sandboxed-containers-operator  1.7.0    1.6.0
Succeeded
```

Additional resources

- Using Operator Lifecycle Manager on restricted networks .
- Configuring proxy support in Operator Lifecycle Manager for disconnected environments.

5.2.2. Modifying the number of peer pod VMs per node

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

Procedure

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

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

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

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

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

5.2.3. Configuring the libvirt volume

You must configure the libvirt volume on your KVM host. Peer pods use the libvirt provider of the Cloud API Adaptor to create and manage virtual machines.

Prerequisites

- You have installed the OpenShift sandboxed containers Operator on your OpenShift Container Platform cluster by using the OpenShift Container Platform web console or the command line.
- You have administrator privileges for your KVM host.
- You have installed **podman** on your KVM host.
- You have installed **virt-customize** on your KVM host.

Procedure

1. Log in to the KVM host.
2. Set the name of the libvirt pool by running the following command:

```
$ export LIBVIRT_POOL=<libvirt_pool>
```

You need the **LIBVIRT_POOL** value to create the secret for the libvirt provider.

3. Set the name of the libvirt pool by running the following command:

```
$ export LIBVIRT_VOL_NAME=<libvirt_volume>
```

You need the **LIBVIRT_VOL_NAME** value to create the secret for the libvirt provider.

4. Set the path of the default storage pool location, by running the following command:

```
$ export LIBVIRT_POOL_DIRECTORY=<target_directory> 1
```

1 To ensure libvirt has read and write access permissions, use a subdirectory of the libvirt storage directory. The default is **/var/lib/libvirt/images/**.

5. Create a libvirt pool by running the following command:

```
$ virsh pool-define-as $LIBVIRT_POOL --type dir --target "$LIBVIRT_POOL_DIRECTORY"
```

6. Start the libvirt pool by running the following command:

```
$ virsh pool-start $LIBVIRT_POOL
```

7. Create a libvirt volume for the pool by running the following command:

```
$ virsh -c qemu:///system \
  vol-create-as --pool $LIBVIRT_POOL \
  --name $LIBVIRT_VOL_NAME \
  --capacity 20G \
  --allocation 2G \
  --prealloc-metadata \
  --format qcow2
```

5.2.4. Creating a custom peer pod VM image

You can create a custom peer pod virtual machine (VM) image instead of using the default Operator-built image.

You build an Open Container Initiative (OCI) container with the peer pod QCOW2 image. Later, you add the container registry URL and the image path to the peer pod VM image config map.

Procedure

1. Create a **Dockerfile.podvm-oci** file:

```
FROM scratch

ARG PODVM_IMAGE_SRC
ENV PODVM_IMAGE_PATH="/image/podvm.qcow2"

COPY $PODVM_IMAGE_SRC $PODVM_IMAGE_PATH
```

2. Build a container with the pod VM QCOW2 image by running the following command:

```
$ docker build -t podvm-libvirt \
  --build-arg PODVM_IMAGE_SRC=<podvm_image_source> \ (1)
  --build-arg PODVM_IMAGE_PATH=<podvm_image_path> \ (2)
  -f Dockerfile.podvm-oci .
```

(1) Specify the QCOW2 image source on the host.

(2) Optional: Specify the path of the QCOW2 image if you do not use the default, **/image/podvm.qcow2**.

5.2.5. Creating the peer pods secret

You must create the peer pods secret for OpenShift sandboxed containers.

The secret stores credentials for creating the pod virtual machine (VM) image and peer pod instances.

By default, the OpenShift sandboxed containers Operator creates the secret based on the credentials used to create the cluster. However, you can manually create a secret that uses different credentials.

Prerequisites

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

  NOTE

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

  ```
  $ virtint=$(bridge_line=$(virsh net-info default | grep Bridge);  echo
  "${bridge_line//Bridge:/}" | tr -d [:blank:])

  $ LIBVIRT_URI=$( ip -4 addr show $virtint | grep -oP '(?<=inet\s)\d+(\.\d+){3}')

  $ LIBVIRT_GATEWAY_URI="qemu+ssh://root@${LIBVIRT_URI}/system?
  no_verify=1"
  ```

- **REDHAT_OFFLINE_TOKEN**. You have generated this token to download the RHEL image at Red Hat API Tokens.

Procedure

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

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: peer-pods-secret
     namespace: openshift-sandboxed-containers-operator
   type: Opaque
   stringData:
     CLOUD_PROVIDER: "libvirt"
     LIBVIRT_URI: "<libvirt_gateway_uri>" 1
     LIBVIRT_POOL: "<libvirt_pool>" 2
     LIBVIRT_VOL_NAME: "<libvirt_volume>" 3
     REDHAT_OFFLINE_TOKEN: "<rh_offline_token>" 4
   ```

   1 Specify the libvirt URI.

   2 Specify the libvirt pool.

   3 Specify the libvirt volume name.

   4 Specify the Red Hat offline token, which is required for the Operator-built image.

2. Create the secret by running the following command:

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

3. Optional: To update an existing peer pods config map, restart the **peerpodconfig-ctrl-caa-daemon** daemon set by running the following command:

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

5.2.6. Creating the peer pods config map

You must create the peer pods config map for OpenShift sandboxed containers.

Procedure

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

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

2. Create the config map by running the following command:

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

3. Optional: To update an existing peer pods config map, restart the **peerpodconfig-ctrl-caa-daemon** daemon set by running the following command:

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

5.2.7. Creating the peer pod VM image config map

You must create the config map for the peer pod VM image.

Procedure

1. Create a **libvirt-podvm-image-cm.yaml** manifest according to the following example:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: libvirt-podvm-image-cm
     namespace: openshift-sandboxed-containers-operator
   data:
     PODVM_DISTRO: "rhel"
     CAA_SRC: "https://github.com/confidential-containers/cloud-api-adaptor"
   ```

```
CAA_REF: "<cloud_api_adaptor_version>" (1)
DOWNLOAD_SOURCES: "no"
CONFIDENTIAL_COMPUTE_ENABLED: "yes"
UPDATE_PEERPODS_CM: "yes"
ORG_ID: "<rhel_organization_id>"
ACTIVATION_KEY: "<rhel_activation_key>" (2)
IMAGE_NAME: "<podvm_libvirt_image>"
PODVM_IMAGE_URI: "oci::<image_repo_url>:<image_tag>::<image_path>" (3)
SE_BOOT: "true" (4)
BASE_OS_VERSION: "<rhel_image_os_version>" (5)
```

(1) Specify the latest version of the Cloud API Adaptor source.

(2) Specify your RHEL activation key.

(3) Optional: Specify the following values if you created a container image:

- **image_repo_url**: Container registry URL.
- **image_tag**: Image tag.
- **image_path**: Image path. Default: **/image/podvm.qcow2**.

(4) **SE_BOOT: "true"** enables IBM Secure Execution for an Operator-built image. Set to **false** if you created a container image.

(5) Specify the RHEL image operating system version. IBM Z® Secure Execution supports RHEL 9.4 and later versions.

2. Create the config map by running the following command:

```
$ oc apply -f libvirt-podvm-image-cm.yaml
```

The libvirt pod VM image config map is created for your libvirt provider.

5.2.8. Creating the KVM host secret

You must create the secret for your KVM host.

Procedure

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

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

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

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

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

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

5. Delete the SSH keys you created:

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

5.2.9. Creating the KataConfig custom resource

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

Creating the **KataConfig** CR triggers the OpenShift sandboxed containers Operator to do the following: * Create a **RuntimeClass** CR named **kata-remote** with a default configuration. This enables users to configure workloads to use **kata-remote** as the runtime by referencing the CR in the **RuntimeClassName** field. This CR also specifies the resource overhead for the runtime.

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

IMPORTANT

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

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

Prerequisites

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

Procedure

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

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

```
# kataConfigPoolSelector:
#   matchLabels:
#     <label_key>: '<label_value>' 1
```

1 Optional: If you have applied node labels to install **kata-remote** on specific nodes, specify the key and value, for example, **osc: 'true'**.

2. Create the **KataConfig** CR by running the following command:

   ```
   $ oc apply -f example-kataconfig.yaml
   ```

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

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

3. Monitor the installation progress by running the following command:

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

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

4. Verify that you have built the peer pod image and uploaded it to the libvirt volume by running the following command:

   ```
   $ oc describe configmap peer-pods-cm -n openshift-sandboxed-containers-operator
   ```

   Example output

   ```
   Name: peer-pods-cm
   Namespace: openshift-sandboxed-containers-operator
   Labels: <none>
   Annotations: <none>

   Data
   ====
   CLOUD_PROVIDER: libvirt

   BinaryData
   ====
   Events: <none>
   ```

5. Monitor the **kata-oc** machine config pool progress to ensure that it is in the **UPDATED** state, when **UPDATEDMACHINECOUNT** equals **MACHINECOUNT**, by running the following command:

   ```
   $ watch oc get mcp/kata-oc
   ```

6. Verify the daemon set by running the following command:

   ```
   $ oc get -n openshift-sandboxed-containers-operator ds/peerpodconfig-ctrl-caa-daemon
   ```

7. Verify the runtime classes by running the following command:

```
$ oc get runtimeclass
```

Example output

```
NAME           HANDLER        AGE
kata           kata           152m
kata-remote    kata-remote    152m
```

5.2.10. Configuring workload objects

You must configure OpenShift sandboxed containers workload objects by setting **kata-remote** as the runtime class for the following pod-templated objects:

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

IMPORTANT

Do not deploy workloads in an Operator namespace. Create a dedicated namespace for these resources.

Prerequisites

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

Procedure

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

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

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

Verification

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

5.3. DEPLOYING CONFIDENTIAL CONTAINERS ON IBM Z AND IBM LINUXONE

You can deploy Confidential Containers on IBM Z® and IBM® LinuxONE after you deploy OpenShift sandboxed containers.

IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see Technology Preview Features Support Scope .

Cluster requirements

- You have installed Red Hat OpenShift Container Platform 4.15 or later on the cluster where you are installing the Confidential compute attestation Operator.

You deploy Confidential Containers by performing the following steps:

1. Install the Confidential compute attestation Operator.
2. Create the route for Trustee.
3. Enable the Confidential Containers feature gate.
4. Update the peer pods config map.
5. Delete the **KataConfig** custom resource (CR).
6. Update the peer pods secret.
7. Re-create the **KataConfig** CR.
8. Create the Trustee authentication secret.
9. Create the Trustee config map.
10. Obtain the IBM Secure Execution (SE) header.
11. Configure the SE certificates and keys.
12. Configure attestation policies:
    a. Create reference values.
    b. Create secrets for attested clients.
    c. Create the resource access policy.
13. Create the attestation policy for SE.

14. Create the **KbsConfig** CR.

15. Verify the attestation process.

5.3.1. Installing the Confidential compute attestation Operator

You can install the Confidential compute attestation Operator on a cluster with an x86-64 architecture by using the CLI.

Prerequisites

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

Procedure

1. Create a **trustee-namespace.yaml** manifest file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: trustee-operator-system
```

2. Create the **trustee-operator-system** namespace by running the following command:

```
$ oc apply -f trustee-namespace.yaml
```

3. Create a **trustee-operatorgroup.yaml** manifest file:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: trustee-operator-group
  namespace: trustee-operator-system
spec:
  targetNamespaces:
  - trustee-operator-system
```

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

```
$ oc apply -f trustee-operatorgroup.yaml
```

5. Create a **trustee-subscription.yaml** manifest file:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: trustee-operator
  namespace: trustee-operator-system
spec:
  channel: stable
  installPlanApproval: Automatic
  name: trustee-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: trustee-operator.v0.1.0
```

6. Create the subscription by running the following command:

```
$ oc apply -f trustee-subscription.yaml
```

7. Verify that the Operator is correctly installed by running the following command:

```
$ oc get csv -n trustee-operator-system
```

   This command can take several minutes to complete.

8. Watch the process by running the following command:

```
$ watch oc get csv -n trustee-operator-system
```

Example output

```
NAME                      DISPLAY                        PHASE
trustee-operator.v0.1.0   Trustee Operator  0.1.0        Succeeded
```

5.3.2. Creating the route for Trustee

You can create a secure route with edge TLS termination for Trustee. External ingress traffic reaches the router pods as HTTPS and passes on to the Trustee pods as HTTP.

Prerequisites

- You have enabled the Confidential Containers feature gate.
- You have installed the Confidential compute attestation Operator.

Procedure

1. Create an edge route by running the following command:

```
$ oc create route edge --service=kbs-service --port kbs-port \
  -n trustee-operator-system
```

NOTE

Note: Currently, only a route with a valid CA-signed certificate is supported. You cannot use a route with self-signed certificate.

2. Set the **TRUSTEE_HOST** variable by running the following command:

```
$ TRUSTEE_HOST=$(oc get route -n trustee-operator-system kbs-service \
  -o jsonpath={.spec.host})
```

3. Verify the route by running the following command:

```
$ echo $TRUSTEE_HOST
```

Example output

```
kbs-service-trustee-operator-system.apps.memvjias.eastus.aroapp.io
```

5.3.3. Enabling the Confidential Containers feature gate

You must enable the Confidential Containers feature gate.

Procedure

1. Create a **cc-feature-gate.yaml** manifest file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: osc-feature-gates
  namespace: openshift-sandboxed-containers-operator
data:
  confidential: "true"
```

2. Create the config map by running the following command:

```
$ oc apply -f cc-feature-gate.yaml
```

5.3.4. Updating the peer pods config map

You must update the peer pods config map for Confidential Containers.

Procedure

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

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

2. Create the config map by running the following command:

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

3. Restart the **peerpodconfig-ctrl-caa-daemon** daemon set by running the following command:

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

5.3.5. Deleting the KataConfig custom resource

You can delete the **KataConfig** custom resource (CR) by using the command line.

Prerequisites

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

Procedure

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

```
$ oc delete kataconfig example-kataconfig
```

2. Verify that the custom resource was deleted by running the following command:

```
$ oc get kataconfig example-kataconfig
```

Example output

```
No example-kataconfig instances exist
```

5.3.6. Updating the peer pods secret

You must update the peer pods secret for Confidential Containers.

The secret stores credentials for creating the pod virtual machine (VM) image and peer pod instances.

By default, the OpenShift sandboxed containers Operator creates the secret based on the credentials used to create the cluster. However, you can manually create a secret that uses different credentials.

Prerequisites

- **REDHAT_OFFLINE_TOKEN**. You have generated this token to download the RHEL image at Red Hat API Tokens .
- **HKD_CRT**. The Host Key Document (HKD) certificate enables secure execution on IBM Z®. For more information, see Obtaining a host key document from Resource Link in the IBM documentation.

Procedure

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

```
apiVersion: v1
kind: Secret
metadata:
  name: peer-pods-secret
  namespace: openshift-sandboxed-containers-operator
type: Opaque
```

```
stringData:
  REDHAT_OFFLINE_TOKEN: "<rh_offline_token>" 1
  HKD_CRT: "<hkd_crt_value>" 2
```

1 Specify the Red Hat offline token, which is required for the Operator-built image.

2 Specify the HKD certificate value to enable IBM Secure Execution for the Operator-built image.

2. Create the secret by running the following command:

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

3. Restart the **peerpodconfig-ctrl-caa-daemon** daemon set by running the following command:

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

5.3.7. Re-creating the KataConfig custom resource

You must re-create the **KataConfig** custom resource (CR) for Confidential Containers.

IMPORTANT

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

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

Prerequisites

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

Procedure

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

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

```
# kataConfigPoolSelector:
#   matchLabels:
#     <label_key>: '<label_value>' 1
```

1 Optional: If you have applied node labels to install **kata-remote** on specific nodes, specify the key and value, for example, **cc: 'true'**.

2. Create the **KataConfig** CR by running the following command:

   ```
   $ oc apply -f example-kataconfig.yaml
   ```

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

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

3. Monitor the installation progress by running the following command:

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

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

4. Verify that you have built the peer pod image and uploaded it to the libvirt volume by running the following command:

   ```
   $ oc describe configmap peer-pods-cm -n openshift-sandboxed-containers-operator
   ```

   Example output

   ```
   Name: peer-pods-cm
   Namespace: openshift-sandboxed-containers-operator
   Labels: <none>
   Annotations: <none>

   Data
   ====
   CLOUD_PROVIDER: libvirt
   DISABLECVM: false 1
   LIBVIRT_IMAGE_ID: fa-pp-vol 2

   BinaryData
   ====
   Events: <none>
   ```

   1 Enables the Confidential VM during IBM Secure Execution for the Operator-built image.

   2 Contains a value if you have built the peer pod image and uploaded it to the libvirt volume.

5. Monitor the **kata-oc** machine config pool progress to ensure that it is in the **UPDATED** state, when **UPDATEDMACHINECOUNT** equals **MACHINECOUNT**, by running the following command:

```
$ watch oc get mcp/kata-oc
```

6. Verify the daemon set by running the following command:

```
$ oc get -n openshift-sandboxed-containers-operator ds/peerpodconfig-ctrl-caa-daemon
```

7. Verify the runtime classes by running the following command:

```
$ oc get runtimeclass
```

Example output

```
NAME           HANDLER        AGE
kata           kata           152m
kata-remote    kata-remote    152m
```

5.3.8. Creating the Trustee authentication secret

You must create the authentication secret for Trustee.

Prerequisites

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

Procedure

1. Create a private key by running the following command:

```
$ openssl genpkey -algorithm ed25519 > privateKey
```

2. Create a public key by running the following command:

```
$ openssl pkey -in privateKey -pubout -out publicKey
```

3. Create a secret by running the following command:

```
$ oc create secret generic kbs-auth-public-key --from-file=publicKey -n trustee-operator-
system
```

4. Verify the secret by running the following command:

```
$ oc get secret -n trustee-operator-system
```

5.3.9. Creating the Trustee config map

You must create the config map to configure the Trustee server.

Prerequisites

- You have created a route for Trustee.

Procedure

1. Create a **kbs-config-cm.yaml** manifest file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kbs-config-cm
  namespace: trustee-operator-system
data:
  kbs-config.json: |
    {
      "insecure_http" : true,
      "sockets": ["0.0.0.0:8080"],
      "auth_public_key": "/etc/auth-secret/publicKey",
      "attestation_token_config": {
        "attestation_token_type": "CoCo"
      },
      "repository_config": {
        "type": "LocalFs",
        "dir_path": "/opt/confidential-containers/kbs/repository"
      },
      "as_config": {
        "work_dir": "/opt/confidential-containers/attestation-service",
        "policy_engine": "opa",
        "attestation_token_broker": "Simple",
          "attestation_token_config": {
          "duration_min": 5
          },
        "rvps_config": {
          "store_type": "LocalJson",
          "store_config": {
            "file_path": "/opt/confidential-containers/rvps/reference-values/reference-values.json"
          }
         }
      },
      "policy_engine_config": {
        "policy_path": "/opt/confidential-containers/opa/policy.rego"
      }
    }
```

2. Create the config map by running the following command:

```
$ oc apply -f kbs-config-cm.yaml
```

5.3.10. Obtaining the IBM Secure Execution header

You must obtain the IBM Secure Execution (SE) header.

Prerequisites

- You have a network block storage device to store the SE header temporarily.

Procedure

1. Create a temporary folder for the SE header by running the following command:

```
$ mkdir -p /tmp/ibmse/hdr
```

2. Download the **pvextract-hdr** tool from IBM s390 Linux repository by running the following command:

```
$ wget https://github.com/ibm-s390-linux/s390-tools/raw/v2.33.1/rust/pvattest/tools/pvextract-
hdr -O /tmp/pvextract-hdr
```

3. Make the tool executable by running the following command:

```
$ chmod +x /tmp/pvextract-hdr
```

4. Set the **$IMAGE_OUTPUT_DIR** variable by running the following command:

```
$ export IMAGE=$IMAGE_OUTPUT_DIR/se-podvm-commit-short-id.qcow2
```

5. Set the **$IMAGE** variable by running the following command:

```
$ export IMAGE=/root/rooo/se-podvm-d1fb986-dirty-s390x.qcow2
```

6. Enable the **nbd** kernel module by running the following command:

```
$ modprobe nbd
```

7. Connect the SE image as a network block device (NBD) by running the following command:

```
$ qemu-nbd --connect=/dev/nbd0 $IMAGE
```

8. Create a mount directory for the SE image by running the following command:

```
$ mkdir -p /mnt/se-image/
```

9. Pause the process by running the following command:

```
$ sleep 1
```

10. List your block devices by running the following command:

```
$ lsblk
```

Example output

```
nbd0                           43:0    0  100G  0 disk
├─nbd0p1                           43:1    0  255M  0 part
├─nbd0p2                           43:2    0    6G  0 part
│ └─luks-e23e15fa-9c2a-45a5-9275-aae9d8e709c3 253:2    0    6G  0 crypt
└─nbd0p3                           43:3    0 12.4G  0 part
nbd1                           43:32   0   20G  0 disk
```

```
├─nbd1p1                               43:33   0  255M  0 part
├─nbd1p2                               43:34   0    6G  0 part
│ └─luks-5a540f7c-c0cb-419b-95e0-487670d91525 253:3    0    6G  0 crypt
└─nbd1p3                               43:35   0 86.9G  0 part
nbd2                               43:64   0    0B  0 disk
nbd3                               43:96   0    0B  0 disk
nbd4                               43:128  0    0B  0 disk
nbd5                               43:160  0    0B  0 disk
nbd6                               43:192  0    0B  0 disk
nbd7                               43:224  0    0B  0 disk
nbd8                               43:256  0    0B  0 disk
nbd9                               43:288  0    0B  0 disk
nbd10                               43:320  0    0B  0 disk
```

11. Mount the SE image directory on an available NBD partition and extract the SE header by running the following command:

    ```
    $ mount /dev/<nbdXp1> /mnt/se-image/ /tmp/pvextract-hdr \
      -o /tmp/ibmse/hdr/hdr.bin /mnt/se-image/se.img
    ```

    Example output

    ```
    SE header found at offset 0x014000
    SE header written to '/tmp/ibmse/hdr/hdr.bin' (640 bytes)
    ```

    The following error is displayed if the NBD is unavailable:

    ```
    mount: /mnt/se-image: can't read superblock on /dev/nbd0p1
    ```

12. Unmount the SE image directory by running the following command:

    ```
    $ umount /mnt/se-image/
    ```

13. Disconnect the network block storage device by running the following command:

    ```
    $ qemu-nbd --disconnect /dev/nbd0
    ```

5.3.11. Configuring the IBM Secure Execution certificates and keys

You must configure the IBM Secure Execution (SE) certificates and keys for your worker nodes.

Prerequisites

- You have the IP address of the bastion node.
- You have the internal IP addresses of the worker nodes.

Procedure

1. Obtain the attestation policy fields by performing the following steps:

    a. Download the **se_parse_hdr.py** script from the OpenShift Trustee repository by running the following command:

```
$ wget https://github.com/openshift/trustee/raw/main/attestation-
service/verifier/src/se/se_parse_hdr.py -O /tmp/se_parse_hdr.py
```

b. Create a temporary directory for the SE Host Key Document (HKD) certificate by running the following command:

```
$ mkdir /tmp/ibmse/hkds/
```

c. Copy your Host Key Document (HKD) certificate to the temporary directory by running the following command:

```
$ cp ~/path/to/<hkd_cert.crt> /tmp/ibmse/hkds/<hkd_cert.crt>
```

NOTE

The HKD certificate must be the same certificate that you downloaded when you created the peer pods secret.

d. Obtain the attestation policy fields by running the **se_parse_hdr.py** script:

```
$ python3 /tmp/se_parse_hdr.py /tmp/ibmse/hdr/hdr.bin /tmp/ibmse/hkds/<hkd_cert.crt>
```

Example output

```
...
  ================================================
  se.image_phkh: xxx
  se.version: 256
  se.tag: xxx
  se.attestation_phkh: xxx
```

Record these values for the SE attestation policy config map.

2. Obtain the certificates and certificate revocation lists (CRLs) by performing the following steps:

a. Create a temporary directory for certificates by running the following command:

```
$ mkdir /tmp/ibmse/certs
```

b. Download the **ibm-z-host-key-signing-gen2.crt** certificate by running the following command:

```
$ wget https://www.ibm.com/support/resourcelink/api/content/public/ibm-z-host-key-
signing-gen2.crt -O /tmp/ibmse/certs/ibm-z-host-key-signing-gen2.crt
```

c. Download the **DigiCertCA.crt** certificate by running the following command:

```
$ wget https://www.ibm.com/support/resourcelink/api/content/public/DigiCertCA.crt -O
/tmp/ibmse/certs/DigiCertCA.crt
```

d. Create a temporary directory for the CRLs by running the following command:

```
$ mkdir /tmp/ibmse/crls
```

e. Download the **DigiCertTrustedRootG4.crl** file by running the following command:

```
$ wget http://crl3.digicert.com/DigiCertTrustedRootG4.crl -O
/tmp/ibmse/crls/DigiCertTrustedRootG4.crl
```

f. Download the **DigiCertTrustedG4CodeSigningRSA4096SHA3842021CA1.crl** file by running the following command:

```
$ wget
http://crl3.digicert.com/DigiCertTrustedG4CodeSigningRSA4096SHA3842021CA1.crl -O
/tmp/ibmse/crls/DigiCertTrustedG4CodeSigningRSA4096SHA3842021CA1.crl
```

3. Generate the RSA keys:

a. Generate an RSA key pair by running the following command:

```
$ openssl genrsa -aes256 -passout pass:<password> -out /tmp/encrypt_key-psw.pem
4096 1
```

1 Specify the RSA key password.

b. Create a temporary directory for the RSA keys by running the following command:

```
$ mkdir /tmp/ibmse/rsa
```

c. Create an **encrypt_key.pub** key by running the following command:

```
$ openssl rsa -in /tmp/encrypt_key-psw.pem -passin pass:<password> -pubout -out
/tmp/ibmse/rsa/encrypt_key.pub
```

d. Create an **encrypt_key.pem** key by running the following command:

```
$ openssl rsa -in /tmp/encrypt_key-psw.pem -passin pass:<password> -out
/tmp/ibmse/rsa/encrypt_key.pem
```

4. Verify the structure of the **/tmp/ibmse** directory by running the following command:

```
$ tree /tmp/ibmse
```

Example output

```
/tmp/ibmse
├── certs
│   ├── ibm-z-host-key-signing-gen2.crt
|   └── DigiCertCA.crt
├── crls
│   └── ibm-z-host-key-gen2.crl
│   └── DigiCertTrustedRootG4.crl
│   └── DigiCertTrustedG4CodeSigningRSA4096SHA3842021CA1.crl
├── hdr
```

```
│   └── hdr.bin
├── hkds
│   └── <hkd_cert.crt>
└── rsa
    ├── encrypt_key.pem
    └── encrypt_key.pub
```

5. Copy these files to the OpenShift Container Platform worker nodes by performing the following steps:

   a. Create a compressed file from the /**tmp/ibmse** directory by running the following command:

   ```
   $ tar -czf ibmse.tar.gz -C /tmp/ibmse
   ```

   b. Copy the **.tar.gz** file to the bastion node in your cluster by running the following command:

   ```
   $ scp /tmp/ibmse.tar.gz root@<ocp_bastion_ip>:/tmp 1
   ```

   1 Specify the IP address of the bastion node.

   c. Connect to the bastion node over SSH by running the following command:

   ```
   $ ssh root@<ocp_bastion_ip>
   ```

   d. Copy the **.tar.gz** file to each worker node by running the following command:

   ```
   $ scp /tmp/ibmse.tar.gz core@<worker_node_ip>:/tmp 1
   ```

   1 Specify the IP address of the worker node.

   e. Extract the **.tar.gz** on each worker node by running the following command:

   ```
   $ ssh core@<worker_node_ip> 'sudo mkdir -p /opt/confidential-containers/ && sudo tar -
   xzf /tmp/ibmse.tar.gz -C /opt/confidential-containers/'
   ```

   f. Update the **ibmse** folder permissions by running the following command:

   ```
   $ ssh core@<worker_node_ip> 'sudo chmod -R 755 /opt/confidential-containers/ibmse/'
   ```

5.3.12. Configuring attestation policies

You can configure the following attestation policy settings:

Reference values

You can configure reference values for the Reference Value Provider Service (RVPS) by specifying the trusted digests of your hardware platform.
The client collects measurements from the running software, the Trusted Execution Environment (TEE) hardware and firmware and it submits a quote with the claims to the Attestation Server. These measurements must match the trusted digests registered to the Trustee. This process ensures that the confidential VM (CVM) is running the expected software stack and has not been tampered with.

Secrets for clients

You must create one or more secrets to share with attested clients.

Resource access policy

You must configure a policy for the Trustee policy engine to determine which resources to access. Do not confuse the Trustee policy engine with the Attestation Service policy engine, which determines the validity of TEE evidence.

Attestation policy

You must create an attestation policy for IBM Secure Execution.

Procedure

1. Create an **rvps-configmap.yaml** manifest file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: rvps-reference-values
  namespace: trustee-operator-system
data:
  reference-values.json: |
    [ (1)
    ]
```

(1) Leave this value empty.

2. Create the RVPS config map by running the following command:

```
$ oc apply -f rvps-configmap.yaml
```

3. Create one or more secrets to share with attested clients according to the following example:

```
$ oc create secret generic kbsres1 --from-literal key1=<res1val1> \
  --from-literal key2=<res1val2> -n trustee-operator-system
```

In this example, the **kbsres1** secret has two entries (**key1**, **key2**), which the Trustee clients retrieve. You can add more secrets according to your requirements.

4. Create a **resourcepolicy-configmap.yaml** manifest file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: resource-policy
  namespace: trustee-operator-system
data:
  policy.rego: | (1)
    package policy (2)
    path := split(data["resource-path"], "/")
    default allow = false
    allow {
      count(path) == 3
      input["tee"] == "se"
    }
```

1. The name of the resource policy, **policy.rego**, must match the resource policy defined in the Trustee config map.

2. The resource policy follows the Open Policy Agent specification. This example allows the retrieval of all resources when the TEE is not the sample attester.

5. Create the resource policy config map by running the following command:

```
$ oc apply -f resourcepolicy-configmap.yaml
```

6. Create an **attestation-policy.yaml** manifest file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: attestation-policy
  namespace: trustee-operator-system
data:
  default.rego: | 1
    package policy
    import rego.v1
    default allow = false
    converted_version := sprintf("%v", [input["se.version"]])
    allow if {
        input["se.attestation_phkh"] == "<se.attestation_phkh>" 2
        input["se.image_phkh"] == "<se.image_phkh>"
        input["se.tag"] == "<se.tag>" 3
        input["se.user_data"] == "00"
        converted_version == "256"
    }
```

1. Do not modify the policy name.

2. Specify the attestation policy fields you obtained by running the **se_parse_hdr.py** script.

7. Create the attestation policy config map by running the following command:

```
$ oc apply -f attestation-policy.yaml
```

5.3.13. Creating the KbsConfig custom resource

You must create the **KbsConfig** custom resource (CR) to launch Trustee.

Then, you check the Trustee pods and pod logs to verify the configuration.

Procedure

1. Create a **kbsconfig-cr.yaml** manifest file:

```
apiVersion: confidentialcontainers.org/v1alpha1
kind: KbsConfig
metadata:
  labels:
    app.kubernetes.io/name: kbsconfig
    app.kubernetes.io/instance: kbsconfig
    app.kubernetes.io/part-of: trustee-operator
    app.kubernetes.io/managed-by: kustomize
    app.kubernetes.io/created-by: trustee-operator
  name: kbsconfig
  namespace: trustee-operator-system
spec:
  kbsConfigMapName: kbs-config-cm
  kbsAuthSecretName: kbs-auth-public-key
  kbsDeploymentType: AllInOneDeployment
  kbsRvpsRefValuesConfigMapName: rvps-reference-values
  kbsSecretResources: ["kbsres1"]
  kbsResourcePolicyConfigMapName: resource-policy
  kbsAttestationPolicyConfigMapName: attestation-policy
  kbsServiceType: NodePort
  ibmSEConfigSpec:
    certStorePvc: ibmse-pvc
```

2. Create the **KbsConfig** CR by running the following command:

```
$ oc apply -f kbsconfig-cr.yaml
```

Verification

1. Set the default project by running the following command:

```
$ oc project trustee-operator-system
```

2. Check the pods by running the following command:

```
$ oc get pods -n trustee-operator-system
```

Example output

```
NAME                                              READY   STATUS    RESTARTS   AGE
trustee-deployment-8585f98449-9bbgl               1/1     Running   0          22m
trustee-operator-controller-manager-5fbd44cd97-55dlh   2/2     Running   0          59m
```

3. Set the **POD_NAME** environmental variable by running the following command:

```
$ POD_NAME=$(oc get pods -l app=kbs -o jsonpath='{.items[0].metadata.name}' -n trustee-
operator-system)
```

4. Check the pod logs by running the following command:

```
$ oc logs -n trustee-operator-system $POD_NAME
```

Example output

```
[2024-05-30T13:44:24Z INFO  kbs] Using config file /etc/kbs-config/kbs-config.json
[2024-05-30T13:44:24Z WARN  attestation_service::rvps] No RVPS address provided and
will launch a built-in rvps
[2024-05-30T13:44:24Z INFO  attestation_service::token::simple] No Token Signer key in
config file, create an ephemeral key and without CA pubkey cert
[2024-05-30T13:44:24Z INFO  api_server] Starting HTTPS server at [0.0.0.0:8080]
[2024-05-30T13:44:24Z INFO  actix_server::builder] starting 12 workers
[2024-05-30T13:44:24Z INFO  actix_server::server] Tokio runtime found; starting in existing
Tokio runtime
```

5. Verify that the **kbs-service** is exposed on a node port by running the following command:

```
$ oc get svc kbs-service -n trustee-operator-system
```

Example output

```
NAME          TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
kbs-service   NodePort   198.51.100.54   <none>        8080:31862/TCP   23h
```

The **kbs-service** URL is `https://<worker_node_ip>:<node_port>`, for example, `https://172.16.0.56:31862`.

5.3.14. Verifying the attestation process

You can verify the attestation process by creating a test pod and retrieving its secret. The pod image deploys the KBS client, a tool for testing the Key Broker Service and basic attestation flows.

IMPORTANT

This procedure is an example to verify that attestation is working. Do not write sensitive data to standard I/O because the data can be captured by using a memory dump. Only data written to memory is encrypted.

Prerequisites

- You have created a route if the Trustee server and the test pod are not running in the same cluster.

Procedure

1. Create a **verification-pod.yaml** manifest file:

```
apiVersion: v1
kind: Pod
metadata:
  name: kbs-client
spec:
  containers:
  - name: kbs-client
    image: quay.io/confidential-containers/kbs-client:latest
    imagePullPolicy: IfNotPresent
    command:
      - sleep
      - "360000"
    env:
      - name: RUST_LOG
        value:  none
```

2. Create the pod by running the following command:

```
$ oc create -f verification-pod.yaml
```

3. Copy the **https.crt** file to the **kbs-client** pod by running the following command:

```
$ oc cp https.crt kbs-client:/
```

4. Fetch the pod secret by running the following command:

```
$ oc exec -it kbs-client -- kbs-client --cert-file https.crt \
  --url https://kbs-service:8080 get-resource \
  --path default/kbsres1/key1
```

Example output

```
res1val1
```

The Trustee server returns the secret only if the attestation is successful.

CHAPTER 6. MONITORING

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

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

6.1. ABOUT METRICS

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

OpenShift sandboxed containers metrics are collected for the following categories:

Kata agent metrics

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

Kata guest operating system metrics

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

Hypervisor metrics

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

Kata monitor metrics

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

Kata containerd shim v2 metrics

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

6.2. VIEWING METRICS

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

Prerequisites

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

Procedure

1. In the OpenShift Container Platform web console, navigate to **Observe → Metrics**.
2. In the input field, enter the query for the metric you want to observe.
   All kata-related metrics begin with **kata**. Typing **kata** displays a list of all available kata metrics.

The metrics from your query are visualized on the page.

Additional resources

- Querying metrics.
- Gathering data about your cluster .

CHAPTER 7. UNINSTALLING

You can uninstall OpenShift sandboxed containers and remove the Confidential Containers environment.

7.1. UNINSTALLING OPENSHIFT SANDBOXED CONTAINERS

You can uninstall OpenShift sandboxed containers by using the OpenShift Container Platform web console or the command line.

You uninstall OpenShift sandboxed containers by performing the following tasks:

1. Delete the workload pods.
2. Delete the **KataConfig** custom resource.
3. Uninstall the OpenShift sandboxed containers Operator.
4. Delete the **KataConfig** custom resource definition.

7.1.1. Uninstalling OpenShift sandboxed containers by using the web console

You can uninstall OpenShift sandboxed containers by using the OpenShift Container Platform web console.

7.1.1.1. Deleting workload pods

You can delete the OpenShift sandboxed containers workload pods by using the OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have a list of pods that use the OpenShift sandboxed containers runtime class.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Workloads** → **Pods**.
2. Enter the name of the pod that you want to delete in the **Search by name** field.
3. Click the pod name to open it.
4. On the **Details** page, check that **kata** or **kata-remote** is displayed for **Runtime class**.
5. Click the **Options** menu ⋮ and select **Delete Pod**.
6. Click **Delete**.

7.1.1.2. Deleting the KataConfig custom resource

You can delete the **KataConfig** custom resource (CR) by using the web console.

Deleting the **KataConfig** CR removes and uninstalls the **kata** runtime and its related resources from your cluster.

IMPORTANT

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

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

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have deleted all running pods that use **kata** as the **runtimeClass**.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators** → **Installed Operators**.
2. Enter **OpenShift sandboxed containers Operator** in the **Search by name** field.
3. Click the Operator to open it and then click the **KataConfig** tab.
4. Click the **Options** menu ⋮ and 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.

7.1.1.3. Uninstalling the OpenShift sandboxed containers Operator

You can uninstall the OpenShift sandboxed containers Operator by using OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have deleted the **KataConfig** custom resource.

Procedure

1. Navigate to **Operators** → **Installed Operators**.
2. Enter **OpenShift sandboxed containers Operator** in the **Search by name** field.
3. On the right side of the **Operator Details** page, select **Uninstall Operator** from the **Actions** list. An **Uninstall Operator?** dialog box is displayed.
4. Click **Uninstall** to remove the Operator, Operator deployments, and pods.
5. Navigate to **Administration** → **Namespaces**.
6. Enter **openshift-sandboxed-containers-operator** in the **Search by name** field.
7. Click the **Options** menu ⋮ and select **Delete Namespace**.
8. In the confirmation dialog, enter **openshift-sandboxed-containers-operator** and click **Delete**.

7.1.1.4. Deleting the KataConfig CRD

You can delete the **KataConfig** custom resource definition (CRD) by using the OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have deleted the **KataConfig** custom resource.
- You have uninstalled the OpenShift sandboxed containers Operator.

Procedure

1. In the web console, navigate to **Administration** → **CustomResourceDefinitions**.
2. Enter the **KataConfig** name in the **Search by name** field.
3. Click the **Options** menu and select **Delete CustomResourceDefinition**.
4. Click **Delete** in the confirmation window.

7.1.2. Uninstalling OpenShift sandboxed containers by using the CLI

You can uninstall OpenShift sandboxed containers by using the command-line interface (CLI).

7.1.2.1. Deleting workload pods

You can delete the OpenShift sandboxed containers workload pods by using the CLI.

Prerequisites

- You have the JSON processor (**jq**) utility installed.

Procedure

1. Search for the pods by running the following command:

   ```
   $ oc get pods -A -o json | jq -r '.items[] | \
     select(.spec.runtimeClassName == "<runtime>").metadata.name' (1)
   ```

   (1) Specify **kata** for bare metal deployments. Specify **kata-remote** for AWS, Azure, IBM Z®, and IBM® LinuxONE.

2. Delete each pod by running the following command:

   ```
   $ oc delete pod <pod>
   ```

7.1.2.2. Deleting the KataConfig custom resource

You can delete the **KataConfig** custom resource (CR) by using the command line.

Deleting the **KataConfig** CR removes the runtime and its related resources from your cluster.

IMPORTANT

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

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

Prerequisites

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

Procedure

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

   ```
   $ oc delete kataconfig example-kataconfig
   ```

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

IMPORTANT

When you delete the **KataConfig** CR, the CLI stops responding until all worker nodes reboot. You must for the deletion process to complete before performing the verification.

2. Verify that the custom resource was deleted by running the following command:

```
$ oc get kataconfig example-kataconfig
```

Example output

```
No example-kataconfig instances exist
```

7.1.2.3. Uninstalling the OpenShift sandboxed containers Operator

You can uninstall the OpenShift sandboxed containers Operator by using the command line.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have deleted the OpenShift sandboxed containers workload pods.
- You have deleted **KataConfig** custom resource.

Procedure

1. Delete the subscription by running the following command:

```
$ oc delete subscription sandboxed-containers-operator -n openshift-sandboxed-containers-
operator
```

2. Delete the namespace by running the following command:

```
$ oc delete namespace openshift-sandboxed-containers-operator
```

7.1.2.4. Deleting the KataConfig CRD

You can delete the **KataConfig** custom resource definition (CRD) by using the command line.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have deleted the **KataConfig** custom resource.
- You have uninstalled the OpenShift sandboxed containers Operator.

Procedure

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

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

2. Verify that the CRD was deleted by running the following command:

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

Example output

```
Unknown CRD kataconfigs.kataconfiguration.openshift.io
```

7.2. REMOVING THE CONFIDENTIAL CONTAINERS ENVIRONMENT

You can remove the Confidential Containers environment by using the OpenShift Container Platform web console or the command line.

You remove the Confidential Containers environment by performing the following tasks:

1. Delete the **KbsConfig** custom resource.
2. Uninstall the Confidential compute attestation Operator.
3. Delete the **KbsConfig** custom resource definition.

7.2.1. Removing the Confidential Containers environment by using the web console

You can remove the Confidential Containers environment by using the OpenShift Container Platform web console.

7.2.1.1. Deleting the KbsConfig custom resource

You can delete the **KbsConfig** custom resource (CR) by using the web console.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have uninstalled OpenShift sandboxed containers.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators** → **Installed Operators**.
2. Enter **Confidential compute attestation** in the **Search by name** field.
3. Click the Operator to open it and then click the **KbsConfig** tab.
4. Click the **Options** menu ⋮ and select **Delete KbsConfig**.

5. Click **Delete** in the confirmation window.

7.2.1.2. Uninstalling the Confidential compute attestation Operator

You can uninstall the Confidential compute attestation Operator by using OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have deleted the **KbsConfig** custom resource.

Procedure

1. Navigate to **Operators** → **Installed Operators**.
2. Enter **Confidential compute attestation** in the **Search by name** field.
3. On the right side of the **Operator Details** page, select **Uninstall Operator** from the **Actions** list. An **Uninstall Operator?** dialog box is displayed.
4. Click **Uninstall** to remove the Operator, Operator deployments, and pods.
5. Navigate to **Administration** → **Namespaces**.
6. Enter **trustee-operator-system** in the **Search by name** field.
7. Click the **Options** menu ⋮ and select **Delete Namespace**.
8. In the confirmation dialog, enter **trustee-operator-system** and click **Delete**.

7.2.1.3. Deleting the KbsConfig CRD

You can delete the **KbsConfig** custom resource definition (CRD) by using the OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have deleted the **KbsConfig** custom resource.
- You have uninstalled the Confidential compute attestation Operator.

Procedure

1. In the web console, navigate to **Administration** → **CustomResourceDefinitions**.
2. Enter the **KbsConfig** name in the **Search by name** field.
3. Click the **Options** menu and select **Delete CustomResourceDefinition**.
4. Click **Delete** in the confirmation window.

7.2.2. Removing the Confidential Containers environment by using the CLI

You can remove the Confidential Containers environment by using the command-line interface (CLI).

7.2.2.1. Deleting the KbsConfig custom resource

You can delete the **KbsConfig** custom resource (CR) by using the command line.

Prerequisites

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

Procedure

1. Delete the **KbsConfig** CR by running the following command:

   ```
   $ oc delete kbsconfig kbsconfig
   ```

2. Verify that the custom resource was deleted by running the following command:

   ```
   $ oc get kbsconfig kbsconfig
   ```

   Example output

   ```
   No kbsconfig instances exist
   ```

7.2.2.2. Uninstalling the Confidential compute attestation Operator

You can uninstall the Confidential compute attestation Operator by using the command line.

Prerequisites

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

Procedure

1. Delete the subscription by running the following command:

   ```
   $ oc delete subscription trustee-operator -n trustee-operator-system
   ```

2. Delete the namespace by running the following command:

   ```
   $ oc delete namespace trustee-operator-system
   ```

7.2.2.3. Deleting the KbsConfig CRD

You can delete the **KbsConfig** custom resource definition (CRD) by using the command line.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have deleted the **KbsConfig** custom resource.
- You have uninstalled the Confidential compute attestation Operator.

Procedure

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

   ```
   $ oc delete crd kbsconfigs.confidentialcontainers.org
   ```

2. Verify that the CRD was deleted by running the following command:

   ```
   $ oc get crd kbsconfigs.confidentialcontainers.org
   ```

   Example output

   ```
   Unknown CRD kbsconfigs.confidentialcontainers.org
   ```

CHAPTER 8. UPGRADING

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

1. Upgrade OpenShift Container Platform to update the **Kata** runtime and its dependencies.

2. Upgrade the OpenShift sandboxed containers Operator to update the Operator subscription.

You can upgrade OpenShift Container Platform before or after the OpenShift sandboxed containers Operator upgrade, with the one exception noted below. Always apply the **KataConfig** patch immediately after upgrading OpenShift sandboxed containers Operator.

8.1. UPGRADING RESOURCES

The OpenShift sandboxed containers resources are deployed onto the cluster using Red Hat Enterprise Linux CoreOS (RHCOS) extensions.

The RHCOS extension **sandboxed containers** contains the required components to run OpenShift sandboxed containers, such as the Kata containers runtime, the hypervisor QEMU, and other dependencies. You upgrade the extension by upgrading the cluster to a new release of OpenShift Container Platform.

For more information about upgrading OpenShift Container Platform, see Updating Clusters.

8.2. UPGRADING THE OPERATOR

Use Operator Lifecycle Manager (OLM) to upgrade the OpenShift sandboxed containers Operator either manually or automatically. Selecting between manual or automatic upgrade during the initial deployment determines the future upgrade mode. For manual upgrades, the OpenShift Container Platform web console shows the available updates that can be installed by the cluster administrator.

For more information about upgrading the OpenShift sandboxed containers Operator in Operator Lifecycle Manager (OLM), see Updating installed Operators.

CHAPTER 9. TROUBLESHOOTING

When troubleshooting OpenShift sandboxed containers, you can open a support case and provide debugging information using the **must-gather** tool.

If you are a cluster administrator, you can also review logs on your own, enabling a more detailed level of logs.

9.1. COLLECTING DATA FOR RED HAT SUPPORT

When opening a support case, it is helpful to provide debugging information about your cluster to Red Hat Support.

The **must-gather** tool enables you to collect diagnostic information about your OpenShift Container Platform cluster, including virtual machines and other data related to OpenShift sandboxed containers.

For prompt support, supply diagnostic information for both OpenShift Container Platform and OpenShift sandboxed containers.

Using the must-gather tool

The **oc adm must-gather** CLI command collects the information from your cluster that is most likely needed for debugging issues, including:

- Resource definitions
- Service logs

By default, the **oc adm must-gather** command uses the default plugin image and writes into **./must-gather.local**.

Alternatively, you can collect specific information by running the command with the appropriate arguments as described in the following sections:

- To collect data related to one or more specific features, use the **--image** argument with an image, as listed in a following section.
  For example:

  ```
  $ oc adm must-gather --image=registry.redhat.io/openshift-sandboxed-containers/osc-must-gather-rhel9:1.7.0
  ```

- To collect the audit logs, use the **-- /usr/bin/gather_audit_logs** argument, as described in a following section.
  For example:

  ```
  $ oc adm must-gather -- /usr/bin/gather_audit_logs
  ```

NOTE

Audit logs are not collected as part of the default set of information to reduce the size of the files.

When you run **oc adm must-gather**, a new pod with a random name is created in a new project on the cluster. The data is collected on that pod and saved in a new directory that starts with **must-gather.local**. This directory is created in the current working directory.

For example:

```
NAMESPACE                     NAME               READY   STATUS      RESTARTS      AGE
...
openshift-must-gather-5drcj   must-gather-bklx4   2/2     Running     0            72s
openshift-must-gather-5drcj   must-gather-s8sdh   2/2     Running     0            72s
...
```

Optionally, you can run the **oc adm must-gather** command in a specific namespace by using the **--run-namespace** option.

For example:

```
$ oc adm must-gather --run-namespace <namespace> --image=registry.redhat.io/openshift-sandboxed-containers/osc-must-gather-rhel9:1.7.0
```

9.2. COLLECTING LOG DATA

The following features and objects are associated with OpenShift sandboxed containers:

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

You can collect the following component logs for each pod running with the **kata** runtime:

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

9.2.1. Enabling debug logs for CRI-O runtime

You can enable debug logs by updating the **logLevel** field in the **KataConfig** CR. This changes the log level in the CRI-O runtime for the worker nodes running OpenShift sandboxed containers.

Prerequisites

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

Procedure

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

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

2. Monitor the **kata-oc** machine config pool until the value of **UPDATED** is **True**, indicating that all worker nodes are updated:

   ```
   $ oc get mcp kata-oc
   ```

   Example output

   ```
   NAME     CONFIG              UPDATED  UPDATING  DEGRADED  MACHINECOUNT
   READYMACHINECOUNT  UPDATEDMACHINECOUNT  DEGRADEDMACHINECOUNT
   AGE
   kata-oc  rendered-kata-oc-169   False    True      False    3          1              1
   0                9h
   ```

Verification

1. Start a debug session with a node in the machine config pool:

   ```
   $ oc debug node/<node_name>
   ```

2. Change the root directory to **/host**:

   ```
   # chroot /host
   ```

3. Verify the changes in the **crio.conf** file:

   ```
   # crio config | egrep 'log_level
   ```

   Example output

   ```
   log_level = "debug"
   ```

9.2.2. Viewing debug logs for components

Cluster administrators can use the debug logs to troubleshoot issues. The logs for each node are printed to the node journal.

You can review the logs for the following OpenShift sandboxed containers components:

- Kata agent
- Kata runtime (**containerd-shim-kata-v2**)
- **virtiofsd**

QEMU only generates warning and error logs. These warnings and errors print to the node journal in both the Kata runtime logs and the CRI-O logs with an extra **qemuPid** field.

Example of QEMU logs

```
Mar 11 11:57:28 openshift-worker-0 kata[2241647]: time="2023-03-11T11:57:28.587116986Z"
level=info msg="Start logging QEMU (qemuPid=2241693)" name=containerd-shim-v2 pid=2241647
sandbox=d1d4d68efc35e5ccb4331af73da459c13f46269b512774aa6bde7da34db48987
source=virtcontainers/hypervisor subsystem=qemu
```

```
Mar 11 11:57:28 openshift-worker-0 kata[2241647]: time="2023-03-11T11:57:28.607339014Z"
level=error msg="qemu-kvm: -machine q35,accel=kvm,kernel_irqchip=split,foo: Expected '=' after
parameter 'foo'" name=containerd-shim-v2 pid=2241647 qemuPid=2241693
sandbox=d1d4d68efc35e5ccb4331af73da459c13f46269b512774aa6bde7da34db48987
source=virtcontainers/hypervisor subsystem=qemu

Mar 11 11:57:28 openshift-worker-0 kata[2241647]: time="2023-03-11T11:57:28.60890737Z"
level=info msg="Stop logging QEMU (qemuPid=2241693)" name=containerd-shim-v2 pid=2241647
sandbox=d1d4d68efc35e5ccb4331af73da459c13f46269b512774aa6bde7da34db48987
source=virtcontainers/hypervisor subsystem=qemu
```

The Kata runtime prints **Start logging QEMU** when QEMU starts, and **Stop Logging QEMU** when QEMU stops. The error appears in between these two log messages with the **qemuPid** field. The actual error message from QEMU appears in red.

The console of the QEMU guest is printed to the node journal as well. You can view the guest console logs together with the Kata agent logs.

Prerequisites

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

Procedure

- To review the Kata agent logs and guest console logs, run the following command:

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t kata -g “reading guest
console”
```

- To review the Kata runtime logs, run the following command:

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t kata
```

- To review the **virtiofsd** logs, run the following command:

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t virtiofsd
```

- To review the QEMU logs, run the following command:

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t kata -g "qemuPid=\d+"
```

Additional resources

- Gathering data about your cluster in the OpenShift Container Platform documentation

APPENDIX A. KATACONFIG STATUS MESSAGES

The following table displays the status messages for the **KataConfig** custom resource (CR) for a cluster with two worker nodes.

Table A.1. **KataConfig** status messages

Status	Description

Initial installation
 	
When a **KataConfig** CR is created and starts installing **kata-remote** on both workers, the following status is displayed for a few seconds.
	```
	conditions:
	  message: Performing initial installation of kata-remote on cluster
	  reason: Installing
	  status: 'True'
	  type: InProgress
	kataNodes:
	 nodeCount: 0
	 readyNodeCount: 0
	```

Installing
 	
Within a few seconds the status changes.
	```
	kataNodes:
	 nodeCount: 2
	 readyNodeCount: 0
	 waitingToInstall:
	 - worker-0
	 - worker-1
	```

Installing (Worker-1 installation starting)
 	
For a short period of time, the status changes, signifying that one node has initiated the installation of **kata-remote**, while the other is in a waiting state. This is because only one node can be unavailable at any given time. The **nodeCount** remains at 2 because both nodes will eventually receive **kata-remote**, but the **readyNodeCount** is currently 0 as neither of them has reached that state yet.
	```
	kataNodes:
	 installing:
	 - worker-1
	 nodeCount: 2
	 readyNodeCount: 0
	 waitingToInstall:
	 - worker-0
	```

Installing (Worker-1 installed, worker-0 installation started)
 	
After some time, **worker-1** will complete its installation, causing a change in the status. The **readyNodeCount** is updated to 1, indicating that **worker-1** is now prepared to execute **kata-remote** workloads. You cannot schedule or run **kata-remote** workloads until the runtime class is created at the end of the installation process.
	```
	kataNodes:
	 installed:
	 - worker-1
	 installing:
	 - worker-0
	 nodeCount: 2
	 readyNodeCount: 1
	```

Status	Description

Installed
 	
When installed, both workers are listed as installed, and the **InProgress** condition transitions to **False** without specifying a reason, indicating the successful installation of **kata-remote** on the cluster.
	`conditions:`
	`  message: ""`
	`  reason: ""`
	`  status: 'False'`
	`  type: InProgress`
	`kataNodes:`
	` installed:`
	` - worker-0`
	` - worker-1`
	` nodeCount: 2`
	` readyNodeCount: 2`

Status	Description

Initial uninstall
 	
If **kata-remote** is installed on both workers, and you delete the **KataConfig** to remove **kata-remote** from the cluster, both workers briefly enter a waiting state for a few seconds.
	`conditions:`
	`  message: Removing kata-remote from cluster`
	`  reason: Uninstalling`
	`  status: 'True'`
	`  type: InProgress`
	`kataNodes:`
	` nodeCount: 0`
	` readyNodeCount: 0`
	` waitingToUninstall:`
	` - worker-0`
	` - worker-1`

Uninstalling
 	
After a few seconds, one of the workers starts uninstalling.
	`kataNodes:`
	` nodeCount: 0`
	` readyNodeCount: 0`
	` uninstalling:`
	` - worker-1`
	` waitingToUninstall:`
	` - worker-0`

Uninstalling
 	
Worker-1 finishes and worker-0 starts uninstalling.
	`kataNodes:`
	` nodeCount: 0`
	` readyNodeCount: 0`
	` uninstalling:`
	` - worker-0`

NOTE

The **reason** field can also report the following causes:

- **Failed**: This is reported if the node cannot finish its transition. The **status** reports **True** and the **message** is **Node <node_name> Degraded: <error_message_from_the_node>**.
- **BlockedByExistingKataPods**: This is reported if there are pods running on a cluster that use the **kata-remote** runtime while **kata-remote** is being uninstalled. The **status** field is **False** and the **message** is **Existing pods using "kata-remote" RuntimeClass found. Please delete the pods manually for KataConfig deletion to proceed**. There could also be a technical error message reported like **Failed to list kata pods: <error_message>** if communication with the cluster control plane fails.