OpenShift sandboxed containers 1.6

用户指南

在 OpenShift Container Platform 中部署沙盒容器

Last Updated: 2024-09-23

OpenShift sandboxed containers 1.6 用户指南

在 OpenShift Container Platform 中部署沙盒容器

法律通告

摘要

在裸机、公共云和 IBM 平台上的 OpenShift Container Platform 中部署 OpenShift 沙盒容器。

目录

前言

对红帽文档提供反馈

您可以通过在 Jira 中提交 **Create Issue** 表单来提供反馈或报告错误。Jira 问题将在 Red Hat Hybrid Cloud Infrastructure Jira 项目中创建，您可以在其中跟踪您反馈的进度。

1. 确保您已登录到 Jira。如果您没有 JIRA 帐户，您必须创建一个 Red Hat Jira account 。
2. 启动 **Create Issue** 表单。
3. 完成 **Summary**、**Description** 和 **Reporter** 字段。
   在 **Description** 字段中，包含文档 URL、章节号以及问题的详细描述。
4. 点 **Create**。

第 1 章 关于 OPENSHIFT 沙盒容器

用于 OpenShift Container Platform 的 OpenShift 沙盒容器将 Kata 容器集成为可选运行时，通过在轻量级虚拟机中运行容器化应用程序来提供增强的安全性和隔离功能。此集成为敏感工作负载提供了一个更安全的运行时环境，而无需对现有 OpenShift 工作流进行大量更改。此运行时支持专用虚拟机(VM)中的容器，从而改进了工作负载隔离。

1.1. 功能

OpenShift 沙盒容器提供以下功能：

运行特权或不受信任的工作负载

您可以安全地运行需要特定特权的工作负载，而无需通过运行特权容器来破坏集群节点的风险。需要特殊权限的工作负载包括：

- 需要内核的特殊功能的工作负载，除了标准容器运行时（如 CRI-O）授予的默认功能外，例如访问低级别网络功能。
- 需要提高 root 特权的工作负载，例如访问特定物理设备。使用 OpenShift 沙盒容器时，只能将特定的设备传递给虚拟机(VM)，确保工作负载无法访问或错误配置系统的其余部分。
- 用于安装或使用 **set-uid** root 二进制文件的工作负载。这些二进制文件授予特殊权限，因此可能会造成安全风险。使用 OpenShift 沙盒容器时，对虚拟机有额外的权限，不授予对集群节点的特殊访问权限。
  有些工作负载需要专门用于配置集群节点的权限。此类工作负载应该仍然使用特权容器，因为在虚拟机上运行可能会阻止它们正常工作。

确保敏感工作负载的隔离

Red Hat OpenShift Container Platform 的 OpenShift 沙盒容器将 Kata 容器集成为可选运行时，通过在轻量级虚拟机中运行容器化应用程序来提供增强的安全性和隔离功能。此集成为敏感工作负载提供了一个更安全的运行时环境，而无需对现有 OpenShift 工作流进行大量更改。此运行时支持专用虚拟机(VM)中的容器，从而改进了工作负载隔离。

确保每个工作负载的内核隔离

您可以运行需要自定义内核调整（如 **sysctl**、调度程序更改或缓存调整）以及创建自定义内核模块（如 **树外** 或特殊参数）的工作负载。

在租户间共享相同的工作负载

您可以从共享同一 OpenShift Container Platform 集群的不同机构运行支持许多用户（租户）的工作负载。系统还支持从多个供应商运行第三方工作负载，如容器网络功能(CNF)和企业应用程序。例如，第三方 CNF 可能不希望其自定义设置与数据包调整或由其他应用程序设置的 **sysctl** 变量干扰。在完全隔离的内核内运行有助于防止"邻居噪音"配置问题。

确保正确隔离和沙盒测试软件

您可以使用已知漏洞运行容器化工作负载，或处理现有应用程序中的问题。通过这种隔离，管理员可以为开发人员提供对 pod 的管理控制，这在开发人员想要测试或验证管理员通常授予的配置时很有用。例如，管理员可以安全地将内核数据包过滤(eBPF)委派给开发人员。eBPF 需要 **CAP_ADMIN** 或 **CAP_BPF** 特权，因此不允许在标准 CRI-O 配置下，因为这会授予容器主机 worker 节点上的每个进程的访问权限。同样，管理员可以授予对 **SystemTap** 等入侵工具的访问权限，或者支持在开发期间加载自定义内核模块。

确保通过虚拟机边界的默认资源控制

默认情况下，OpenShift 沙盒容器以强大和安全的方式管理 CPU、内存、存储和网络等资源。由于 OpenShift 沙盒容器部署到虚拟机上，因此额外的隔离层和安全性可为资源提供更精细的访问控制。例如，错误容器将无法为虚拟机分配超过可用内存更多的内存。相反，需要专用访问网卡或磁盘的容器可以完全控制该设备，而无需访问其他设备。

1.2. 与 OPENSHIFT CONTAINER PLATFORM 的兼容性

OpenShift Container Platform 平台所需的功能由两个主要组件支持：

- Kata Runtime：包括 Red Hat Enterprise Linux CoreOS (RHCOS)和每个 OpenShift Container Platform 发行版本的 更新。
- OpenShift 沙盒容器 Operator：使用 Web 控制台或 OpenShift CLI (**oc**)安装 Operator。

OpenShift 沙盒容器 Operator 是一个 Rolling Stream Operator，这意味着最新版本是唯一受支持的版本。它可用于所有当前支持的 OpenShift Container Platform 版本。如需更多信息，请参阅 OpenShift Container Platform 生命周期政策 以了解更多详细信息。

Operator 依赖于 RHCOS 主机及其在其中运行的环境的功能。

注意

您必须在 worker 节点上安装 Red Hat Enterprise Linux CoreOS (RHCOS)。不支持 RHEL 节点。

OpenShift 沙盒容器和 OpenShift Container Platform 版本之间的以下兼容性列表用于标识兼容的功能和环境。

表 1.1. 支持的构架

架构	OpenShift Container Platform 版本
x86_64	4.8 或更高版本
s390x	4.14 或更高版本

部署 Kata 容器运行时的方法有两种：

- 裸机
- 对等 pod

对等 pod 技术从 OpenShift 沙盒容器 1.5 / OpenShift Container Platform 4.14 开始，允许在公有云中部署 OpenShift 沙盒容器。

表 1.2. OpenShift 版本的功能可用性

功能	部署方法	OpenShift Container Platform 4.15	OpenShift Container Platform 4.16
机密容器[1]	裸机	否	否
	对等 pod	开发者预览	开发者预览
GPU 支持[2]	裸机	否	否
	对等 pod	开发者预览	开发者预览

1. 机密容器仅在 AMD SEV-SNP 上被支持。
2. s390x 不提供 GPU 功能。

表 1.3. OpenShift 沙盒容器支持的平台

平台	对等 pod	GPU	机密容器
AWS Cloud Computing Services	是	开发者预览	否
Microsoft Azure Cloud Computing Services	是	开发者预览	开发者预览

其他资源

- 开发者预览支持范围
- 在 AWS 上部署工作负载
- 在 Azure 上部署工作负载
- 在裸机上部署工作负载

1.3. 节点资格检查

在部署 OpenShift 沙盒容器前，您可以检查裸机集群中的节点是否可以运行 OpenShift 沙盒容器。节点不合格的最常见原因是没有虚拟化支持。如果您在不符合节点上运行沙盒工作负载，则会出现错误。

高级工作流

1. 安装 Node Feature Discovery Operator。
2. 创建 **NodeFeatureDiscovery** 自定义资源(CR)。
3. 在创建 **Kataconfig** CR 时启用节点资格检查。您可以在所有 worker 节点或所选节点上运行节点资格检查。

其他资源

- 安装 Node Feature Discovery Operator

1.4. 常见术语

以下是整个文档中所使用的术语：

Sandbox

沙盒（sandbox）是一种隔离的环境，程序可以在其中运行。在沙盒中，您可以运行未经测试或不受信任的程序，而不影响到主机机器或操作系统。
在 OpenShift 沙盒容器环境中，沙盒通过使用虚拟化在不同的内核中运行工作负载来实现，从而增强了对在同一主机上运行的多个工作负载之间的交互的控制。

Pod

pod 是继承自 Kubernetes 和 OpenShift Container Platform 的构造。它代表了可以部署容器的资源。容器在 pod 内运行，pod 用于指定可以在多个容器之间共享的资源。
在 OpenShift 沙盒容器上下文中，pod 被实施为一个虚拟机。多个容器可以在同一虚拟机上在同一 pod 中运行。

OpenShift 沙盒容器 Operator

Operator 是一个软件组件，可自动执行一般需要人工在系统上执行的操作。
OpenShift 沙盒容器 Operator 的任务是管理集群上沙盒容器的生命周期。您可以使用 OpenShift 沙盒容器 Operator 来执行任务，如安装和删除沙盒容器、软件更新和状态监控。

Kata 容器

Kata 容器是一个上游核心项目，用于构建 OpenShift 沙盒容器。OpenShift 沙盒容器将 Kata 容器与 OpenShift Container Platform 集成。

KataConfig

KataConfig 对象代表沙盒容器的配置。它们存储有关集群状态的信息，如部署软件的节点。

运行时类

RuntimeClass 对象用于描述可以使用哪个运行时来运行给定工作负载。OpenShift 沙盒容器 Operator 安装和部署了名为 **kata** 的运行时类。运行时类包含有关运行时的信息，用于描述运行时需要运行的资源，如 pod 开销。

对等（peer）pod

OpenShift 沙盒容器中的对等 pod 扩展标准 pod 的概念。与标准沙盒容器不同，在 worker 节点本身上创建虚拟机，在对等 pod 中，虚拟机会使用任何支持的虚拟机监控程序或云供应商 API 通过远程 hypervisor 创建。对等 pod 作为 worker 节点上的常规 pod，其对应的虚拟机在其他位置运行。虚拟机的远程位置对用户是透明的，并由 pod 规格中运行时类指定。对等 pod 设计对嵌套虚拟化的需求。

1.5. OPENSHIFT 沙盒容器 OPERATOR

OpenShift 沙盒容器 Operator 封装了来自 Kata 容器的所有组件。它管理安装、生命周期和配置任务。

OpenShift 沙盒容器 Operator 以 Operator 捆绑包格式 打包为两个容器镜像：

- 捆绑包镜像包含元数据，这是使 operator OLM 就绪所必需的。
- 第二个容器镜像包含监控和管理 **KataConfig** 资源的实际控制器。

OpenShift 沙盒容器 Operator 基于 Red Hat Enterprise Linux CoreOS（RHCOS）扩展概念。RHCOS 扩展是安装可选 OpenShift Container Platform 软件的机制。OpenShift 沙盒容器 Operator 使用此机制在集群中部署沙盒容器。

沙盒容器 RHCOS 扩展包含用于 Kata、QEMU 及其依赖项的 RPM。您可以使用 Machine Config Operator 提供的 **MachineConfig** 资源启用它们。

其他资源

- 为 RHCOS 添加扩展

1.6. OPENSHIFT VIRTUALIZATION

您可以使用 OpenShift Virtualization 在集群中部署 OpenShift 沙盒容器。

要同时运行 OpenShift Virtualization 和 OpenShift 沙盒容器，您的虚拟机必须可实时迁移，以便它们不会阻止节点重启。详情请参阅 OpenShift Virtualization 文档中的关于实时迁移 的内容。

1.7. 存储注意事项

1.7.1. 块卷支持

OpenShift Container Platform 可以静态置备原始块卷。这些卷没有文件系统。对于可以直接写入磁盘或者实现其自己的存储服务的应用程序来说，使用它可以获得性能优势。

您可以将本地块设备用作 OpenShift 沙盒容器的持久性卷(PV)存储。此块设备可以使用 Local Storage Operator (LSO)来置备。

默认情况下，OpenShift Container Platform 中不会安装 Local Storage Operator。有关安装说明，请参阅安装 Local Storage Operator。

OpenShift 沙盒容器的原始块卷可以通过在 PV 规格中指定 **volumeMode: Block** 来置备。

块卷示例

```
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. 需要把 **volumeMode** 设置为 **Block** 来代表这个 PV 是一个原始块卷。
2. 使用到 **LocalVolume** 资源 **by-id** 的实际本地磁盘文件路径替换这个值。当置备程序已被成功部署时，会为这些本地磁盘创建 PV。您还必须使用此路径在部署 OpenShift 沙盒容器时标记使用块设备的节点。

1.8. FIPS 合规性

OpenShift Container Platform 是为联邦信息处理标准(FIPS) 140-2 和 140-3 设计的。当以 FIPS 模式运行 Red Hat Enterprise Linux (RHEL) 或 Red Hat Enterprise Linux CoreOS (RHCOS)时，OpenShift Container Platform 核心组件使用 RHEL 加密库，在 **x86_64**、**ppc64le**、**s390x** 架构上提交给 NIST 的 FIPS 140-2/140-3 Validation。

有关 NIST 验证程序的更多信息，请参阅加密模块验证程序。有关为验证提交的 RHEL 加密库的单独版本的最新 NIST 状态，请参阅 Compliance Activities 和 Government Standards。

OpenShift 沙盒容器可以在启用了 FIPS 的集群中使用。

在 FIPS 模式下运行时，OpenShift 沙盒容器组件、虚拟机和虚拟机镜像会根据 FIPS 进行调整。

注意

OpenShift 沙盒容器的 FIPS 合规性只适用于 **kata** 运行时类。对等 pod 运行时类 **kata-remote** 尚未被完全支持，且还没有为 FIPS 合规性进行测试。

FIPS 合规性是高安全性环境中所需的最重要的组件之一，可确保节点上只允许使用支持的加密技术。

重要

只有在 **x86_64** 架构中的 OpenShift Container Platform 部署支持 FIPS 验证的/Modules in Process 加密库。

要了解红帽对 OpenShift Container Platform 合规框架的观点，请参阅 OpenShift 安全性指南手册中的“风险管理和法规就绪状态”一章。

第 2 章 在裸机上部署工作负载

您可以使用 worker 节点上安装的 Red Hat Enterprise Linux CoreOS (RHCOS)在内部裸机服务器上部署 OpenShift 沙盒容器工作负载。

注意

- 不支持 RHEL 节点。
- 不支持嵌套虚拟化。

您可以使用任何安装方法，包括 用户置备的、安装程序置备的 或 Assisted Installer 来部署集群。

您还可以在 Amazon Web Services (AWS)裸机实例上安装 OpenShift 沙盒容器。不支持由其他云提供商提供的裸机实例。

部署工作流

您可以通过执行以下步骤部署 OpenShift 沙盒容器工作负载：

1. 准备您的环境。
2. 创建 **KataConfig** 自定义资源。
3. 将您的工作负载对象配置为使用 **kata** 运行时类。

2.1. 准备您的环境

执行以下步骤准备您的环境：

1. 确保集群有足够的资源。
2. 安装 OpenShift 沙盒容器 Operator。
3. 可选：配置节点 资格 检查以确保 worker 节点支持 OpenShift 沙盒容器：
   a. 安装 Node Feature Discovery (NFD) Operator。详情请参阅 NFD Operator 文档。
   b. 创建 **NodeFeatureDiscovery** 自定义资源(CR)以定义 NFD Operator 检查的节点配置参数。

2.1.1. 资源要求

OpenShift 沙盒容器允许用户在沙盒运行时 (Kata) 中的 OpenShift Container Platform 集群中运行工作负载。每个 pod 由一个虚拟机（VM）表示。每个虚拟机都在 QEMU 进程中运行，并托管一个 **kata-agent** 进程，它充当管理容器工作负载的监管程序，以及这些容器中运行的进程。两个额外的进程会增加开销：

- **containerd-shim-kata-v2** 用于与 pod 通信。
- **virtiofsd** 代表客户机处理主机文件系统访问。

每个虚拟机都配置有默认内存量。对于明确请求内存的容器，额外的内存会被热插到虚拟机中。

在没有内存资源的情况下运行的容器会消耗可用内存，直到虚拟机使用的总内存达到默认分配。客户机及其 I/O 缓冲区也消耗内存。

如果容器被授予特定数量的内存，那么该内存会在容器启动前热插到虚拟机中。

当指定内存限制时，如果消耗的内存超过限制，工作负载将被终止。如果没有指定内存限制，则虚拟机中运行的内核可能会耗尽内存。如果内核内存不足，它可能会终止虚拟机上的其他进程。

默认内存大小

下表列出了资源分配的一些默认值。

资源	value
默认为虚拟机分配的内存	2Gi
启动时客户机 Linux 内核内存使用	~110Mi
QEMU 进程使用的内存（虚拟机内存除外）	~30Mi
virtiofsd 进程使用的内存（虚拟机 I/O 缓冲区除外）	~10Mi
containerd-shim-kata-v2 进程使用的内存	~20Mi
在 Fedora 上运行 **dnf install** 后的文件缓冲区缓存数据	~300Mi* [1]

文件缓冲区会出现并在多个位置考虑：

- 在客户机中它被显示为文件缓冲缓存。
- 在映射允许的用户空间文件 I/O 操作的 **virtiofsd** 守护进程中。
- 在 QEMU 进程中作为客户机内存。

注意

内存使用率指标正确考虑内存用量总量，该指标仅计算该内存一次。

Pod 开销描述了节点上 pod 使用的系统资源量。您可以使用 **oc describe runtimeclass kata** 获取 Kata 运行时的当前 pod 开销，如下所示。

Example

```
$ oc describe runtimeclass kata
```

输出示例

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

您可以通过更改 **RuntimeClass** 的 **spec.overhead** 字段来更改 pod 开销。例如，如果您为容器运行的配置消耗 QEMU 进程和客户机内核数据的 350Mi 内存，您可以更改 **RuntimeClass** 开销来满足您的需要。

注意

红帽支持指定的默认开销值。不支持更改默认开销值，这可能会导致技术问题。

在客户机中执行任何类型的文件系统 I/O 时，将在客户机内核中分配文件缓冲区。文件缓冲区也在主机上的 QEMU 进程以及 **virtiofsd** 进程中映射。

例如，如果您在客户机中使用 300Mi 文件缓冲区缓存，QEMU 和 **virtiofsd** 都显示使用 300Mi 额外内存。但是，所有三种情况下都使用相同的内存。因此，内存使用总量仅为 300Mi，映射在三个不同的位置。报告内存使用率指标时，会正确计算。

2.1.2. 安装 OpenShift 沙盒容器 Operator

您可以使用 OpenShift Container Platform Web 控制台或命令行界面(CLI)安装 OpenShift 沙盒容器 Operator。

2.1.2.1. 使用 Web 控制台安装 Operator

您可以使用 Red Hat OpenShift Container Platform Web 控制台安装 OpenShift 沙盒容器 Operator。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 在 OpenShift Container Platform Web 控制台中导航至 **Operators → OperatorHub**。
2. 在 **Filter by keyword** 字段中，输入 **OpenShift sandboxed containers**。
3. 选择 **OpenShift 沙盒容器 Operator** 标题并点 **Install**。
4. 在 **Install Operator** 页面中，从可用 **Update Channel** 选项列表中选择 **stable**。
5. 验证为 **Installed Namespace** 选择了 **Operator recommended Namespace**。这会在 **openshift-sandboxed-containers-operator** 命名空间中安装 Operator。如果此命名空间尚不存在，则会自动创建。

   注意

   尝试在 **openshift-sandboxed-containers-operator** 以外的命名空间中安装 OpenShift 沙盒容器 Operator 会导致安装失败。

6. 验证是否为 **Approval Strategy** 选择了 **Automatic**。**Automatic** 是默认值，当有新的 z-stream 发行版本可用时，自动启用对 OpenShift 沙盒容器的自动更新。
7. 点 **Install**。

OpenShift 沙盒容器 Operator 现已安装在集群中。

验证

1. 导航到 **Operators → Installed Operators**。
2. 验证 OpenShift 沙盒容器 Operator 是否已显示。

其他资源

- 在受限网络中使用 Operator Lifecycle Manager。
- 为断开连接的环境 在 Operator Lifecycle Manager 中配置代理支持。

2.1.2.2. 使用 CLI 安装 Operator

您可以使用 CLI 安装 OpenShift 沙盒容器 Operator。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您可以使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 创建 **Namespace.yaml** 清单文件：

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

2. 运行以下命令创建命名空间：

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

3. 创建 **OperatorGroup.yaml** 清单文件：

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

4. 运行以下命令来创建 operator 组：

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

5. 创建 **Subscription.yaml** 清单文件：

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

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

6. 运行以下命令来创建订阅：

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

OpenShift 沙盒容器 Operator 现已安装在集群中。

验证

- 运行以下命令确保 Operator 已正确安装：

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

输出示例

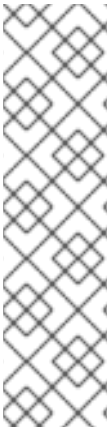
```
NAME                                   DISPLAY                                   VERSION    REPLACES
PHASE
openshift-sandboxed-containers   openshift-sandboxed-containers-operator  1.6.0    1.5.3
Succeeded
```

2.1.2.3. 其他资源

- 在受限网络中使用 Operator Lifecycle Manager
- 为断开连接的环境 在 Operator Lifecycle Manager 中配置代理支持

2.1.3. 创建 NodeFeatureDiscovery CR

您可以创建一个 **NodeFeatureDiscovery** 自定义资源(CR)来定义 Node Feature Discovery (NFD) Operator 检查的配置参数，以确定 worker 节点可以支持 OpenShift 沙盒容器。



注意

要仅在您了解的所选 worker 节点上安装 **kata** 运行时，请将 **feature.node.kubernetes.io/runtime.kata=true** 标签应用到所选节点，并在 **KataConfig** CR 中设置 **checkNodeEligibility: true**。

要在所有 worker 节点上安装 **kata** 运行时，请在 **KataConfig** CR 中设置 **checkNodeEligibility: false**。

在这两种情况下，您不需要创建 **NodeFeatureDiscovery** CR。如果您确定节点有资格运行 OpenShift 沙盒容器，则应仅应用 **feature.node.kubernetes.io/runtime.kata=true** 标签。

以下流程将 **feature.node.kubernetes.io/runtime.kata=true** 标签应用到所有有资格的节点，并将 **KataConfig** 资源配置为检查节点资格。

先决条件

- 已安装 NFD Operator。

流程

1. 根据以下示例创建 **nfd.yaml** 清单文件：

```
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. 创建 **NodeFeatureDiscovery** CR：

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

NodeFeatureDiscovery CR 将 **feature.node.kubernetes.io/runtime.kata=true** 标签应用到所有合格的 worker 节点。

1. 根据以下示例创建 **kata-config.yaml** 清单文件：

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

2. 创建 **KataConfig** CR：

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

验证

- 验证集群中是否应用了正确的标签：

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

输出示例

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

2.2. 使用 WEB 控制台部署工作负载

您可以使用 Web 控制台部署 OpenShift 沙盒容器工作负载。

2.2.1. 创建 KataConfig 自定义资源

您必须创建一个 **KataConfig** 自定义资源(CR)，以便在 worker 节点上安装 **kata** 作为 **RuntimeClass**。

Kata 运行时类默认安装在所有 worker 节点上。如果只想在特定节点上安装 **kata**，您可以向这些节点添加标签，然后在 **KataConfig** CR 中定义该标签。

OpenShift 沙盒容器将 **kata** *作为集群中的辅助* 可选运行时安装，而不是作为主要运行时。

重要

创建 **KataConfig** CR 会自动重启 worker 节点。重启可能需要 10 到 60 分钟。以下因素可能会增加重启时间：

- 带有更多 worker 节点的大型 OpenShift Container Platform 部署。
- 激活 BIOS 和 Diagnostics 实用程序。
- 在硬盘而不是 SSD 上部署。
- 在物理节点上部署，如裸机，而不是在虚拟节点上部署。
- CPU 和网络较慢。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 可选：如果要启用节点资格检查，已安装了 Node Feature Discovery Operator。

流程

1. 在 OpenShift Container Platform web 控制台中导航至 **Operators → Installed Operators**。
2. 选择 OpenShift 沙盒容器 Operator。
3. 在 **KataConfig** 选项卡中，点 **Create KataConfig**。
4. 输入以下详情：
   - **Name**: 可选：默认名称为 **example-kataconfig**。
   - **标签** ：可选：输入任何相关的、识别到 **KataConfig** 资源的属性。每个标签代表一个键值对。

- **checkNodeEligibility**: 可选：选择使用 Node Feature Discovery Operator (NFD)来检测节点资格。
- **KataConfigPoolSelector**。可选： 要在所选节点上安装 **kata**，请在所选节点上为标签添加匹配表达式：
  a. 展开 **kataConfigPoolSelector** 区域。
  b. 在 **kataConfigPoolSelector** 区域中，展开 **matchExpressions**。这是标签选择器要求列表。
  c. 点 **Add matchExpressions**。
  d. 在 **Key** 字段中，输入选择器应用到的标签键。
  e. 在 **Operator** 字段中，输入键与标签值的关系。有效的运算符为 **In**、**NotIn**、**Exists** 和 **DoesNotExist**。
  f. 展开 **Values** 区域，然后点 **Add value**。
  g. 在 **Value** 字段中，为 **key** 标签值输入 **true** 或 **false**。
- **loglevel** ：定义使用 **kata** 运行时类为节点检索的日志数据级别。

5. 点 **Create**。**KataConfig** CR 会被创建并在 worker 节点上安装 **kata** 运行时类。
   在验证安装前，等待 **kata** 安装完成，以及 worker 节点重新引导。

验证

1. 在 **KataConfig** 选项卡中，点 **KataConfig** CR 查看其详情。
2. 点 **YAML** 选项卡查看 **status** 小节。
   status 小节包含 **conditions** 和 **kataNodes** 键。**status.kataNodes** 的值是一个节点数组，每个节点都列出处于 **kata** 安装的特定状态的节点。每次有更新时都会出现一条消息。
3. 点 **Reload** 以刷新 YAML。
   当 **status.kataNodes** 数组中的所有 worker 都会显示 **installed** 和 **conditions.InProgress: False** 时，没有指定的原因，则会在集群中安装 **kata**。

详情请参阅 KataConfig 状态信息。

2.2.2. 配置工作负载对象

您可以通过将 **kata** 配置为以下 pod 模板对象的运行时类来部署 OpenShift 沙盒容器工作负载：

- **Pod** 对象
- **ReplicaSet** 对象
- **ReplicationController** 对象
- **StatefulSet** 对象
- **Deployment** 对象
- **deploymentConfig** 对象

重要

不要在 **openshift-sandboxed-containers-operator** 命名空间中部署工作负载。为这些资源创建一个专用命名空间。

先决条件

- 您已为供应商创建了 secret 对象。
- 您已为供应商创建了配置映射。
- 您已创建了 **KataConfig** 自定义资源 (CR)。

流程

1. 在 OpenShift Container Platform Web 控制台中，导航到 **Workloads** → workload type，如 **Pods**。
2. 在工作负载类型页面中，点对象查看其详情。
3. 点 **YAML** 标签。
4. 将 **spec.runtimeClassName: kata** 添加到每个 pod 模板工作负载对象的清单中，如下例所示：

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

OpenShift Container Platform 创建工作负载对象并开始调度它。

验证

- 检查 pod 模板对象的 **spec.runtimeClassName** 字段。如果值为 **kata**，则工作负载在 OpenShift 沙盒容器中运行，使用对等 pod。

2.3. 使用命令行部署工作负载

您可以使用命令行部署 OpenShift 沙盒容器工作负载。

2.3.1. 可选：使用 Local Storage Operator 置备本地块卷

OpenShift 沙盒容器的本地块卷可以使用 Local Storage Operator (LSO)来置备。本地卷置备程序会在定义的资源中指定的路径上查找任何块设备。

先决条件

- 安装了 Local Storage Operator。
- 您有一个满足以下条件的本地磁盘：
  - 它附加到一个节点。

- 它尚未挂载。
- 它不包含分区。

流程

1. 创建本地卷资源。此资源必须定义本地卷的节点和路径。

> **注意**
>
> 不要在同一设备中使用不同的存储类名称。这样做可创建多个持久性卷 (PV)。

例如：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. 安装了 Local Storage Operator 的命名空间。
2. 可选：包含附加了本地存储卷的节点列表的节点选择器。本例使用从 **oc get node** 获取的节点主机名。如果没有定义值，则 Local Storage Operator 会尝试在所有可用节点上查找匹配的磁盘。
3. 创建持久性卷对象时使用的存储类的名称。
4. 此设置定义是否调用 **wipefs**，它会删除分区表签名（魔法字符串），使磁盘准备好用于 Local Storage Operator 置备。除了签名外，没有其它数据会被清除。默认为 "false"（不调用 **wipefs**）。当在需要重新使用的磁盘中，将 **forceWipeDevicesAndDestroyAllData** 设置为 "true" 很有用。在这些情况下，将此字段设置为 true 可消除管理员手动擦除磁盘的需要。
5. 包含要从中选择的本地存储设备列表的路径。在块设备上部署沙盒容器节点时，您必须使用此路径。
6. 使用到 **LocalVolume** 资源 **by-id** 的实际本地磁盘文件路径（如 **/dev/disk/by-id/wwn**）替换这个值。当置备程序已被成功部署时，会为这些本地磁盘创建 PV。

这个值。当置备程序已被成功部署时，会为这些本地磁盘创建 PV。

2. 在 OpenShift Container Platform 集群中创建本地卷资源。指定您刚才创建的文件：

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

3. 验证置备程序是否已创建并创建了相应的守护进程集：

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

输出示例

```
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
```

请注意 **所需的** **和当前的** 守护进程设定进程数。**所需的** 数量为 **0** 表示标签选择器无效。

4. 验证持久性卷是否已创建：

```
$ oc get pv
```

输出示例

```
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
```

重要

编辑 **LocalVolume** 对象不会更改现有的持久性卷，因为这样做可能会导致破坏性操作。

2.3.2. 可选：在块设备上部署节点

如果为 OpenShift 沙盒容器置备本地块卷，您可以选择在定义的卷资源中指定的路径上部署节点。

先决条件

- 已使用 Local Storage Operator 置备块设备

流程

- 对您要使用块设备部署的每个节点运行以下命令：

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

+ **/path/to/device** 必须是您在创建本地存储资源时定义的相同路径。

+ .输出示例

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

2.3.3. 创建 KataConfig 自定义资源

您必须创建一个 **KataConfig** 自定义资源(CR)来作为 worker 节点上的运行时类安装 **kata**。

创建 **KataConfig** CR 会触发 OpenShift 沙盒容器 Operator 来执行以下操作：

- 在 RHCOS 节点上安装所需的 RHCOS 扩展，如 QEMU 和 **kata-containers**。
- 确保 CRI-O 运行时配置了正确的运行时处理程序。
- 使用默认配置创建一个名为 **kata** 的 **RuntimeClass** CR。这可让用户在 **RuntimeClassName** 字段中引用 CR 将工作负载配置为使用 **kata** 作为运行时。此 CR 也指定运行时的资源开销。

OpenShift 沙盒容器将 **kata** *作为集群中的辅助* 可选运行时安装，而不是作为主要运行时。

重要

创建 **KataConfig** CR 会自动重启 worker 节点。重启可能需要 10 到 60 分钟。妨碍重启时间的因素如下：

- 带有更多 worker 节点的大型 OpenShift Container Platform 部署。
- 激活 BIOS 和 Diagnostics 实用程序。
- 在硬盘而不是 SSD 上部署。
- 在物理节点上部署，如裸机，而不是在虚拟节点上部署。
- CPU 和网络较慢。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 可选：如果要启用节点资格检查，已安装了 Node Feature Discovery Operator。

流程

1. 根据以下示例创建 **cluster-kataconfig.yaml** 清单文件：

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

(1) 可选：将'checkNodeEligibility' 设置为 **true** 以运行节点资格检查。

2. 可选： 要在所选节点上安装 **kata**，请按照以下示例指定节点标签：

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

(1) 指定所选节点的标签。

3. 创建 **KataConfig** CR：

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

新的 **KataConfig** CR 会被创建，并在 worker 节点上作为运行时类安装 **kata**。

在验证安装前，等待 **kata** 安装完成，以及 worker 节点重新引导。

验证

- 运行以下命令监控安装进度：

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

当安装 **kataNodes** 下的所有 worker 的状态并且条件 **InProgress** 为 **False** 时，如果没有指定原因，则会在集群中安装 **kata**。

详情请参阅 **KataConfig 状态信息**。

2.3.4. 可选：修改 pod 开销

Pod 开销描述了节点上 pod 使用的系统资源量。您可以通过更改 RuntimeClass 自定义资源的 spec.overhead 字段来修改 pod 开销。例如，如果您为容器运行的配置消耗 QEMU 进程和客户机内核数据的 350Mi 内存，您可以更改 RuntimeClass 开销来满足您的需要。

在客户机中执行任何类型的文件系统 I/O 时，将在客户机内核中分配文件缓冲区。文件缓冲区也在主机上的 QEMU 进程以及 virtiofsd 进程中映射。

例如，如果您在客户机中使用 300Mi 文件缓冲区缓存，QEMU 和 virtiofsd 都显示使用 300Mi 额外内存。但是，所有三种情况下都使用相同的内存。因此，内存使用总量仅为 300Mi，映射在三个不同的位置。报告内存使用率指标时，会正确计算。

注意

红帽支持默认值。不支持更改默认开销值，这可能会导致技术问题。

流程

1. 运行以下命令来获取 RuntimeClass 对象：

```
$ oc describe runtimeclass kata
```

2. 更新 overhead.podFixed.memory 和 cpu 值，保存为 RuntimeClass.yaml ：

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

2.3.5. 配置工作负载对象

您可以通过将 kata 配置为以下 pod 模板对象的运行时类来部署 OpenShift 沙盒容器工作负载：

- Pod 对象

- ReplicaSet 对象
- ReplicationController 对象
- StatefulSet 对象
- Deployment 对象
- deploymentConfig 对象

重要

不要在 openshift-sandboxed-containers-operator 命名空间中部署工作负载。为这些资源创建一个专用命名空间。

先决条件

- 您已为供应商创建了 secret 对象。
- 您已为供应商创建了配置映射。
- 您已创建了 KataConfig 自定义资源 (CR)。

流程

1. 将 spec.runtimeClassName: kata 添加到每个 pod 模板工作负载对象的清单中，如下例所示：

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

OpenShift Container Platform 创建工作负载对象并开始调度它。

验证

- 检查 pod 模板对象的 spec.runtimeClassName 字段。如果值为 kata，则工作负载在 OpenShift 沙盒容器中运行，使用对等 pod。

第 3 章 在公共云中部署工作负载

您可以在 AWS Cloud Computing Services 和 Microsoft Azure Cloud Computing Services 上部署 OpenShift 沙盒容器工作负载。

集群要求

- 已安装 Red Hat OpenShift Container Platform 4.13 或更高版本。
- 您的集群至少有一个 worker 节点。

3.1. 在 AWS 上部署工作负载

您可以使用 OpenShift Container Platform Web 控制台或命令行界面(CLI)在 AWS Cloud Computing Services 上部署 OpenShift 沙盒容器工作负载。

部署工作流

1. 启用端口。
2. 为 AWS 创建 secret。
3. 为 AWS 创建配置映射。
4. 创建 KataConfig 自定义资源。
5. 可选：修改每个节点的对等 pod VM 限制。
6. 将您的工作负载对象配置为使用 kata-remote 运行时类。

3.1.1. 准备您的环境

执行以下步骤准备您的环境：

1. 确保集群有足够的资源。
2. 安装 OpenShift 沙盒容器 Operator。
3. 启用端口 15150 和 9000，以允许内部与对等 pod 通信。

3.1.1.1. 资源要求

对等 pod 虚拟机(VM)需要位于两个位置的资源：

- worker 节点。worker 节点存储元数据、Kata shim 资源(containerd-shim-kata-v2)、remote-hypervisor 资源(cloud-api-adaptor)，以及 worker 节点和对等 pod 虚拟机之间的隧道设置。
- 云实例。这是在云中运行的实际对等 pod 虚拟机。

Kubernetes worker 节点中使用的 CPU 和内存资源由 RuntimeClass (kata-remote)定义中包含的 pod 开销 处理，用于创建对等 pod。

在云中运行的对等 pod 虚拟机总数定义为 Kubernetes 节点扩展资源。这个限制是每个节点，并由 peerpodConfig 自定义资源(CR)中的 limit 属性设置。

在创建 kataConfig CR 并启用对等 pod 时，名为 peerpodconfig-openshift 的 peerpodConfig CR 会被创建，位于 openshift-sandboxed-containers-operator 命名空间中。

以下 peerpodConfig CR 示例显示默认的 spec 值：

```
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. 默认限制为每个节点 10 个虚拟机。

扩展资源名为 kata.peerpods.io/vm，并允许 Kubernetes 调度程序处理容量跟踪和核算。

您可以根据环境要求编辑每个节点的限制。如需更多信息，请参阅"修改对等 pod 中每个节点的虚拟机限制"。

变异 Webhook 将扩展的资源 kata.peerpods.io/vm 添加到 pod 规格中。如果存在，它还会从 pod 规格中删除任何特定于资源的条目。这可让 Kubernetes 调度程序考虑这些扩展资源，确保仅在资源可用时调度对等 pod。

变异 Webhook 修改 Kubernetes pod，如下所示：

- 变异 Webhook 会检查 pod 是否有预期的 RuntimeClassName 值，在 TARGET_RUNTIME_CLASS 环境变量中指定。如果 pod 规格中的值与 TARGET_RUNTIME_CLASS 的值不匹配，则 Webhook 会在不修改 pod 的情况下退出。
- 如果 RuntimeClassName 值匹配，webhook 会对 pod 规格进行以下更改：
  1. Webhook 从 pod 中所有容器和 init 容器的 resources 字段中删除每个资源规格。
  2. Webhook 通过修改 pod 中第一个容器的 resources 字段，将扩展资源 (kata.peerpods.io/vm)添加到 spec。Kubernetes 调度程序使用扩展资源 kata.peerpods.io/vm 用于核算目的。

注意

变异 Webhook 排除 OpenShift Container Platform 中的特定系统命名空间。如果在这些系统命名空间中创建了对等 pod，则使用 Kubernetes 扩展资源的资源核算不起作用，除非 pod spec 包含扩展资源。

作为最佳实践，定义集群范围的策略，仅允许在特定命名空间中创建对等 pod。

3.1.1.2. 为 AWS 启用端口

您必须启用端口 15150 和 9000，以允许内部与 AWS 上运行的对等 pod 通信。

先决条件

- 已安装 OpenShift 沙盒容器 Operator。
- 已安装 AWS 命令行工具。
- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 登录您的 OpenShift Container Platform 集群并检索实例 ID：

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

2. 检索 AWS 区域：

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

3. 检索安全组 ID，并将其存储在阵列中：

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

4. 对于每个安全组 ID，授权 peer pod shim 访问 kata-agent 通信，并设置对等 pod 隧道：

```
$ 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
```

现在启用这些端口。

3.1.1.3. 安装 OpenShift 沙盒容器 Operator

您可以使用 OpenShift Container Platform Web 控制台或命令行界面(CLI)安装 OpenShift 沙盒容器 Operator。

3.1.1.3.1. 使用 Web 控制台安装 Operator

您可以使用 Red Hat OpenShift Container Platform Web 控制台安装 OpenShift 沙盒容器 Operator。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 在 OpenShift Container Platform Web 控制台中导航至 Operators → OperatorHub。
2. 在 Filter by keyword 字段中，输入 OpenShift sandboxed containers。
3. 选择 OpenShift 沙盒容器 Operator 标题并点 Install。

4. 在 Install Operator 页面中，从可用 Update Channel 选项列表中选择 stable。

5. 验证为 Installed Namespace 选择了 Operator recommended Namespace。这会在 openshift-sandboxed-containers-operator 命名空间中安装 Operator。如果此命名空间尚不存在，则会自动创建。

注意

尝试在 openshift-sandboxed-containers-operator 以外的命名空间中安装 OpenShift 沙盒容器 Operator 会导致安装失败。

6. 验证是否为 Approval Strategy 选择了 Automatic。Automatic 是默认值，当有新的 z-stream 发行版本可用时，自动启用对 OpenShift 沙盒容器的自动更新。

7. 点 Install。

OpenShift 沙盒容器 Operator 现已安装在集群中。

验证

1. 导航到 Operators → Installed Operators。

2. 验证 OpenShift 沙盒容器 Operator 是否已显示。

其他资源

- 在受限网络中使用 Operator Lifecycle Manager。
- 为断开连接的环境 在 Operator Lifecycle Manager 中配置代理支持。

3.1.1.3.2. 使用 CLI 安装 Operator

您可以使用 CLI 安装 OpenShift 沙盒容器 Operator。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 创建 Namespace.yaml 清单文件：

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

2. 运行以下命令创建命名空间：

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

3. 创建 OperatorGroup.yaml 清单文件：

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

4. 运行以下命令来创建 operator 组：

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

5. 创建 Subscription.yaml 清单文件：

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

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

6. 运行以下命令来创建订阅：

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

OpenShift 沙盒容器 Operator 现已安装在集群中。

验证

- 运行以下命令确保 Operator 已正确安装：

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

输出示例

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

3.1.1.3.3. 其他资源

- 在受限网络中使用 Operator Lifecycle Manager
- 为断开连接的环境 在 Operator Lifecycle Manager 中配置代理支持

3.1.2. 使用 Web 控制台部署工作负载

您可以使用 Web 控制台部署 OpenShift 沙盒容器工作负载。

3.1.2.1. 创建 secret

您必须在 OpenShift Container Platform 集群中创建 Secret 对象。secret 存储云供应商凭证，用于创建 pod 虚拟机(VM)镜像和对等 pod 实例。默认情况下，OpenShift 沙盒容器 Operator 根据用于创建集群的凭证创建 secret。但是，您可以手动创建使用不同的凭证的 secret。

先决条件

- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY

  您可以在 AWS 控制台中生成这些值。

流程

1. 在 OpenShift Container Platform web 控制台中导航至 Operators → Installed Operators。
2. 点 OpenShift 沙盒容器 Operator 标题。
3. 单击右上角的 Import 图标(+)。
4. 在 Import YAML 窗口中，粘贴以下 YAML 清单：

```
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)
```

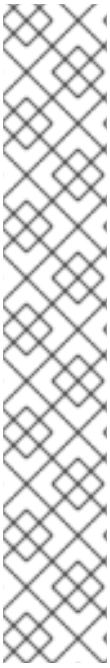
(1)

指定 AWS_ACCESS_KEY_ID 值。

2

指定 AWS_SECRET_ACCESS_KEY 值。

5. 点 Save 应用更改。



注意

如果更新 peer pod secret，您必须重启 peerpodconfig-ctrl-caa-daemon DaemonSet 来应用更改。

更新 secret 后，点 Save 应用更改。然后运行以下命令来重启 cloud-api-adaptor pod：

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

重启守护进程集会重新创建对等 pod。它不会更新现有的 pod。

验证

- 导航到 Workloads → Secrets 以查看 secret。

3.1.2.2. 创建配置映射

您必须在 OpenShift Container Platform 集群上为您的云供应商创建配置映射。

您必须设置 Amazon Machine Image (AMI) ID。在创建配置映射前，您可以检索这个值。

流程

1. 从 AWS 实例获取以下值：

a. 检索并记录实例 ID：

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

这用于检索 secret 对象的其他值。

b. 检索并记录 AWS 区域：

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

c. 检索并记录 AWS 子网 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. 检索并记录 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. 检索并记录 AWS 安全组 ID：

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

2. 在 OpenShift Container Platform web 控制台中导航至 Operators → Installed Operators。

3. 从 Operator 列表中选择 OpenShift 沙盒容器 Operator。

4.

4. 单击右上角的 Import 图标 (+)。

5. 在 Import YAML 窗口中，粘贴以下 YAML 清单：

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

(1) 定义工作负载中没有定义类型时使用的默认实例类型。

(2) 列出创建 pod 时可以指定的所有实例类型。这可让您为大型工作负载需要较少的内存和更多 CPU 或更大的实例类型的工作负载定义较小的实例类型。

(3) 可选：默认情况下，这个值会在运行 KataConfig CR 时填充，使用基于集群凭证的 AMI ID。如果您创建自己的 AMI，请指定正确的 AMI ID。

(4) 指定您检索到的 AWS_REGION 值。

(5) 指定您检索到的 AWS_SUBNET_ID 值。

(6)

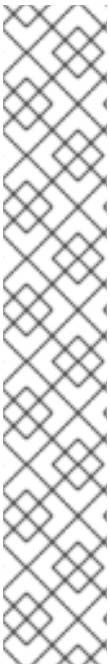
指定您检索到的 AWS_VPC_ID 值。

7

指定您检索到的 AWS_SG_IDS 值。

6. 点 Save 应用更改。

为您的云供应商创建一个配置映射。



注意

如果更新 peer pod 配置映射，您必须重启 peerpodconfig-ctrl-caa-daemon daemonset 以应用更改。

更新配置映射后，点 Save 应用更改。然后运行以下命令来重启 cloud-api-adaptor pod：

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

重启 daemonset 会重新创建对等 pod。它不会更新现有的 pod。

验证

- 导航到 Workloads → ConfigMaps 以查看新的配置映射。

3.1.2.3. 创建 KataConfig 自定义资源

您必须创建一个 KataConfig 自定义资源(CR)，以便在 worker 节点上作为 RuntimeClass 安装 kata-remote。

kata-remote 运行时类默认安装在所有 worker 节点上。如果只想在特定节点上安装 kata-remote，您可以向这些节点添加标签，然后在 KataConfig CR 中定义该标签。

OpenShift 沙盒容器将 kata-remote 安装为集群上的 *辅助* 可选运行时，而不是主运行时。

重要

创建 KataConfig CR 会自动重启 worker 节点。重启可能需要 10 到 60 分钟。以下因素可能会增加重启时间：

- 带有更多 worker 节点的大型 OpenShift Container Platform 部署。
- 激活 BIOS 和 Diagnostics 实用程序。
- 在硬盘而不是 SSD 上部署。
- 在物理节点上部署，如裸机，而不是在虚拟节点上部署。
- CPU 和网络较慢。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 在 OpenShift Container Platform web 控制台中导航至 Operators → Installed Operators。
2. 选择 OpenShift 沙盒容器 Operator。
3. 在 KataConfig 选项卡中，点 Create KataConfig。
4. 输入以下详情：

- **Name:** 可选：默认名称为 **example-kataconfig**。

- 标签 ：可选：输入任何相关的、识别到 **KataConfig** 资源的属性。每个标签代表一个键值对。

- **enablePeerPods** ：为公共云、**IBM Z®** 和 **IBM® LinuxONE** 部署选择。

- **KataConfigPoolSelector**。可选： 要在所选节点上安装 **kata-remote**，请在所选节点上安装标签的匹配表达式：

  a. 展开 **kataConfigPoolSelector** 区域。

  b. 在 **kataConfigPoolSelector** 区域中，展开 **matchExpressions**。这是标签选择器要求列表。

  c. 点 **Add matchExpressions**。

  d. 在 **Key** 字段中，输入选择器应用到的标签键。

  e. 在 **Operator** 字段中，输入键与标签值的关系。有效的运算符为 **In**、**NotIn**、**Exists** 和 **DoesNotExist**。

  f. 展开 **Values** 区域，然后点 **Add value**。

  g. 在 **Value** 字段中，为 **key** 标签值输入 **true** 或 **false**。

- **loglevel** ：定义使用 **kata-remote** 运行时类为节点检索的日志数据级别。

5. 点 **Create**。**KataConfig CR** 会被创建并在 **worker** 节点上安装 **kata-remote** 运行时类。

在验证安装前，等待 kata-remote 安装完成，以及 worker 节点重新引导。

验证

1. 在 KataConfig 选项卡中，点 KataConfig CR 查看其详情。

2. 点 YAML 选项卡查看 status 小节。

   status 小节包含 conditions 和 kataNodes 键。status.kataNodes 的值是一个节点数组，每个节点都列出处于 kata-remote 安装的特定状态的节点。每次有更新时都会出现一条消息。

3. 点 Reload 以刷新 YAML。

   当 status.kataNodes 数组中的所有 worker 都会显示 installed 和 conditions.InProgress: False 时，集群中会安装 kata-remote。

详情请参阅 KataConfig 状态信息。

3.1.2.3.1. 可选：验证 pod 虚拟机镜像

在集群中安装 kata-remote 后，OpenShift 沙盒容器 Operator 会创建一个 pod 虚拟机镜像，用于创建对等 pod。此过程可能需要很长时间，因为镜像是在云实例上创建的。您可以通过检查您为云供应商创建的配置映射来验证 pod 虚拟机镜像是否已成功创建。

流程

1. 进入 Workloads → ConfigMaps。

2. 点供应商配置映射查看其详情。

3. 点 YAML 标签。

4. 检查 YAML 文件 的状态 小节。

如果 PODVM_AMI_ID 参数被填充，则 pod 虚拟机镜像已创建成功。

故障排除

1. 运行以下命令来检索事件日志：

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

2. 运行以下命令来检索作业日志：

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

如果您无法解决这个问题，请提交红帽支持问题单并附加这两个日志的输出。

3.1.2.4. 可选：修改每个节点的对等 pod 虚拟机数量

您可以通过编辑 peerpodConfig 自定义资源(CR)来更改每个节点对等 pod 虚拟机(VM)的限制。

流程

1. 运行以下命令检查当前的限制：

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

2. 运行以下命令修改 peerpodConfig CR 的 limit 属性：

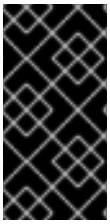
```
$ oc patch peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-
containers-operator \
--type merge --patch '{"spec":{"limit":"<value>"}}' (1)
```

(1) 将 <value> 替换为您要定义的限制。

3.1.2.5. 配置工作负载对象

您可以通过将 kata-remote 配置为以下 pod 模板对象的运行时类来部署 OpenShift 沙盒容器工作负载：

- Pod 对象
- ReplicaSet 对象
- ReplicationController 对象
- StatefulSet 对象
- Deployment 对象
- deploymentConfig 对象



重要

不要在 openshift-sandboxed-containers-operator 命名空间中部署工作负载。为这些资源创建一个专用命名空间。

您可以通过在 YAML 文件中添加注解，定义工作负载是否使用配置映射中定义的默认实例类型部署。

如果您不想手动定义实例类型，您可以添加注解来使用自动实例类型，具体取决于可用内存。

先决条件

- 您已为供应商创建了 secret 对象。
- 您已为供应商创建了配置映射。

- 您已创建了 KataConfig 自定义资源 (CR)。

流程

1. 在 OpenShift Container Platform Web 控制台中，导航到 Workloads → workload type，如 Pods。

2. 在工作负载类型页面中，点对象查看其详情。

3. 点 YAML 标签。

4. 将 spec.runtimeClassName: kata-remote 添加到每个 pod 模板工作负载对象的清单中，如下例所示：

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

5. 向 pod 模板对象添加注解，以使用手动定义的实例类型或自动实例类型：

   - 要使用手动定义的实例类型，请添加以下注解：

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

(1) 指定配置映射中定义的实例类型。

   - 要使用自动实例类型，请添加以下注解：

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

定义可供工作负载使用的内存量。工作负载将根据可用内存量在自动实例类型上运行。

6. 点 Save 应用更改。

OpenShift Container Platform 创建工作负载对象并开始调度它。

验证

- 检查 pod 模板对象的 spec.runtimeClassName 字段。如果值为 kata-remote，则工作负载在 OpenShift 沙盒容器上运行，使用对等 pod。

3.1.3. 使用命令行部署工作负载

您可以使用命令行部署 OpenShift 沙盒容器工作负载。

3.1.3.1. 创建 secret

您必须在 OpenShift Container Platform 集群中创建 Secret 对象。secret 存储云供应商凭证，用于创建 pod 虚拟机(VM)镜像和对等 pod 实例。默认情况下，OpenShift 沙盒容器 Operator 根据用于创建集群的凭证创建 secret。但是，您可以手动创建使用不同的凭证的 secret。

先决条件

- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY

  您可以在 AWS 控制台中生成这些值。

流程

1. 根据以下示例创建 peer-pods-secret.yaml 清单文件：

```
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) 指定 AWS_ACCESS_KEY_ID 值。

(2) 指定 AWS_SECRET_ACCESS_KEY 值。

2. 通过应用清单来创建 secret 对象：

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

注意

如果更新 peer pod secret，您必须重启 peerpodconfig-ctrl-caa-daemon DaemonSet 来应用更改。

更新 secret 后，应用清单。然后运行以下命令来重启 cloud-api-adaptor pod：

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

重启守护进程集会重新创建对等 pod。它不会更新现有的 pod。

3.1.3.2. 创建配置映射

您必须在 OpenShift Container Platform 集群上为您的云供应商创建配置映射。

您必须设置 Amazon Machine Image (AMI) ID。在创建配置映射前，您可以检索这个值。

流程

1. 从 AWS 实例获取以下值：

   a. 检索并记录实例 ID：

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

   这用于检索 secret 对象的其他值。

   b. 检索并记录 AWS 区域：

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

   c. 检索并记录 AWS 子网 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. 检索并记录 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. 检索并记录 AWS 安全组 ID：

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

2. 根据以下示例创建 peer-pods-cm.yaml 清单：

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

1. 定义工作负载中没有定义类型时使用的默认实例类型。

2. 列出创建 pod 时可以指定的所有实例类型。这可让您为大型工作负载需要较少的内存和更多 CPU 或更大的实例类型的工作负载定义较小的实例类型。

3. 可选：默认情况下，这个值会在运行 KataConfig CR 时填充，使用基于集群凭证的 AMI ID。如果您创建自己的 AMI，请指定正确的 AMI ID。

4. 指定您检索到的 AWS_REGION 值。

5. 指定您检索到的 AWS_SUBNET_ID 值。

6. 指定您检索到的 AWS_VPC_ID 值。

7.

指定您检索到的 AWS_SG_IDS 值。

3. 应用清单以创建配置映射：

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

为您的云供应商创建一个配置映射。

注意

如果更新 peer pod 配置映射，您必须重启 peerpodconfig-ctrl-caa-daemon daemonset 以应用更改。

更新配置映射后，应用清单。然后运行以下命令来重启 cloud-api-adaptor pod：

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

重启 daemonset 会重新创建对等 pod。它不会更新现有的 pod。

3.1.3.3. 创建 KataConfig 自定义资源

您必须创建一个 KataConfig 自定义资源(CR)来作为 worker 节点上的运行时类安装 kata-remote。

创建 KataConfig CR 会触发 OpenShift 沙盒容器 Operator 来执行以下操作：

- 使用默认配置创建一个名为 kata-remote 的 RuntimeClass CR。这可让用户在 RuntimeClassName 字段中引用 CR 将工作负载配置为使用 kata-remote 作为运行时。此 CR 也指定运行时的资源开销。

OpenShift 沙盒容器将 kata-remote 安装为集群上的 *辅助* 可选运行时，而不是主运行时。

重要

创建 KataConfig CR 会自动重启 worker 节点。重启可能需要 10 到 60 分钟。妨碍重启时间的因素如下：

- 带有更多 worker 节点的大型 OpenShift Container Platform 部署。
- 激活 BIOS 和 Diagnostics 实用程序。
- 在硬盘而不是 SSD 上部署。
- 在物理节点上部署，如裸机，而不是在虚拟节点上部署。
- CPU 和网络较慢。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 根据以下示例创建 cluster-kataconfig.yaml 清单文件：

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

2. 可选：要在所选节点上安装 kata-remote，请根据以下示例指定节点标签：

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

```
  kataConfigPoolSelector:
    matchLabels:
      <label_key>: '<label_value>' (1)
# ...
```

(1) 指定所选节点的标签。

3. 创建 KataConfig CR：

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

新的 KataConfig CR 被创建，并在 worker 节点上作为运行时类安装 kata-remote。

在验证安装前，等待 kata-remote 安装完成，以及 worker 节点重新引导。

验证

- 运行以下命令监控安装进度：

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

安装 kataNodes 下所有 worker 的状态并且条件 InProgress 为 False 时，而不指定原因，则会在集群中安装 kata-remote。

详情请参阅 KataConfig 状态信息。

3.1.3.3.1. 可选：验证 pod 虚拟机镜像

在集群中安装 kata-remote 后，OpenShift 沙盒容器 Operator 会创建一个 pod 虚拟机镜像，用于创建对等 pod。此过程可能需要很长时间，因为镜像是在云实例上创建的。您可以通过检查您为云供应商创建的配置映射来验证 pod 虚拟机镜像是否已成功创建。

流程

1. 获取您为对等 pod 创建的配置映射：

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

2. 检查 YAML 文件 的状态 小节。

   如果 PODVM_AMI_ID 参数被填充，则 pod 虚拟机镜像已创建成功。

故障排除

1. 运行以下命令来检索事件日志：

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

2. 运行以下命令来检索作业日志：

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

如果您无法解决这个问题，请提交红帽支持问题单并附加这两个日志的输出。

3.1.3.4. 可选：修改每个节点的对等 pod 虚拟机数量

您可以通过编辑 peerpodConfig 自定义资源(CR)来更改每个节点对等 pod 虚拟机(VM)的限制。

流程

1. 运行以下命令检查当前的限制：

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

2. 运行以下命令修改 peerpodConfig CR 的 limit 属性：

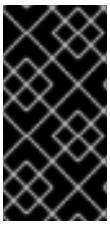
```
$ oc patch peerpodconfig peerpodconfig-openshift -n openshift-sandboxed-
containers-operator \
--type merge --patch '{"spec":{"limit":"<value>"}}' (1)
```

1 将 <value> 替换为您要定义的限制。

3.1.3.5. 配置工作负载对象

您可以通过将 kata-remote 配置为以下 pod 模板对象的运行时类来部署 OpenShift 沙盒容器工作负载：

- Pod 对象
- ReplicaSet 对象
- ReplicationController 对象
- StatefulSet 对象
- Deployment 对象
- deploymentConfig 对象



重要

不要在 openshift-sandboxed-containers-operator 命名空间中部署工作负载。为这些资源创建一个专用命名空间。

您可以通过在 YAML 文件中添加注解，定义工作负载是否使用配置映射中定义的默认实例类型部署。

如果您不想手动定义实例类型，您可以添加注解来使用自动实例类型，具体取决于可用内存。

先决条件

- 您已为供应商创建了 secret 对象。
- 您已为供应商创建了配置映射。
- 您已创建了 KataConfig 自定义资源 (CR)。

流程

1. 将 spec.runtimeClassName: kata-remote 添加到每个 pod 模板工作负载对象的清单中，如下例所示：

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

2. 向 pod 模板对象添加注解，以使用手动定义的实例类型或自动实例类型：

   - 要使用手动定义的实例类型，请添加以下注解：

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

(1) 指定配置映射中定义的实例类型。

   - 要使用自动实例类型，请添加以下注解：

```
apiVersion: v1
kind: <Pod>
metadata:
  annotations:
```

```
    io.katacontainers.config.hypervisor.default_vcpus: <vcpus>
    io.katacontainers.config.hypervisor.default_memory: <memory>
# ...
```

定义可供工作负载使用的内存量。工作负载将根据可用内存量在自动实例类型上运行。

3. 运行以下命令，将更改应用到工作负载对象：

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

OpenShift Container Platform 创建工作负载对象并开始调度它。

验证

- 检查 pod 模板对象的 spec.runtimeClassName 字段。如果值为 kata-remote，则工作负载在 OpenShift 沙盒容器上运行，使用对等 pod。

3.2. 在 AZURE 上部署工作负载

您可以使用 OpenShift Container Platform Web 控制台或命令行界面(CLI)在 Microsoft Azure Cloud Computing Services 上部署 OpenShift 沙盒容器工作负载。

部署工作流

1. 为您的 Azure 访问密钥创建 secret。
2. 创建配置映射以定义 Azure 实例大小和其他参数。
3. 创建 SSH 密钥 secret。
4. 创建 KataConfig 自定义资源。
5. 可选：修改每个节点的对等 pod VM 限制。

6. 将您的工作负载对象配置为使用 kata-remote 运行时类。

3.2.1. 准备您的环境

执行以下步骤准备您的环境：

1. 确保集群有足够的资源。
2. 安装 OpenShift 沙盒容器 Operator。

3.2.1.1. 资源要求

对等 pod 虚拟机(VM)需要位于两个位置的资源：

- worker 节点。worker 节点存储元数据、Kata shim 资源(containerd-shim-kata-v2)、remote-hypervisor 资源(cloud-api-adaptor)，以及 worker 节点和对等 pod 虚拟机之间的隧道设置。
- 云实例。这是在云中运行的实际对等 pod 虚拟机。

Kubernetes worker 节点中使用的 CPU 和内存资源由 RuntimeClass (kata-remote)定义中包含的 pod 开销 处理，用于创建对等 pod。

在云中运行的对等 pod 虚拟机总数定义为 Kubernetes 节点扩展资源。这个限制是每个节点，并由 peerpodConfig 自定义资源(CR)中的 limit 属性设置。

在创建 kataConfig CR 并启用对等 pod 时，名为 peerpodconfig-openshift 的 peerpodConfig CR 会被创建，位于 openshift-sandboxed-containers-operator 命名空间中。

以下 peerpodConfig CR 示例显示默认的 spec 值：

```
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. 默认限制为每个节点 10 个虚拟机。

扩展资源名为 kata.peerpods.io/vm，并允许 Kubernetes 调度程序处理容量跟踪和核算。

您可以根据环境要求编辑每个节点的限制。如需更多信息，请参阅"修改对等 pod 中每个节点的虚拟机限制"。

变异 Webhook 将扩展的资源 kata.peerpods.io/vm 添加到 pod 规格中。如果存在，它还会从 pod 规格中删除任何特定于资源的条目。这可让 Kubernetes 调度程序考虑这些扩展资源，确保仅在资源可用时调度对等 pod。

变异 Webhook 修改 Kubernetes pod，如下所示：

- 变异 Webhook 会检查 pod 是否有预期的 RuntimeClassName 值，在 TARGET_RUNTIME_CLASS 环境变量中指定。如果 pod 规格中的值与 TARGET_RUNTIME_CLASS 的值不匹配，则 Webhook 会在不修改 pod 的情况下退出。
- 如果 RuntimeClassName 值匹配，webhook 会对 pod 规格进行以下更改：
  1. Webhook 从 pod 中所有容器和 init 容器的 resources 字段中删除每个资源规格。
  2. Webhook 通过修改 pod 中第一个容器的 resources 字段，将扩展资源 (kata.peerpods.io/vm)添加到 spec。Kubernetes 调度程序使用扩展资源 kata.peerpods.io/vm 用于核算目的。

注意

变异 Webhook 排除 OpenShift Container Platform 中的特定系统命名空间。如果在这些系统命名空间中创建了对等 pod，则使用 Kubernetes 扩展资源的资源核算不起作用，除非 pod spec 包含扩展资源。

作为最佳实践，定义集群范围的策略，仅允许在特定命名空间中创建对等 pod。

3.2.1.2. 安装 OpenShift 沙盒容器 Operator

您可以使用 OpenShift Container Platform Web 控制台或命令行界面(CLI)安装 OpenShift 沙盒容器 Operator。

3.2.1.2.1. 使用 Web 控制台安装 Operator

您可以使用 Red Hat OpenShift Container Platform Web 控制台安装 OpenShift 沙盒容器 Operator。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 在 OpenShift Container Platform Web 控制台中导航至 Operators → OperatorHub。
2. 在 Filter by keyword 字段中，输入 OpenShift sandboxed containers。
3. 选择 OpenShift 沙盒容器 Operator 标题并点 Install。
4. 在 Install Operator 页面中，从可用 Update Channel 选项列表中选择 stable。
5. 验证为 Installed Namespace 选择了 Operator recommended Namespace。这会在 openshift-sandboxed-containers-operator 命名空间中安装 Operator。如果此命名空间尚不存在，则会自动创建。

注意

尝试在 openshift-sandboxed-containers-operator 以外的命名空间中安装 OpenShift 沙盒容器 Operator 会导致安装失败。

6. 验证是否为 Approval Strategy 选择了 Automatic。Automatic 是默认值，当有新的 z-stream 发行版本可用时，自动启用对 OpenShift 沙盒容器的自动更新。

7. 点 Install。

OpenShift 沙盒容器 Operator 现已安装在集群中。

验证

1. 导航到 Operators → Installed Operators。

2. 验证 OpenShift 沙盒容器 Operator 是否已显示。

其他资源

- 在受限网络中使用 Operator Lifecycle Manager。
- 为断开连接的环境 在 Operator Lifecycle Manager 中配置代理支持。

3.2.1.2.2. 使用 CLI 安装 Operator

您可以使用 CLI 安装 OpenShift 沙盒容器 Operator。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 创建 Namespace.yaml 清单文件：

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

2. 运行以下命令创建命名空间：

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

3. 创建 OperatorGroup.yaml 清单文件：

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

4. 运行以下命令来创建 operator 组：

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

5. 创建 Subscription.yaml 清单文件：

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

6.

5. 运行以下命令来创建订阅：

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

OpenShift 沙盒容器 Operator 现已安装在集群中。

验证

- 运行以下命令确保 Operator 已正确安装：

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

输出示例

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

3.2.1.2.3. 其他资源

- 在受限网络中使用 Operator Lifecycle Manager
- 为断开连接的环境 在 Operator Lifecycle Manager 中配置代理支持

3.2.2. 使用 Web 控制台部署工作负载

您可以使用 Web 控制台部署 OpenShift 沙盒容器工作负载。

3.2.2.1. 创建 secret

您必须在 OpenShift Container Platform 集群中创建 Secret 对象。secret 存储云供应商凭证，用于创建 pod 虚拟机(VM)镜像和对等 pod 实例。默认情况下，OpenShift 沙盒容器 Operator 根据用于创建集群的凭证创建 secret。但是，您可以手动创建使用不同的凭证的 secret。

先决条件

- 已安装并配置了 Azure CLI 工具。

流程

1. 检索 Azure 订阅 ID：

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

2. 生成 RBAC 内容。这会生成客户端 ID、客户端 secret 和租户 ID：

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

输出示例：

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

3. 记录要在 secret 对象中使用的 RBAC 输出。

4. 在 OpenShift Container Platform web 控制台中导航至 Operators → Installed Operators。

5. 点 OpenShift 沙盒容器 Operator 标题。

6. 单击右上角的 Import 图标(+)。

7. 在 Import YAML 窗口中，粘贴以下 YAML 清单：

```
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. 指定 AZURE_CLIENT_ID 值。
2. 指定 AZURE_CLIENT_SECRET 值。
3. 指定 AZURE_TENANT_ID 值。
4. 指定 AZURE_SUBSCRIPTION_ID 值。

8. 点 Save 应用更改。

注意

如果更新 peer pod secret，您必须重启 peerpodconfig-ctrl-caa-daemon DaemonSet 来应用更改。

更新 secret 后，点 Save 应用更改。然后运行以下命令来重启 cloud-api-adaptor pod：

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

重启守护进程集会重新创建对等 pod。它不会更新现有的 pod。

验证

- 导航到 Workloads → Secrets 以查看 secret。

3.2.2.2. 创建配置映射

您必须在 OpenShift Container Platform 集群上为您的云供应商创建配置映射。

流程

1. 从 Azure 实例获取以下值：

   a. 检索并记录 Azure VNet 名称：

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

   这个值用于检索 Azure 子网 ID。

   b. 检索并记录 Azure 子网 ID：

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

   c. 检索并记录 Azure 网络安全组(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\""
```

   d. 检索并记录 Azure 资源组：

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

e. 检索并记录 Azure 区域：

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

2. 在 OpenShift Container Platform web 控制台中导航至 Operators → Installed Operators。

3. 从 Operator 列表中选择 OpenShift 沙盒容器 Operator。

4. 单击右上角的 Import 图标 (+)。

5. 在 Import YAML 窗口中，粘贴以下 YAML 清单：

```
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"
  DISABLECVM: "true"
  AZURE_IMAGE_ID: "<azure_image_id>" (5)
  AZURE_REGION: "<azure_region>" (6)
  AZURE_RESOURCE_GROUP: "<azure_resource_group>" (7)
```

(1) 定义工作负载中没有定义类型时使用的默认实例大小。

(2) 列出创建 pod 时可以指定的所有实例大小。这可让您为大型工作负载需要较少的内存和更小的实例大小的工作负载定义较小的实例大小。

3. 指定您检索的 AZURE_SUBNET_ID 值。

4. 指定您检索的 AZURE_NSG_ID 值。

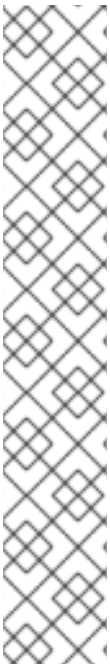
5. 可选：默认情况下，这个值会在运行 KataConfig CR 时填充，使用基于集群凭证的 Azure 镜像 ID。如果创建自己的 Azure 镜像，请指定正确的镜像 ID。

6. 指定您检索到的 AZURE_REGION 值。

7. 指定您检索的 AZURE_RESOURCE_GROUP 值。

6. 点 Save 应用更改。

为您的云供应商创建一个配置映射。



注意

如果更新 peer pod 配置映射，您必须重启 peerpodconfig-ctrl-caa-daemon daemonset 以应用更改。

更新配置映射后，点 Save 应用更改。然后运行以下命令来重启 cloud-api-adaptor pod：

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

重启 daemonset 会重新创建对等 pod。它不会更新现有的 pod。

验证

- 导航到 Workloads → ConfigMaps 以查看新的配置映射。

3.2.2.3. 创建 SSH 密钥 secret

您必须为 Azure 创建 SSH 密钥 secret 对象。

流程

1. 登录您的 OpenShift Container Platform 集群。
2. 运行以下命令来生成 SSH 密钥对：

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

3. 在 OpenShift Container Platform Web 控制台中导航至 Workloads → Secrets。
4. 在 Secrets 页面中，验证您是否位于 openshift-sandboxed-containers-operator 项目中。
5. 点 Create 并选择 Key/value secret。
6. 在 Secret name 字段中，输入 ssh-key-secret。
7. 在 Key 字段中，输入 id_rsa.pub。
8. 在 Value 字段中，粘贴您的公共 SSH 密钥。
9. 点 Create。

   SSH 密钥 secret 已创建。

10. 删除您创建的 SSH 密钥：

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

3.2.2.4. 创建 KataConfig 自定义资源

您必须创建一个 KataConfig 自定义资源(CR)，以便在 worker 节点上作为 RuntimeClass 安装 kata-remote。

kata-remote 运行时类默认安装在所有 worker 节点上。如果只想在特定节点上安装 kata-remote，您可以向这些节点添加标签，然后在 KataConfig CR 中定义该标签。

OpenShift 沙盒容器将 kata-remote 安装为集群上的 *辅助* 可选运行时，而不是主运行时。

重要

创建 KataConfig CR 会自动重启 worker 节点。重启可能需要 10 到 60 分钟。以下因素可能会增加重启时间：

- 带有更多 worker 节点的大型 OpenShift Container Platform 部署。
- 激活 BIOS 和 Diagnostics 实用程序。
- 在硬盘而不是 SSD 上部署。
- 在物理节点上部署，如裸机，而不是在虚拟节点上部署。
- CPU 和网络较慢。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 在 OpenShift Container Platform web 控制台中导航至 Operators → Installed Operators。

2. 选择 OpenShift 沙盒容器 Operator。

3. 在 KataConfig 选项卡中，点 Create KataConfig。

4. 输入以下详情：

   - Name: 可选：默认名称为 example-kataconfig。

   - 标签 ：可选：输入任何相关的、识别到 KataConfig 资源的属性。每个标签代表一个键值对。

   - enablePeerPods ：为公共云、IBM Z® 和 IBM® LinuxONE 部署选择。

   - KataConfigPoolSelector。可选： 要在所选节点上安装 kata-remote，请在所选节点上安装标签的匹配表达式：

     a. 展开 kataConfigPoolSelector 区域。

     b. 在 kataConfigPoolSelector 区域中，展开 matchExpressions。这是标签选择器要求列表。

     c. 点 Add matchExpressions。

     d. 在 Key 字段中，输入选择器应用到的标签键。

     e. 在 Operator 字段中，输入键与标签值的关系。有效的运算符为 In、NotIn、Exists 和 DoesNotExist。

f. 展开 Values 区域，然后点 Add value。

g. 在 Value 字段中，为 key 标签值输入 true 或 false。

- loglevel ：定义使用 kata-remote 运行时类为节点检索的日志数据级别。

5. 点 Create。KataConfig CR 会被创建并在 worker 节点上安装 kata-remote 运行时类。

   在验证安装前，等待 kata-remote 安装完成，以及 worker 节点重新引导。

验证

1. 在 KataConfig 选项卡中，点 KataConfig CR 查看其详情。

2. 点 YAML 选项卡查看 status 小节。

   status 小节包含 conditions 和 kataNodes 键。status.kataNodes 的值是一个节点数组，每个节点都列出处于 kata-remote 安装的特定状态的节点。每次有更新时都会出现一条消息。

3. 点 Reload 以刷新 YAML。

   当 status.kataNodes 数组中的所有 worker 都会显示 installed 和 conditions.InProgress: False 时，集群中会安装 kata-remote。

详情请参阅 KataConfig 状态信息。

3.2.2.4.1. 可选：验证 pod 虚拟机镜像

在集群中安装 kata-remote 后，OpenShift 沙盒容器 Operator 会创建一个 pod 虚拟机镜像，用于创建对等 pod。此过程可能需要很长时间，因为镜像是在云实例上创建的。您可以通过检查您为云供应商创建的配置映射来验证 pod 虚拟机镜像是否已成功创建。

流程

1. 进入 Workloads → ConfigMaps。

2. 点供应商配置映射查看其详情。

3. 点 YAML 标签。

4. 检查 YAML 文件 的状态 小节。

   如果 AZURE_IMAGE_ID 参数被填充，则 pod 虚拟机镜像已被成功创建。

故障排除

1. 运行以下命令来检索事件日志：

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

2. 运行以下命令来检索作业日志：

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

如果您无法解决这个问题，请提交红帽支持问题单并附加这两个日志的输出。

3.2.2.5. 可选：修改每个节点的对等 pod 虚拟机数量

您可以通过编辑 peerpodConfig 自定义资源(CR)来更改每个节点对等 pod 虚拟机(VM)的限制。

流程

1. 运行以下命令检查当前的限制：

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

2. 运行以下命令修改 peerpodConfig CR 的 limit 属性：

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

(1) 将 <value> 替换为您要定义的限制。

3.2.2.6. 配置工作负载对象

您可以通过将 kata-remote 配置为以下 pod 模板对象的运行时类来部署 OpenShift 沙盒容器工作负载：

- Pod 对象
- ReplicaSet 对象
- ReplicationController 对象
- StatefulSet 对象
- Deployment 对象
- deploymentConfig 对象

重要

不要在 openshift-sandboxed-containers-operator 命名空间中部署工作负载。为这些资源创建一个专用命名空间。

您可以通过在 YAML 文件中添加注解，定义工作负载是否使用配置映射中定义的默认实例大小进行部署。

如果您不想手动定义实例大小，您可以添加注解来使用自动实例大小，具体取决于可用内存。

先决条件

- 您已为供应商创建了 secret 对象。
- 您已为供应商创建了配置映射。
- 您已创建了 KataConfig 自定义资源 (CR)。

流程

1. 在 OpenShift Container Platform Web 控制台中，导航到 Workloads → workload type，如 Pods。
2. 在工作负载类型页面中，点对象查看其详情。
3. 点 YAML 标签。
4. 将 spec.runtimeClassName: kata-remote 添加到每个 pod 模板工作负载对象的清单中，如下例所示：

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

5. 向 pod 模板对象添加注解，以使用手动定义的实例大小或自动实例大小：
   - 要使用手动定义的实例大小，请添加以下注解：

```
apiVersion: v1
```

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

(1) 指定配置映射中定义的实例大小。

- 要使用自动实例大小，请添加以下注解：

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

定义可供工作负载使用的内存量。工作负载将根据可用内存量在自动实例大小上运行。

6. 点 Save 应用更改。

OpenShift Container Platform 创建工作负载对象并开始调度它。

验证

- 检查 pod 模板对象的 spec.runtimeClassName 字段。如果值为 kata-remote，则工作负载在 OpenShift 沙盒容器上运行，使用对等 pod。

3.2.3. 使用命令行部署工作负载

您可以使用命令行部署 OpenShift 沙盒容器工作负载。

3.2.3.1. 创建 secret

您必须在 OpenShift Container Platform 集群中创建 Secret 对象。secret 存储云供应商凭证，用于创建 pod 虚拟机(VM)镜像和对等 pod 实例。默认情况下，OpenShift 沙盒容器 Operator 根据用于创建集群的凭证创建 secret。但是，您可以手动创建使用不同的凭证的 secret。

先决条件

- 已安装并配置了 Azure CLI 工具。

流程

1. 检索 Azure 订阅 ID：

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

2. 生成 RBAC 内容。这会生成客户端 ID、客户端 secret 和租户 ID：

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

输出示例：

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

3. 记录要在 secret 对象中使用的 RBAC 输出。

4. 根据以下示例创建 peer-pods-secret.yaml 清单文件：

```
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)

2. 指定 AZURE_CLIENT_SECRET 值。

3. 指定 AZURE_TENANT_ID 值。

4. 指定 AZURE_SUBSCRIPTION_ID 值。

5. 通过应用清单来创建 secret 对象：

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

注意

如果更新 peer pod secret，您必须重启 peerpodconfig-ctrl-caa-daemon DaemonSet 来应用更改。

更新 secret 后，应用清单。然后运行以下命令来重启 cloud-api-adaptor pod：

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

重启守护进程集会重新创建对等 pod。它不会更新现有的 pod。

3.2.3.2. 创建配置映射

您必须在 OpenShift Container Platform 集群上为您的云供应商创建配置映射。

流程

1. 从 Azure 实例获取以下值：

a. 检索并记录 Azure VNet 名称：

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

这个值用于检索 Azure 子网 ID。

b. 检索并记录 Azure 子网 ID：

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

c. 检索并记录 Azure 网络安全组(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\""
```

d. 检索并记录 Azure 资源组：

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

e. 检索并记录 Azure 区域：

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

2. 根据以下示例创建 peer-pods-cm.yaml 清单：

```
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"
DISABLECVM: "true"
AZURE_IMAGE_ID: "<azure_image_id>" (5)
AZURE_REGION: "<azure_region>" (6)
AZURE_RESOURCE_GROUP: "<azure_resource_group>" (7)
```

1. 定义工作负载中没有定义类型时使用的默认实例大小。

2. 列出创建 pod 时可以指定的所有实例大小。这可让您为大型工作负载需要较少的内存和更小的实例大小的工作负载定义较小的实例大小。

3. 指定您检索的 AZURE_SUBNET_ID 值。

4. 指定您检索的 AZURE_NSG_ID 值。

5. 可选：默认情况下，这个值会在运行 KataConfig CR 时填充，使用基于集群凭证的 Azure 镜像 ID。如果创建自己的 Azure 镜像，请指定正确的镜像 ID。

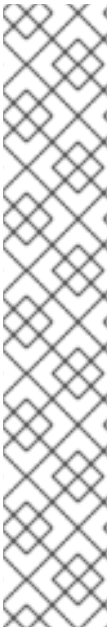
6. 指定您检索到的 AZURE_REGION 值。

7. 指定您检索的 AZURE_RESOURCE_GROUP 值。

3. 应用清单以创建配置映射：

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

为您的云供应商创建一个配置映射。



注意

如果更新 peer pod 配置映射，您必须重启 peerpodconfig-ctrl-caa-daemon daemonset 以应用更改。

更新配置映射后，应用清单。然后运行以下命令来重启 cloud-api-adaptor pod：

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

重启 daemonset 会重新创建对等 pod。它不会更新现有的 pod。

3.2.3.3. 创建 SSH 密钥 secret

您必须为 Azure 创建 SSH 密钥 secret 对象。

流程

1. 登录您的 OpenShift Container Platform 集群。

2. 运行以下命令来生成 SSH 密钥对：

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

3. 运行以下命令来创建 Secret 对象：

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

SSH 密钥 secret 已创建。

4. 删除您创建的 SSH 密钥：

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

3.2.3.4. 创建 KataConfig 自定义资源

您必须创建一个 KataConfig 自定义资源(CR)来作为 worker 节点上的运行时类安装 kata-remote。

创建 KataConfig CR 会触发 OpenShift 沙盒容器 Operator 来执行以下操作：

- 使用默认配置创建一个名为 kata-remote 的 RuntimeClass CR。这可让用户在 RuntimeClassName 字段中引用 CR 将工作负载配置为使用 kata-remote 作为运行时。此 CR 也指定运行时的资源开销。

OpenShift 沙盒容器将 kata-remote 安装为集群上的 *辅助* 可选运行时，而不是主运行时。

重要

创建 KataConfig CR 会自动重启 worker 节点。重启可能需要 10 到 60 分钟。妨碍重启时间的因素如下：

- 带有更多 worker 节点的大型 OpenShift Container Platform 部署。
- 激活 BIOS 和 Diagnostics 实用程序。
- 在硬盘而不是 SSD 上部署。
- 在物理节点上部署，如裸机，而不是在虚拟节点上部署。
- CPU 和网络较慢。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 根据以下示例创建 cluster-kataconfig.yaml 清单文件：

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

2. 可选：要在所选节点上安装 kata-remote，请根据以下示例指定节点标签：

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

(1) 指定所选节点的标签。

3. 创建 KataConfig CR：

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

新的 KataConfig CR 被创建，并在 worker 节点上作为运行时类安装 kata-remote。

在验证安装前，等待 kata-remote 安装完成，以及 worker 节点重新引导。

验证

- 运行以下命令监控安装进度：

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

安装 kataNodes 下所有 worker 的状态并且条件 InProgress 为 False 时，而不指定原因，则会在集群中安装 kata-remote。

详情请参阅 KataConfig 状态信息。

3.2.3.4.1. 可选：验证 pod 虚拟机镜像

在集群中安装 kata-remote 后，OpenShift 沙盒容器 Operator 会创建一个 pod 虚拟机镜像，用于创建对等 pod。此过程可能需要很长时间，因为镜像是在云实例上创建的。您可以通过检查您为云供应商创建的配置映射来验证 pod 虚拟机镜像是否已成功创建。

流程

1. 获取您为对等 pod 创建的配置映射：

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

2. 检查 YAML 文件 的状态 小节。

   如果 AZURE_IMAGE_ID 参数被填充，则 pod 虚拟机镜像已被成功创建。

故障排除

1. 运行以下命令来检索事件日志：

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

2. 运行以下命令来检索作业日志：

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

如果您无法解决这个问题，请提交红帽支持问题单并附加这两个日志的输出。

3.2.3.5. 可选：修改每个节点的对等 pod 虚拟机数量

您可以通过编辑 peerpodConfig 自定义资源(CR)来更改每个节点对等 pod 虚拟机(VM)的限制。

流程

1. 运行以下命令检查当前的限制：

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

2. 运行以下命令修改 peerpodConfig CR 的 limit 属性：

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

(1) 将 <value> 替换为您要定义的限制。

3.2.3.6. 配置工作负载对象

您可以通过将 kata-remote 配置为以下 pod 模板对象的运行时类来部署 OpenShift 沙盒容器工作负载：

- Pod 对象
- ReplicaSet 对象
- ReplicationController 对象

- StatefulSet 对象
- Deployment 对象
- deploymentConfig 对象

重要

不要在 openshift-sandboxed-containers-operator 命名空间中部署工作负载。为这些资源创建一个专用命名空间。

您可以通过在 YAML 文件中添加注解，定义工作负载是否使用配置映射中定义的默认实例大小进行部署。

如果您不想手动定义实例大小，您可以添加注解来使用自动实例大小，具体取决于可用内存。

先决条件

- 您已为供应商创建了 secret 对象。
- 您已为供应商创建了配置映射。
- 您已创建了 KataConfig 自定义资源 (CR)。

流程

1. 将 spec.runtimeClassName: kata-remote 添加到每个 pod 模板工作负载对象的清单中，如下例所示：

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

2. 向 pod 模板对象添加注解，以使用手动定义的实例大小或自动实例大小：

- 要使用手动定义的实例大小，请添加以下注解：

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

(1) 指定配置映射中定义的实例大小。

- 要使用自动实例大小，请添加以下注解：

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

定义可供工作负载使用的内存量。工作负载将根据可用内存量在自动实例大小上运行。

3. 运行以下命令，将更改应用到工作负载对象：

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

OpenShift Container Platform 创建工作负载对象并开始调度它。

验证

- 检查 pod 模板对象的 spec.runtimeClassName 字段。如果值为 kata-remote，则工作负载在 OpenShift 沙盒容器上运行，使用对等 pod。

第 4 章 在 IBM 上部署工作负载

您可以在 IBM Z® 和 IBM® LinuxONE 上部署 OpenShift 沙盒容器工作负载。

重要

在 IBM Z® 和 IBM® LinuxONE 上部署 OpenShift 沙盒容器工作负载只是一个技术预览功能。技术预览功能不受红帽产品服务等级协议（SLA）支持，且功能可能并不完整。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关红帽技术预览功能支持范围的更多信息，请参阅技术预览功能支持范围。

集群先决条件

- 已安装 Red Hat OpenShift Container Platform 4.14 或更高版本。
- 集群有三个控制节点和两个 worker 节点。

部署流程

虽然本文档只涉及 IBM Z®，但所有流程也适用于 IBM® LinuxONE。

您可以通过执行以下步骤部署 OpenShift 沙盒容器工作负载：

1. 在 KVM 主机上配置 libvirt 卷。
2. 创建 KVM 客户机镜像并将其上传到 libvirt 卷。
3. 创建对等 pod 虚拟机镜像并将其上传到 libvirt 卷。
4. 为 libvirt 提供程序创建 secret。

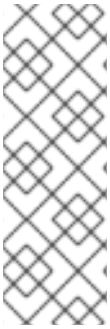
5. 为 libvirt 供应商创建配置映射。

6. 为您的 KVM 主机创建 SSH 密钥 secret。

7. 创建 KataConfig CR。

8. 可选：修改每个节点的对等 pod VM 限制。

9. 将您的工作负载对象配置为使用 kata-remote 运行时类。



注意

- 集群节点和对等 pod 必须位于同一 IBM Z® KVM 主机逻辑分区(LPAR)中。
- 集群节点和对等 pod 必须连接到同一子网。

4.1. 准备您的环境

执行以下步骤准备您的环境：

1. 确保集群有足够的资源。

2. 安装 OpenShift 沙盒容器 Operator。

4.1.1. 资源要求

对等 pod 虚拟机(VM)需要位于两个位置的资源：

- worker 节点。worker 节点存储元数据、Kata shim 资源(containerd-shim-kata-v2)、remote-hypervisor 资源(cloud-api-adaptor)，以及 worker 节点和对等 pod 虚拟机之间的隧道设置。

- 云实例。这是在云中运行的实际对等 pod 虚拟机。

Kubernetes worker 节点中使用的 CPU 和内存资源由 RuntimeClass (kata-remote)定义中包含的 pod 开销 处理，用于创建对等 pod。

在云中运行的对等 pod 虚拟机总数定义为 Kubernetes 节点扩展资源。这个限制是每个节点，并由 peerpodConfig 自定义资源(CR)中的 limit 属性设置。

在创建 kataConfig CR 并启用对等 pod 时，名为 peerpodconfig-openshift 的 peerpodConfig CR 会被创建，位于 openshift-sandboxed-containers-operator 命名空间中。

以下 peerpodConfig CR 示例显示默认的 spec 值：

```
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. 默认限制为每个节点 10 个虚拟机。

扩展资源名为 kata.peerpods.io/vm，并允许 Kubernetes 调度程序处理容量跟踪和核算。

您可以根据环境要求编辑每个节点的限制。如需更多信息，请参阅"修改对等 pod 中每个节点的虚拟机限制"。

变异 Webhook 将扩展的资源 kata.peerpods.io/vm 添加到 pod 规格中。如果存在，它还会从 pod 规格中删除任何特定于资源的条目。这可让 Kubernetes 调度程序考虑这些扩展资源，确保仅在资源可用时调度对等 pod。

变异 Webhook 修改 Kubernetes pod，如下所示：

- 变异 Webhook 会检查 pod 是否有预期的 RuntimeClassName 值，在 TARGET_RUNTIME_CLASS 环境变量中指定。如果 pod 规格中的值与 TARGET_RUNTIME_CLASS 的值不匹配，则 Webhook 会在不修改 pod 的情况下退出。

- 如果 RuntimeClassName 值匹配，webhook 会对 pod 规格进行以下更改：

  1. Webhook 从 pod 中所有容器和 init 容器的 resources 字段中删除每个资源规格。

  2. Webhook 通过修改 pod 中第一个容器的 resources 字段，将扩展资源 (kata.peerpods.io/vm)添加到 spec。Kubernetes 调度程序使用扩展资源 kata.peerpods.io/vm 用于核算目的。

注意

变异 Webhook 排除 OpenShift Container Platform 中的特定系统命名空间。如果在这些系统命名空间中创建了对等 pod，则使用 Kubernetes 扩展资源的资源核算不起作用，除非 pod spec 包含扩展资源。

作为最佳实践，定义集群范围的策略，仅允许在特定命名空间中创建对等 pod。

4.1.2. 安装 OpenShift 沙盒容器 Operator

您可以使用 OpenShift Container Platform Web 控制台或命令行界面(CLI)安装 OpenShift 沙盒容器 Operator。

4.1.2.1. 使用 Web 控制台安装 Operator

您可以使用 Red Hat OpenShift Container Platform Web 控制台安装 OpenShift 沙盒容器 Operator。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 在 OpenShift Container Platform Web 控制台中导航至 Operators → OperatorHub。

2. 在 Filter by keyword 字段中，输入 OpenShift sandboxed containers。

3. 选择 OpenShift 沙盒容器 Operator 标题并点 Install。

4. 在 Install Operator 页面中，从可用 Update Channel 选项列表中选择 stable。

5. 验证为 Installed Namespace 选择了 Operator recommended Namespace。这会在 openshift-sandboxed-containers-operator 命名空间中安装 Operator。如果此命名空间尚不存在，则会自动创建。

   注意

   尝试在 openshift-sandboxed-containers-operator 以外的命名空间中安装 OpenShift 沙盒容器 Operator 会导致安装失败。

6. 验证是否为 Approval Strategy 选择了 Automatic。Automatic 是默认值，当有新的 z-stream 发行版本可用时，自动启用对 OpenShift 沙盒容器的自动更新。

7. 点 Install。

OpenShift 沙盒容器 Operator 现已安装在集群中。

验证

1. 导航到 Operators → Installed Operators。

2. 验证 OpenShift 沙盒容器 Operator 是否已显示。

其他资源

- 在受限网络中使用 Operator Lifecycle Manager。
- 为断开连接的环境 在 Operator Lifecycle Manager 中配置代理支持。

4.1.2.2. 使用 CLI 安装 Operator

您可以使用 CLI 安装 OpenShift 沙盒容器 Operator。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 创建 Namespace.yaml 清单文件：

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

2. 运行以下命令创建命名空间：

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

3. 创建 OperatorGroup.yaml 清单文件：

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

4. 运行以下命令来创建 operator 组：

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

5. 创建 Subscription.yaml 清单文件：

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

6. 运行以下命令来创建订阅：

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

OpenShift 沙盒容器 Operator 现已安装在集群中。

验证

- 运行以下命令确保 Operator 已正确安装：

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

输出示例

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

4.1.2.3. 其他资源

- 在受限网络中使用 Operator Lifecycle Manager
- 为断开连接的环境 在 Operator Lifecycle Manager 中配置代理支持

4.2. 使用命令行部署工作负载

您可以使用命令行部署 OpenShift 沙盒容器工作负载。

4.2.1. 配置 libvirt 卷

您必须在 KVM 主机上配置 libvirt 卷。对等 pod 使用 Cloud API Adaptor 的 libvirt 提供程序来创建和管理虚拟机。

先决条件

- 已使用 OpenShift Container Platform Web 控制台或命令行在 OpenShift Container Platform 集群上安装 OpenShift 沙盒容器 Operator。
- 您有 KVM 主机的管理员特权。
- 您已在 KVM 主机上安装了 podman。
- 您已在 KVM 主机上安装了 virt-customize。

流程

1. 登录到 KVM 主机。
2. 运行以下命令设置 libvirt 池的名称：

```
$ export LIBVIRT_POOL=<libvirt_pool>
```

您需要 LIBVIRT_POOL 值来为 libvirt 提供程序创建 secret。

3. 运行以下命令设置 libvirt 池的名称：

```
$ export LIBVIRT_VOL_NAME=<libvirt_volume>
```

您需要 LIBVIRT_VOL_NAME 值来为 libvirt 提供程序创建 secret。

4. 运行以下命令，设置默认存储池位置的路径：

```
$ export LIBVIRT_POOL_DIRECTORY=<target_directory> (1)
```

(1) 为确保 libvirt 具有读写访问权限，请使用 libvirt 存储目录的子目录。默认为 /var/lib/libvirt/images/。

5. 运行以下命令来创建 libvirt 池：

```
$ virsh pool-define-as $LIBVIRT_POOL --type dir --target
"$LIBVIRT_POOL_DIRECTORY"
```

6. 运行以下命令来启动 libvirt 池：

```
$ virsh pool-start $LIBVIRT_POOL
```

7. 运行以下命令，为池创建 libvirt 卷：

```
$ virsh -c qemu:///system \
  vol-create-as --pool $LIBVIRT_POOL \
  --name $LIBVIRT_VOL_NAME \
  --capacity 20G \
  --allocation 2G \
  --prealloc-metadata \
  --format qcow2
```

4.2.2. 创建 KVM 客户机镜像

您必须创建一个 KVM 客户机镜像，并将其上传到 libvirt 卷。

先决条件

- IBM z15 或更高版本，或 IBM® LinuxONE114 或更高版本。
- 至少在使用 KVM 的 RHEL 9 或更高版本中运行一个 LPAR。

流程

1. 登录您的 OpenShift Container Platform 集群。
2. 如果您有 RHEL 订阅，请为 Red Hat Subscription Management 设置订阅环境变量：
   - 运行以下命令来设置机构 ID：

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

   - 运行以下命令来设置激活码：

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

3. 如果您没有 RHEL 订阅，请为 RHEL 设置订阅值：
   - 运行以下命令来设置机构 ID：

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

     (1) 指定您的 RHEL 机构 ID。

   - 运行以下命令来设置激活码：

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

(1) 指定 RHEL 激活码。

4. 登录到您的 IBM Z® 系统。

5. 将 s390x RHEL KVM 客户机镜像 从红帽客户门户网站下载 到 libvirt 存储目录，以授予 libvirt 正确访问权限。

   默认目录为 /var/lib/libvirt/images。此镜像用于生成对等 pod 虚拟机镜像，其中包含相关的二进制文件。

6. 运行以下命令，为下载的镜像设置 IMAGE_URL ：

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

(1) 指定 KVM 客户机镜像的路径。

7. 运行以下命令来注册客户端 KVM 镜像：

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

8. 运行以下命令来自定义客户机 KVM 镜像：

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

9. 运行以下命令设置镜像的校验和：

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

4.2.3. 构建对等 pod 虚拟机镜像

您必须构建对等 pod 虚拟机(VM)镜像，并将其上传到 libvirt 卷。

流程

1. 登录您的 OpenShift Container Platform 集群。

2. 运行以下命令克隆 cloud-api-adaptor 存储库：

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

3. 运行以下命令，进入 podvm 目录：

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

4. 创建生成最终 QCOW2 镜像的构建器镜像。

   - 如果您有一个订阅的 RHEL 系统，请运行以下命令：

```
$ podman build -t podvm_builder_rhel_s390x \
  --build-arg ARCH="s390x" \
  --build-arg GO_VERSION="1.21.3" \
  --build-arg PROTOC_VERSION="25.1" \
  --build-arg PACKER_VERSION="v1.9.4" \
  --build-arg RUST_VERSION="1.72.0" \
  --build-arg YQ_VERSION="v4.35.1" \
  --build-arg
YQ_CHECKSUM="sha256:4e6324d08630e7df733894a11830412a43703682d65a76f1fc925aac08268a45" \
  -f podvm/Dockerfile.podvm_builder.rhel .
```

   - 如果您有未订阅的 RHEL 系统，请运行以下命令：

```
$ podman build -t podvm_builder_rhel_s390x \
  --build-arg ORG_ID=$ORG_ID \
  --build-arg ACTIVATION_KEY=$ACTIVATION_KEY \
  --build-arg ARCH="s390x" \
  --build-arg GO_VERSION="1.21.3" \
  --build-arg PROTOC_VERSION="25.1" \
  --build-arg PACKER_VERSION="v1.9.4" \
  --build-arg RUST_VERSION="1.72.0" \
```

```
  --build-arg YQ_VERSION="v4.35.1" \
  --build-arg
YQ_CHECKSUM="sha256:4e6324d08630e7df733894a11830412a43703682d65a76f1f
c925aac08268a45" \
  -f podvm/Dockerfile.podvm_builder.rhel .
```

5. 运行以下命令，使用运行对等 pod 所需的二进制文件生成中间镜像软件包：

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

这个过程需要大量时间。

6. 运行以下命令，提取二进制文件并构建对等 pod QCOW2 镜像：

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

7. 运行以下命令来创建镜像目录环境变量：

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

(1) 指定镜像的目录。

8. 运行以下命令来创建镜像目录：

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

9. 运行以下命令保存提取的对等 pod QCOW2 镜像：

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

10. 将对等 pod QCOW2 镜像上传到 libvirt 卷：

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

4.2.4. 创建 secret

您必须在 OpenShift Container Platform 集群中创建 Secret 对象。

先决条件

- LIBVIRT_POOL.使用您在 KVM 主机上配置 libvirt 时设置的值。
- LIBVIRT_VOL_NAME.使用您在 KVM 主机上配置 libvirt 时设置的值。
- LIBVIRT_URI.这个值是 libvirt 网络的默认网关 IP 地址。检查 libvirt 网络设置以获取此值。

注意

如果 libvirt 使用默认网桥虚拟网络，您可以通过运行以下命令来获取 LIBVIRT_URI ：

```
$ 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}')
```

流程

1. 根据以下示例创建 peer-pods-secret.yaml 清单文件：

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

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

1. 指定 libvirt URI。
2. 指定 libvirt 池。
3. 指定 libvirt 卷名称。

2. 通过应用清单来创建 secret 对象：

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

注意

如果更新 peer pod secret，您必须重启 peerpodconfig-ctrl-caa-daemon DaemonSet 来应用更改。

更新 secret 后，应用清单。然后运行以下命令来重启 cloud-api-adaptor pod：

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

重启守护进程集会重新创建对等 pod。它不会更新现有的 pod。

4.2.5. 创建配置映射

您必须在 OpenShift Container Platform 集群上为 libvirt 供应商创建配置映射。

流程

1. 根据以下示例创建 peer-pods-cm.yaml 清单：

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

2. 应用清单以创建配置映射：

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

为您的 libvirt 供应商创建一个配置映射。

注意

如果更新 peer pod 配置映射，您必须重启 peerpodconfig-ctrl-caa-daemon daemonset 以应用更改。

更新配置映射后，应用清单。然后运行以下命令来重启 cloud-api-adaptor pod：

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

重启 daemonset 会重新创建对等 pod。它不会更新现有的 pod。

4.2.6. 创建 SSH 密钥 secret

您必须为 KVM 主机创建 SSH 密钥 secret 对象。

流程

1. 登录您的 OpenShift Container Platform 集群。

2. 运行以下命令来生成 SSH 密钥对：

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

3. 将公共 SSH 密钥复制到 KVM 主机：

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

4. 运行以下命令来创建 Secret 对象：

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

SSH 密钥 secret 已创建。

5. 删除您创建的 SSH 密钥：

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

4.2.7. 创建 KataConfig 自定义资源

您必须创建一个 KataConfig 自定义资源(CR)来作为 worker 节点上的运行时类安装 kata-remote。

创建 KataConfig CR 会触发 OpenShift 沙盒容器 Operator 来执行以下操作：

- 使用默认配置创建一个名为 kata-remote 的 RuntimeClass CR。这可让用户在 RuntimeClassName 字段中引用 CR 将工作负载配置为使用 kata-remote 作为运行时。此 CR 也指定运行时的资源开销。

OpenShift 沙盒容器将 kata-remote 安装为集群上的 *辅助* 可选运行时，而不是主运行时。

重要

创建 KataConfig CR 会自动重启 worker 节点。重启可能需要 10 到 60 分钟。妨碍重启时间的因素如下：

- 带有更多 worker 节点的大型 OpenShift Container Platform 部署。
- 激活 BIOS 和 Diagnostics 实用程序。
- 在硬盘而不是 SSD 上部署。
- 在物理节点上部署，如裸机，而不是在虚拟节点上部署。
- CPU 和网络较慢。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 根据以下示例创建 cluster-kataconfig.yaml 清单文件：

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

2. 可选：要在所选节点上安装 kata-remote，请根据以下示例指定节点标签：

```
apiVersion: kataconfiguration.openshift.io/v1
kind: KataConfig
metadata:
  name: cluster-kataconfig
```

```
spec:
  kataConfigPoolSelector:
    matchLabels:
      <label_key>: '<label_value>' (1)
# ...
```

(1) 指定所选节点的标签。

3. 创建 KataConfig CR：

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

新的 KataConfig CR 被创建，并在 worker 节点上作为运行时类安装 kata-remote。

在验证安装前，等待 kata-remote 安装完成，以及 worker 节点重新引导。

验证

- 运行以下命令监控安装进度：

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

安装 kataNodes 下所有 worker 的状态并且条件 InProgress 为 False 时，而不指定原因，则会在集群中安装 kata-remote。

详情请参阅 KataConfig 状态信息。

4.2.8. 可选：修改每个节点的对等 pod 虚拟机数量

您可以通过编辑 peerpodConfig 自定义资源(CR)来更改每个节点对等 pod 虚拟机(VM)的限制。

流程

1. 运行以下命令检查当前的限制：

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

2. 运行以下命令修改 peerpodConfig CR 的 limit 属性：

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

(1) 将 <value> 替换为您要定义的限制。

4.2.9. 配置工作负载对象

您可以通过将 kata-remote 配置为以下 pod 模板对象的运行时类来部署 OpenShift 沙盒容器工作负载：

- Pod 对象
- ReplicaSet 对象
- ReplicationController 对象
- StatefulSet 对象
- Deployment 对象
- deploymentConfig 对象

重要

不要在 openshift-sandboxed-containers-operator 命名空间中部署工作负载。为这些资源创建一个专用命名空间。

先决条件

- 您已为供应商创建了 secret 对象。
- 您已为供应商创建了配置映射。
- 您已创建了 KataConfig 自定义资源 (CR)。

流程

1. 将 spec.runtimeClassName: kata-remote 添加到每个 pod 模板工作负载对象的清单中，如下例所示：

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

   OpenShift Container Platform 创建工作负载对象并开始调度它。

验证

- 检查 pod 模板对象的 spec.runtimeClassName 字段。如果值为 kata-remote，则工作负载在 OpenShift 沙盒容器上运行，使用对等 pod。

第 5 章 监控

您可以使用 OpenShift Container Platform Web 控制台监控与沙盒工作负载和节点的健康状态相关的指标。

OpenShift 沙盒容器在 OpenShift Container Platform Web 控制台中有一个预先配置的仪表板。管理员还可以通过 Prometheus 访问和查询原始指标。

5.1. 关于指标

OpenShift 沙盒容器指标让管理员能够监控沙盒容器的运行方式。您可以在 OpenShift Container Platform Web 控制台中的 Metrics UI 中查询这些指标。

OpenShift 沙盒容器指标为以下类别收集：

Kata 代理指标

Kata 代理指标显示有关嵌入在沙盒容器中运行的 kata 代理进程的信息。这些指标包括 /proc/<pid>/[io, stat, status] 中的数据。

Kata 客户机操作系统指标

Kata 客户机操作系统指标显示沙盒容器中运行的客户机操作系统中的数据。这些指标包括 /proc/[stats, diskstats, meminfo, vmstats] 和 /proc/net/dev 中的数据。

hypervisor 指标

hypervisor 指标显示有关运行嵌入在沙盒容器中虚拟机的虚拟机监控程序的数据。这些指标主要包括 /proc/<pid>/[io, stat, status] 中的数据。

Kata 监控指标

Kata 监控器是收集指标数据并提供给 Prometheus 的进程。kata 监控指标显示有关 kata-monitor 进程本身的资源使用情况的详细信息。这些指标还包括 Prometheus 数据收集的计数器。

Kata containerd shim v2 指标

Kata containerd shim v2 指标显示有关 kata shim 进程的详细信息。这些指标包括来自 /proc/<pid>/[io, stat, status] 和详细的资源使用量指标的数据。

5.2. 查看指标

您可以在 OpenShift Container Platform Web 控制台的 Metrics 页面中访问 OpenShift 沙盒容器的指标。

先决条件

- 您可以使用具有 cluster-admin 角色或所有项目的查看权限的用户访问集群。

流程

1. 在 OpenShift Container Platform web 控制台中进入到 Observe → Metrics。
2. 在输入字段中，输入您要观察到的指标的查询。

   所有与 kata 相关的指标都以 kata 开头。键入 kata 会显示所有可用 kata 指标的列表。

在页面中会视觉化查询的指标。

其他资源

- 查询指标.
- 收集集群的数据。

第 6 章 卸装

您可以使用 OpenShift Container Platform Web 控制台或命令行卸载 OpenShift 沙盒容器。

卸载工作流

1. 删除工作负载 pod。
2. 删除 KataConfig 自定义资源。
3. 卸载 OpenShift 沙盒容器 Operator。
4. 删除 KataConfig 自定义资源定义。

6.1. 使用 WEB 控制台卸载

您可以使用 OpenShift Container Platform Web 控制台卸载 OpenShift 沙盒容器。

6.1.1. 删除工作负载 pod

您可以使用 OpenShift Container Platform Web 控制台删除 OpenShift 沙盒容器工作负载 pod。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。
- 您有一个使用 OpenShift 沙盒容器运行时类的 pod 列表。

流程

1. 在 OpenShift Container Platform Web 控制台中导航至 Workloads → Pods。

2. 在 **Search by name** 字段中输入您要删除的 **pod** 的名称。

3. 点 **pod** 名称打开它。

4. 在 **Details** 页面中，检查是否为 **Runtime** 类 显示 **kata** 或 **kata-remote**。

5. 点击 **Options** 菜单

   并选择 **Delete Pod**。

6. 点击 **Delete**。

6.1.2. 删除 KataConfig CR

您可以使用 Web 控制台删除 KataConfig 自定义资源(CR)。删除 KataConfig CR 会从集群中移除并卸载 kata 运行时及其相关资源。

重要

删除 KataConfig CR 会自动重启 worker 节点。重启可能需要 10 到 60 分钟。妨碍重启时间的因素如下：

- 带有更多 worker 节点的大型 OpenShift Container Platform 部署。
- 激活 BIOS 和 Diagnostics 实用程序。
- 在硬盘而不是 SSD 上部署。
- 在物理节点上部署，如裸机，而不是在虚拟节点上部署。
- CPU 和网络较慢。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。
- 您已删除所有使用 kata 作为 runtimeClass 的 pod。

流程

1. 在 OpenShift Container Platform web 控制台中导航至 Operators → Installed Operators。
2. 使用 Search by name 字段搜索 OpenShift 沙盒容器 Operator。
3. 点 Operator 打开它，然后选择 KataConfig 选项卡。
4. 点 KataConfig 资源的 Options 菜单

，然后选择 Delete KataConfig。

5. 在确认窗口中点击 Delete。

等待 kata 运行时和资源卸载，并使 worker 节点重启，然后继续下一步。

6.1.3. 卸载 Operator

您可以使用 OpenShift Container Platform Web 控制台卸载 OpenShift 沙盒容器 Operator。卸载 Operator 会删除该 Operator 的目录订阅、Operator 组和集群服务版本(CSV)。然后，您可以删除 openshift-sandboxed-containers-operator 命名空间。

先决条件

- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 在 OpenShift Container Platform web 控制台中导航至 Operators → Installed Operators。
2. 在 Search by name 字段中输入 OpenShift 沙盒容器 Operator 名称。
3. 点击 Operator 的 Options 菜单

   并选择 Uninstall Operator。
4. 在确认窗口中点 Uninstall。
5. 在 Search by name 字段中输入 openshift-sandboxed-containers-operator 名称。
6. 点命名空间的 Options 菜单

⋮

并选择 **Delete Namespace**。

注意

如果 **Delete Namespace** 选项不可用，代表您没有删除命名空间的权限。

7. 在 **Delete Namespace** 窗口中，输入 **openshift-sandboxed-containers-operator** 并点 **Delete**。

8. 点击 **Delete**。

6.1.4. 删除 KataConfig CRD

您可以使用 OpenShift Container Platform Web 控制台删除 KataConfig 自定义资源定义(CRD)。

先决条件

- 您可以使用具有 **cluster-admin** 角色的用户访问集群。
- 已删除 KataConfig CR。
- 已卸载 OpenShift 沙盒容器 Operator。

流程

1. 在 Web 控制台中，导航到 **Administration** → **CustomResourceDefinitions**。
2. 在 **Search by name** 字段中输入 KataConfig 名称。
3. 点 KataConfig CRD 的 **Options** 菜单

⋮

，然后选择 Delete CustomResourceDefinition。

4. 在确认窗口中点击 Delete。

5. 等待 KataConfig CRD 会从列表中消失。这可能需要几分钟。

6.2. 使用 CLI 卸载

您可以使用命令行界面(CLI)卸载 OpenShift 沙盒容器。

6.2.1. 删除工作负载 pod

您可以使用 CLI 删除 OpenShift 沙盒容器工作负载 pod。

先决条件

- 已安装 JSON 处理器(jq)工具。

流程

1. 运行以下命令来搜索 pod：

```
$ oc get pods -A -o json | jq -r '.items[] | \
  select(.spec.runtimeClassName == "<runtime>").metadata.name' (1)
```

(1) 为裸机部署指定 kata。为公有云、IBM Z® 和 IBM® LinuxONE 部署指定 kata-remote。

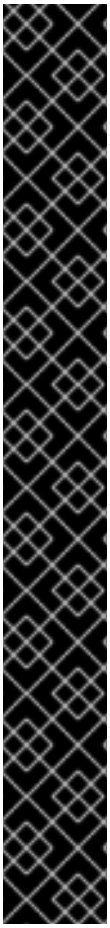
2. 运行以下命令来删除每个 pod：

```
$ oc delete pod <pod>
```

6.2.2. 删除 KataConfig CR

您可以使用命令行删除 KataConfig 自定义资源(CR)。

删除 KataConfig CR 会从集群中移除运行时及其相关资源。



重要

删除 KataConfig CR 会自动重启 worker 节点。重启可能需要 10 到 60 分钟。妨碍重启时间的因素如下：

- 带有更多 worker 节点的大型 OpenShift Container Platform 部署。
- 激活 BIOS 和 Diagnostics 实用程序。
- 在硬盘而不是 SSD 上部署。
- 在物理节点上部署，如裸机，而不是在虚拟节点上部署。
- CPU 和网络较慢。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

- 运行以下命令来删除 KataConfig CR：

```
$ oc delete kataconfig <kataconfig>
```

OpenShift 沙盒容器 Operator 会删除最初为在集群中启用运行时创建的所有资源。

验证

当您删除 KataConfig CR 时，CLI 会停止响应，直到所有 worker 节点重启为止。在执行验证前，您必须使删除过程完成。

- 要验证 KataConfig 自定义资源是否已删除，请运行以下命令：

```
$ oc get kataconfig <kataconfig>
```

输出示例

```
No KataConfig instances exist
```

6.2.3. 卸载 Operator

您可以使用 CLI 卸载 OpenShift 沙盒容器 Operator。您可以通过删除 Operator 订阅、Operator 组、集群服务版本(CSV)和命名空间来卸载 Operator。

先决条件

- 已安装 OpenShift CLI（oc）。
- 已安装命令行 JSON 处理器(jq)。
- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 运行以下命令，从订阅中获取 OpenShift 沙盒容器的集群服务版本(CSV)名称：

```
CSV_NAME=$(oc get csv -n openshift-sandboxed-containers-operator -o=custom-columns=:metadata.name)
```

2. 运行以下命令，从 Operator Lifecyle Manager (OLM)中删除 Operator 订阅：

```
$ oc delete subscription sandboxed-containers-operator -n openshift-sandboxed-containers-operator
```

3. 运行以下命令，删除 OpenShift 沙盒容器的 CSV 名称：

```
$ oc delete csv ${CSV_NAME} -n openshift-sandboxed-containers-operator
```

4. 运行以下命令来获取 Operator 组名称：

```
$ OG_NAME=$(oc get operatorgroup -n openshift-sandboxed-containers-operator -o=jsonpath={..name})
```

5. 运行以下命令来删除 Operator 组名称：

```
$ oc delete operatorgroup ${OG_NAME} -n openshift-sandboxed-containers-operator
```

6. 运行以下命令来删除 Operator 命名空间：

```
$ oc delete namespace openshift-sandboxed-containers-operator
```

6.2.4. 删除 KataConfig CRD

您可以使用命令行删除 KataConfig 自定义资源定义(CRD)。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您可以使用具有 cluster-admin 角色的用户访问集群。

- 已删除 KataConfig CR。
- 已卸载 OpenShift 沙盒容器 Operator。

流程

1. 运行以下命令来删除 KataConfig CRD：

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

验证

- 要验证 KataConfig CRD 是否已删除，请运行以下命令：

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

输出示例

```
Unknown CR KataConfig
```

第 7 章 升级

OpenShift 沙盒容器组件的升级由以下三个步骤组成：

1. 升级 OpenShift Container Platform 以更新 Kata 运行时及其依赖项。
2. 升级 OpenShift 沙盒容器 Operator 以更新 Operator 订阅。

您可以在 OpenShift 沙盒容器 Operator 升级前或之后升级 OpenShift Container Platform，但有以下例外。在升级 OpenShift 沙盒容器 Operator 后，始终立即应用 KataConfig 补丁。

7.1. 升级资源

OpenShift 沙盒容器资源使用 Red Hat Enterprise Linux CoreOS (RHCOS) 扩展部署到集群中。

RHCOS 扩展 沙盒容器 包含运行 OpenShift 沙盒容器所需的组件，如 Kata 容器运行时、虚拟机监控程序 QEMU 和其他依赖项。您可以通过将集群升级到 OpenShift Container Platform 的新版本来升级扩展。

有关升级 OpenShift Container Platform 的更多信息，请参阅更新集群。

7.2. 升级 OPERATOR

使用 Operator Lifecycle Manager (OLM) 手动或自动升级 OpenShift 沙盒容器 Operator。在初始部署期间，选择手动或自动升级可决定将来的升级模式。对于手动升级，OpenShift Container Platform Web 控制台会显示集群管理员可安装的可用更新。

有关在 Operator Lifecycle Manager (OLM)中升级 OpenShift 沙盒容器 Operator 的更多信息，请参阅更新已安装的 Operator。

第 8 章 故障排除

当对 OpenShift 沙盒容器进行故障排除时，您可以创建一个支持问题单，并使用 must-gather 工具提供调试信息。

如果您是集群管理员，您还可以自行查看日志，启用更详细的日志级别。

8.1. 为红帽支持收集数据

在提交问题单时同时提供您的集群信息，可以帮助红帽支持为您进行排除故障。

您可使用 must-gather 工具来收集有关 OpenShift Container Platform 集群的诊断信息，包括虚拟机和有关 OpenShift 沙盒容器的其他数据。

为了获得快速支持，请提供 OpenShift Container Platform 和 OpenShift 沙盒容器的诊断信息。

使用 must-gather 工具

oc adm must-gather CLI 命令可收集最有助于解决问题的集群信息，包括：

- 资源定义
- 服务日志

默认情况下，oc adm must-gather 命令使用默认的插件镜像，并写入 ./must-gather.local。

另外，您可以使用适当的参数运行命令来收集具体信息，如以下部分所述：

- 要收集与一个或多个特定功能相关的数据，请使用 --image 参数和镜像，如以下部分所述。

  例如：

```
$ oc adm must-gather --image=registry.redhat.io/openshift-sandboxed-containers/osc-must-gather-rhel9:1.6.0
```

- 要收集审计日志，请使用 -- /usr/bin/gather_audit_logs 参数，如以下部分所述。

  例如：

```
$ oc adm must-gather -- /usr/bin/gather_audit_logs
```

> **注意**
>
> 作为默认信息集合的一部分，不会收集审计日志来减小文件的大小。

当您运行 oc adm must-gather 时，集群的新项目中会创建一个带有随机名称的新 pod。在该 pod 上收集数据，并保存至以 must-gather.local 开头的一个新目录中。此目录在当前工作目录中创建。

例如：

```
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
...
```

另外，您可以使用 --run-namespace 选项在特定命名空间中运行 oc adm must-gather 命令。

例如：

```
$ oc adm must-gather --run-namespace <namespace> --image=registry.redhat.io/openshift-sandboxed-containers/osc-must-gather-rhel9:1.6.0
```

8.2. 收集日志数据

以下功能和对象与 OpenShift 沙盒容器关联：

- 属于 OpenShift 沙盒容器资源的所有命名空间及其子对象
- 所有 OpenShift 沙盒容器自定义资源定义 (CRD)

您可以为使用 kata 运行时运行的每个 pod 收集以下组件日志：

- Kata 代理日志
- Kata 运行时日志
- QEMU 日志
- 审计日志
- CRI-O 日志

8.2.1. 为 CRI-O 运行时启用调试日志

您可以通过更新 KataConfig CR 中的 logLevel 字段来启用调试日志。这会更改运行 OpenShift 沙盒容器的 worker 节点的 CRI-O 运行时中的日志级别。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

1. 将现有 KataConfig CR 中的 logLevel 字段更改为 debug ：

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

2. 监控 kata-oc 机器配置池，直到 UPDATED 的值为 True，表示所有 worker 节点都已更新：

```
$ oc get mcp kata-oc
```

输出示例

```
NAME     CONFIG              UPDATED  UPDATING  DEGRADED  MACHINECOUNT
READYMACHINECOUNT  UPDATEDMACHINECOUNT  DEGRADEDMACHINECOUNT
AGE
kata-oc  rendered-kata-oc-169   False    True      False     3           1                1
0                  9h
```

验证

1. 使用机器配置池中的节点启动 debug 会话：

```
$ oc debug node/<node_name>
```

2. 将根目录改为 /host ：

```
# chroot /host
```

3. 验证 crio.conf 文件中的更改：

```
# crio config | egrep 'log_level
```

输出示例

```
log_level = "debug"
```

8.2.2. 查看组件的调试日志

集群管理员可以使用调试日志进行故障排除。每个节点的日志会输出到节点日志中。

您可以查看以下 OpenShift 沙盒容器组件的日志：

- Kata 代理
- Kata runtime (containerd-shim-kata-v2)
- virtiofsd

QEMU 仅生成警告和错误日志。这些警告和错误会在 Kata 运行时日志和带有额外的 qemuPid 字段的 CRI-O 日志中打印到节点日志。

QEMU 日志示例

```
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
```

当 QEMU 启动时，Kata 运行时会在 QEMU 启动时打印 Start logging QEMU，并在 QEMU 停止时停止日志记录 QEMU。使用 qemuPid 字段的两个日志消息之间会出现这个错误。QEMU 的实际错误消息以红色显示。

QEMU 客户机的控制台也会输出到节点日志中。您可以查看客户机控制台日志以及 Kata 代理日志。

先决条件

- 已安装 OpenShift CLI(oc)。
- 您可以使用具有 cluster-admin 角色的用户访问集群。

流程

- 要查看 Kata 代理日志和客户机控制台日志，请运行以下命令：

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t kata -g “reading guest console”
```

- 要查看 Kata 运行时日志，请运行以下命令：

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t kata
```

- 要查看 virtiofsd 日志，请运行以下命令：

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t virtiofsd
```

- 要查看 QEMU 日志，请运行以下命令：

```
$ oc debug node/<nodename> -- journalctl -D /host/var/log/journal -t kata -g "qemuPid=\d+"
```

其他资源

- 在 OpenShift Container Platform 文档中收集有关集群的数据 https://docs.redhat.com/documentation/en-us/openshift_container_platform/4.15/html-single/support/index#support_gathering_data_gathering-cluster-data

附录 A. KATACONFIG 状态信息

下表显示了具有两个 worker 节点的集群的 KataConfig 自定义资源(CR)的状态消息。

表 A.1. KataConfig 状态信息

Status	描述
初始安装 当创建 **KataConfig** CR 并在两个 worker 上安装 **kata** 时，会在几秒钟内显示以下状态。	conditions: message: Performing initial installation of kata on cluster reason: Installing status: 'True' type: InProgress kataNodes: nodeCount: 0 readyNodeCount: 0
安装 在几秒钟内，状态会改变。	kataNodes: nodeCount: 2 readyNodeCount: 0 waitingToInstall: - worker-0 - worker-1
安装 (Worker-1 安装开始) 在短时间内，状态会改变，表示一个节点启动了 **kata** 的安装，而另一个则处于等待状态。这是因为在任何给定时间只能有一个节点不可用。**nodeCount** 保留为 2，因为两个节点最终都会收到 **kata**，但 **readyNodeCount** 当前还未达到该状态。	kataNodes: installing: - worker-1 nodeCount: 2 readyNodeCount: 0 waitingToInstall: - worker-0
安装 （安装了Worker-1，worker-0 安装已启动） 一段时间后，**worker-1** 将完成安装，从而导致状态的变化。**readyNodeCount** 更新至 1，这表示 **worker-1** 现在已准备好执行 **kata** 工作负载。在安装过程结束时创建运行时类之前，您无法调度或运行 **kata** 工作负载。	kataNodes: installed: - worker-1 installing: - worker-0 nodeCount: 2 readyNodeCount: 1

Status	描述

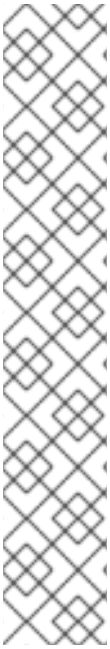
已安装
 	
安装后，两个 worker 都被列出为 installed，**InProgress** 条件过渡到 **False**，而不指定原因，表示集群中安装 **kata**。
	conditions:
	  message: ""
	  reason: ""
	  status: 'False'
	  type: InProgress
	kataNodes:
	  installed:
	  - worker-0
	  - worker-1
	  nodeCount: 2
	  readyNodeCount: 2

Status	描述

初始卸载
 	
如果在 worker 上同时安装了 **kata**，并且删除了 **KataConfig** 从集群中删除 **kata**，则两个 worker 会简要进入等待状态。
	conditions:
	  message: Removing kata from cluster
	  reason: Uninstalling
	  status: 'True'
	  type: InProgress
	kataNodes:
	  nodeCount: 0
	  readyNodeCount: 0
	  waitingToUninstall:
	  - worker-0
	  - worker-1

卸装
 	
几秒钟后，其中一个 worker 开始卸载。
	kataNodes:
	  nodeCount: 0
	  readyNodeCount: 0
	  uninstalling:
	  - worker-1
	  waitingToUninstall:
	  - worker-0

卸装
 	
worker-1 完成，worker-0 开始卸载。
	kataNodes:
	  nodeCount: 0
	  readyNodeCount: 0
	  uninstalling:
	  - worker-0



注意

reason 字段也可以报告以下原因：

- 失败 ：如果节点无法完成转换，则报告此报告。状态报告 True，消息 为 Node <node_name> Degraded: <error_message_from_the_node>。
- BlockedByExistingKataPods ：如果在卸载 kata 运行时的集群中运行了 pod，则会报告它。status 字段为 False，消息 为 Existing pod，使用 "kata" RuntimeClass found。Please delete the pods for KataConfig delete to proceed。如果与集群 control plane 的通信失败，则报告的技术错误消息，如 Failed to list kata pod: <error_message >。