OpenShift Container Platform 4.16

虚拟化

OpenShift Virtualization 安装、使用和发行注记

Last Updated: 2024-10-12

OpenShift Container Platform 4.16 虚拟化

OpenShift Virtualization 安装、使用和发行注记

法律通告

摘要

本文档提供有关如何在 OpenShift Container Platform 中使用 OpenShift Virtualization 的信息。

目录

第 1 章 关于

1.1. 关于 OPENSHIFT VIRTUALIZATION

OpenShift Virtualization 的功能与支持范围。

1.1.1. OpenShift Virtualization 的作用

OpenShift 虚拟化（OpenShift virtualization）是 OpenShift Container Platform 的一个附加组件，可用于运行和管理虚拟机工作负载以及容器工作负载。

OpenShift Virtualization 通过 Kubernetes 自定义资源添加新对象至 OpenShift Container Platform 集群中，以启用虚拟化任务。这些任务包括：

- 创建和管理 Linux 和 Windows 虚拟机 (VM)
- 在集群中运行 pod 和虚拟机工作负载
- 通过各种控制台和 CLI 工具连接至虚拟机
- 导入和克隆现有虚拟机
- 管理虚拟机上附加的网络接口控制器和存储磁盘
- 在节点间实时迁移虚拟机

增强版 web 控制台提供了一个图形化的门户界面 来管理虚拟化资源以及 OpenShift Container Platform 集群容器和基础架构。

OpenShift Virtualization 的设计和测试，可与 Red Hat OpenShift Data Foundation 功能配合工作。

重要

使用 OpenShift Data Foundation 部署 OpenShift Virtualization 时，您必须为 Windows 虚拟机磁盘创建一个专用存储类。详情请参阅为 Windows 虚拟机优化 ODF PersistentVolume。

您可以将 OpenShift Virtualization 与 OVN-Kubernetes、OpenShift SDN 或 认证的 OpenShift CNI 插件中列出的其他认证网络插件一起使用。

您可以通过安装 Compliance Operator 并运行带有 **ocp4-moderate** 和 **ocp4-moderate-node** 配置集的扫描来检查您的 OpenShift Virtualization 集群的合规性。Compliance Operator 使用 NIST 认证工具 OpenSCAP 扫描并执行安全策略。

1.1.1.1. OpenShift Virtualization 支持的集群版本

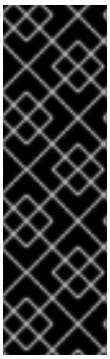
支持在 OpenShift Container Platform 4.16 集群中使用 OpenShift Virtualization 4.16。要使用 OpenShift Virtualization 的最新 z-stream 版本，您必须首先升级到 OpenShift Container Platform 的最新版本。

1.1.2. 关于虚拟机磁盘的卷和访问模式

如果您将存储 API 与已知的存储供应商搭配使用，则会自动选择卷和访问模式。但是，如果您使用没有存储配置集的存储类，您必须配置卷和访问模式。

要获得最佳结果，请使用 **ReadWriteMany** (RWX)访问模式和 **Block** 卷模式。这一点非常重要：

- 实时迁移需要 **ReadWriteMany** (RWX) 访问模式。
- 块 卷模式的性能优于 **Filesystem** 卷模式。这是因为 **Filesystem** 卷模式使用更多存储层，包括文件系统层和磁盘镜像文件。虚拟机磁盘存储不需要这些层。
  例如，如果您使用 Red Hat OpenShift Data Foundation，Ceph RBD 卷优先于 CephFS 卷。



重要

您不能使用以下配置实时迁移虚拟机：

- 具有 **ReadWriteOnce** (RWO) 访问模式的存储卷
- 透传功能，比如 GPU

对于这些虚拟机，将 **evictionStrategy** 字段设置为 **None**。**None** 策略会在节点重启过程中关闭虚拟机。

1.1.3. 单节点 Openshift 的不同

您可以在单节点 OpenShift 上安装 OpenShift Virtualization。

但是，您应该注意单节点 OpenShift 不支持以下功能：

- 高可用性
- Pod 中断预算
- 实时迁移
- 配置了驱除策略的虚拟机或模板

1.1.4. 其他资源

- OpenShift Container Platform 存储的常见术语表
- 关于单节点 OpenShift
- 支持的安装程序
- Pod 中断预算
- 关于实时迁移
- 驱除策略
- 调整和扩展指南
- OpenShift Virtualization 4.x 支持的限制

1.2. 安全策略

了解 OpenShift Virtualization 安全和授权。

关键点

- OpenShift Virtualization 遵循 **restricted** Kubernetes pod 安全标准配置集，旨在强制实施 Pod 安全性的当前最佳实践。
- 虚拟机 (VM) 工作负载作为非特权 pod 运行。
- 为 **kubevirt-controller** 服务帐户定义 安全性上下文约束 (SCC)。
- OpenShift Virtualization 组件的 TLS 证书都会被更新并自动轮转。

1.2.1. 关于工作负载安全性

默认情况下，虚拟机 (VM) 工作负载不会在 OpenShift Virtualization 中使用 root 权限运行，且不支持的 OpenShift Virtualization 功能需要 root 权限。

对于每个虚拟机，**virt-launcher** pod 以 *会话模式*运行一个 **libvirt** 实例，用于管理虚拟机进程。在会话模式中，**libvirt** 守护进程以非 root 用户帐户运行，仅允许同一用户标识符 (UID) 下运行的客户端的连接。因此，虚拟机作为非特权 pod 运行，遵循最小特权的安全原则。

1.2.2. TLS 证书

OpenShift Virtualization 组件的 TLS 证书都会被更新并自动轮转。您不需要手动刷新它们。

自动续订计划

TLS 证书会根据以下调度自动删除并替换：

- kubeVirt 证书每天都会被更新。
- 容器化数据导入程序控制器（CDI）证书每 15 天更新一次。
- MAC 池证书会每年续订。

自动 TLS 证书轮转不会破坏任何操作。例如，以下操作可在没有任何中断的情况下继续工作：

- 迁移
- 镜像上传
- VNC 和控制台连接

1.2.3. 授权

OpenShift Virtualization 使用基于角色的访问控制 (RBAC)为人类用户和服务帐户定义权限。为服务帐户定义的权限控制 OpenShift Virtualization 组件可以执行的操作。

您还可以使用 RBAC 角色管理用户对虚拟化功能的访问。例如，管理员可以创建一个 RBAC 角色，它提供启动虚拟机所需的权限。然后，管理员可以通过将角色绑定到特定用户来限制访问权限。

1.2.3.1. OpenShift Virtualization 的默认集群角色

通过使用集群角色聚合，OpenShift Virtualization 会扩展默认的 OpenShift Container Platform 集群角色，使其包含访问虚拟化对象的权限。

表 1.1. OpenShift Virtualization 集群角色

默认集群角色	OpenShift Virtualization 集群角色	OpenShift Virtualization 集群角色描述
view	**kubevirt.io:view**	此用户可以查看集群中的所有 OpenShift Virtualization 资源，但不能创建、删除、修改或访问它们。例如，用户可以看到虚拟机 (VM) 正在运行，但不能将其关闭或访问其控制台。
edit	**kubevirt.io:edit**	可以修改集群中的所有 OpenShift Virtualization 资源的用户。例如，用户可以创建虚拟机、访问虚拟机控制台和删除虚拟机。
admin	**kubevirt.io:admin**	具有所有 OpenShift Virtualization 资源的完整权限的用户，包括删除资源集合。用户也可以查看和修改 OpenShift Virtualization 运行时配置，该配置位于 **openshift-cnv** 命名空间中的 **HyperConverged** 自定义资源中。

1.2.3.2. OpenShift Virtualization 中存储功能的 RBAC 角色

为 Containerized Data Importer (CDI) 授予以下权限，包括 **cdi-operator** 和 **cdi-controller** 服务帐户。

1.2.3.2.1. 集群范围的 RBAC 角色

表 1.2. cdi.kubevirt.io API 组的聚合集群角色

CDI 集群角色	Resources	Verbs
cdi.kubevirt.io:admin	**datavolumes**, **uploadtokenrequests**	* (all)
	dataVolumes/source	**create**
cdi.kubevirt.io:edit	**datavolumes**, **uploadtokenrequests**	*
	dataVolumes/source	**create**
cdi.kubevirt.io:view	**cdiconfigs**, **dataimportcrons**, **datasources**, **datavolumes**, **objecttransfers**, **storageprofiles**, **volumeimportsources**, **volumeuploadsources**, **volumeclonesources**	**get**, **list**, **watch**
	dataVolumes/source	**create**
cdi.kubevirt.io:config-reader	**cdiconfigs**, **storageprofiles**	**get**, **list**, **watch**

表 1.3. cdi-operator 服务帐户的集群范围角色

API 组	Resources	Verbs
rbac.authorization.k8s.io	**clusterrolebindings**, **clusterroles**	**get**, **list**, **watch**, **create**, **update**, **delete**
security.openshift.io	**securitycontextconstraints**	**get**, **list**, **watch**, **update**, **create**
apiextensions.k8s.io	**customresourcedefinitions**, **customresourcedefinitions/status**	**get**, **list**, **watch**, **create**, **update**, **delete**
cdi.kubevirt.io	*	*
upload.cdi.kubevirt.io	*	*
admissionregistration.k8s.io	**validatingwebhookconfigurations**, **mutatingwebhookconfigurations**	**create**, **list**, **watch**
admissionregistration.k8s.io	**validatingwebhookconfigurations** 允许列表：**cdi-api-dataimportcron-validate, cdi-api-populator-validate, cdi-api-datavolume-validate, cdi-api-validate, objecttransfer-api-validate**	**get**, **update**, **delete**
admissionregistration.k8s.io	**mutatingwebhookconfigurations** 允许列表：**cdi-api-datavolume-mutate**	**get**, **update**, **delete**
apiregistration.k8s.io	**apiservices**	**get**, **list**, **watch**, **create**, **update**, **delete**

表 1.4. cdi-controller 服务帐户的集群范围角色

API 组	Resources	Verbs
"" (core)	**events**	**create**, **patch**

API 组	Resources	Verbs
"" (core)	**persistentvolumeclaims**	**get**, **list**, **watch**, **create**, **update**, **delete**, **deletecollection**, **patch**
"" (core)	**persistentvolumes**	**get**, **list**, **watch**, **update**
"" (core)	**persistentvolumeclaims/finalizers**, **pods/finalizers**	**update**
"" (core)	**pods**, **services**	**get**, **list**, **watch**, **create**, **delete**
"" (core)	**configmaps**	**get**, **create**
storage.k8s.io	**storageclasses**, **csidrivers**	**get**, **list**, **watch**
config.openshift.io	**proxies**	**get**, **list**, **watch**
cdi.kubevirt.io	*	*
snapshot.storage.k8s.io	**volumesnapshots**, **volumesnapshotclasses**, **volumesnapshotcontents**	**get**, **list**, **watch**, **create**, **delete**
snapshot.storage.k8s.io	**volumesnapshots**	**update**, **deletecollection**
apiextensions.k8s.io	**customresourcedefinitions**	**get**, **list**, **watch**
scheduling.k8s.io	**priorityclasses**	**get**, **list**, **watch**
image.openshift.io	**imagestreams**	**get**, **list**, **watch**
"" (core)	**secrets**	**create**
kubevirt.io	**virtualmachines/finalizers**	**update**

1.2.3.2.2. 命名空间的 RBAC 角色

表 1.5. cdi-operator 服务帐户的命名空间角色

API 组	Resources	Verbs
rbac.authorization.k8s.io	**rolebindings**, **roles**	**get**, **list**, **watch**, **create**, **update**, **delete**
"" (core)	**serviceaccounts**, **configmaps**, **events**, **secrets**, **services**	**get**, **list**, **watch**, **create**, **update**, **patch**, **delete**
apps	**deployments**, **deployments/finalizers**	**get**, **list**, **watch**, **create**, **update**, **delete**
route.openshift.io	**routes**, **routes/custom-host**	**get**,**list**,**watch**,**create**,**update**
config.openshift.io	**proxies**	**get**, **list**, **watch**
monitoring.coreos.com	**servicemonitors**, **prometheusrules**	**get**, **list**, **watch**, **create**, **delete**, **update**, **patch**
coordination.k8s.io	**leases**	**get**, **create**, **update**

表 1.6. cdi-controller 服务帐户的命名空间角色

API 组	Resources	Verbs
"" (core)	**configmaps**	**get**, **list**, **watch**, **create**, **update**, **delete**
"" (core)	**secrets**	**get**, **list**, **watch**
batch	**cronjobs**	**get**, **list**, **watch**, **create**, **update**, **delete**
batch	**jobs**	**create**, **delete**, **list**, **watch**
coordination.k8s.io	**leases**	**get**, **create**, **update**
networking.k8s.io	**ingresses**	**get**, **list**, **watch**
route.openshift.io	**Routes**	**get**, **list**, **watch**

1.2.3.3. kubevirt-controller 服务帐户的额外 SCC 和权限

Pod 的安全上下文约束（SCC）控制权限。这些权限包括 Pod（容器集合）可以执行的操作以及它们可以访问的资源。您可以使用 SCC 定义 Pod 运行必须满足的一组条件，以便其能被系统接受。

virt-controller 是一个集群控制器，可为集群中的虚拟机创建 **virt-launcher** pod。这些 pod 由 **kubevirt-controller** 服务帐户授予权限。

kubevirt-controller 服务帐户被授予额外的 SCC 和 Linux 功能，以便能够创建具有适当权限的 **virt-launcher** Pod。这些扩展权限允许虚拟机使用超出典型 pod 范围的 OpenShift Virtualization 功能。

kubevirt-controller 服务帐户被授予以下 SCC:

- **scc.AllowHostDirVolumePlugin = true**
  这允许虚拟机使用 hostpath 卷插件。
- **scc.AllowPrivilegedContainer = false**
  可确保 virt-launcher pod 不是作为特权容器运行。
- **scc.AllowedCapabilities = []corev1.Capability{"SYS_NICE", "NET_BIND_SERVICE"}**
  - **SYS_NICE** 允许设置 CPU 关联性。
  - **NET_BIND_SERVICE** 允许 DHCP 和 Slirp 操作。

查看 kubevirt-controller 的 SCC 和 RBAC 定义

您可以使用 **oc** 工具查看 **kubevirt-controller** 的 **SecurityContextConstraints** 定义：

```
$ oc get scc kubevirt-controller -o yaml
```

您可以使用 **oc** 工具查看 **kubevirt-controller** clusterrole 的 RBAC 定义：

```
$ oc get clusterrole kubevirt-controller -o yaml
```

1.2.4. 其他资源

- 管理安全性上下文约束
- 使用 RBAC 定义和应用权限
- 创建集群角色
- 集群角色绑定命令
- 启用用户权限跨命名空间克隆数据卷

1.3. OPENSHIFT VIRTUALIZATION 架构

Operator Lifecycle Manager (OLM) 为 OpenShift Virtualization 的每个组件部署 Operator pod：

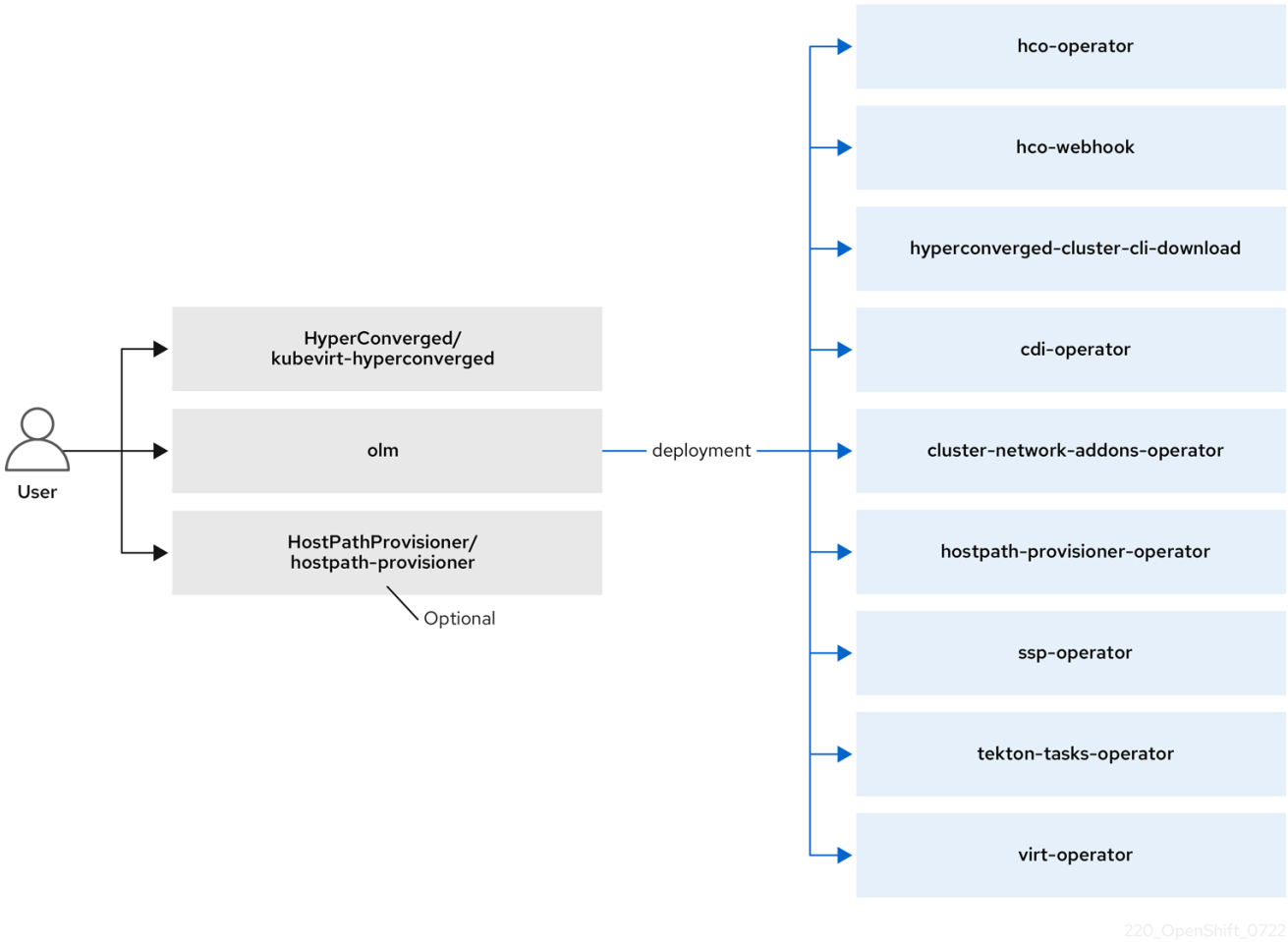
- Compute: **virt-operator**
- Storage: **cdi-operator**
- Network: **cluster-network-addons-operator**
- Scaling: **ssp-operator**

OLM 还会部署 **hyperconverged-cluster-operator** pod，它负责其他组件的部署、配置和生命周期，以及几个 helper pod: **hco-webhook** 和 **hyperconverged-cluster-cli-download**。

成功部署所有 Operator pod 后，您应该创建 **HyperConverged** 自定义资源 (CR)。**HyperConverged** CR 中的配置充当 OpenShift Virtualization 的单个来源，并指导 CR 的行为。

HyperConverged CR 为其协调循环中的所有其他组件的 operator 创建对应的 CR。然后，每个 Operator 会为 OpenShift Virtualization control plane 创建资源，如守护进程集、配置映射和其他组件。例如，当 HyperConverged Operator (HCO) 创建 **KubeVirt** CR 时，OpenShift Virtualization Operator 会协调它并创建其他资源，如 **virt-controller**、**virt-handler** 和 **virt-api**。

OLM 部署 Hostpath Provisioner (HPP) Operator，但在创建 **hostpath-provisioner** CR 前它无法正常工作。



- Virtctl 客户端命令

1.3.1. 关于 HyperConverged Operator (HCO)

HCO、**hco-operator** 提供了一种单一入口点，用于部署和管理 OpenShift Virtualization 和几个带有建议的默认值的帮助程序运算符。它还会为这些操作器创建自定义资源(CR)。

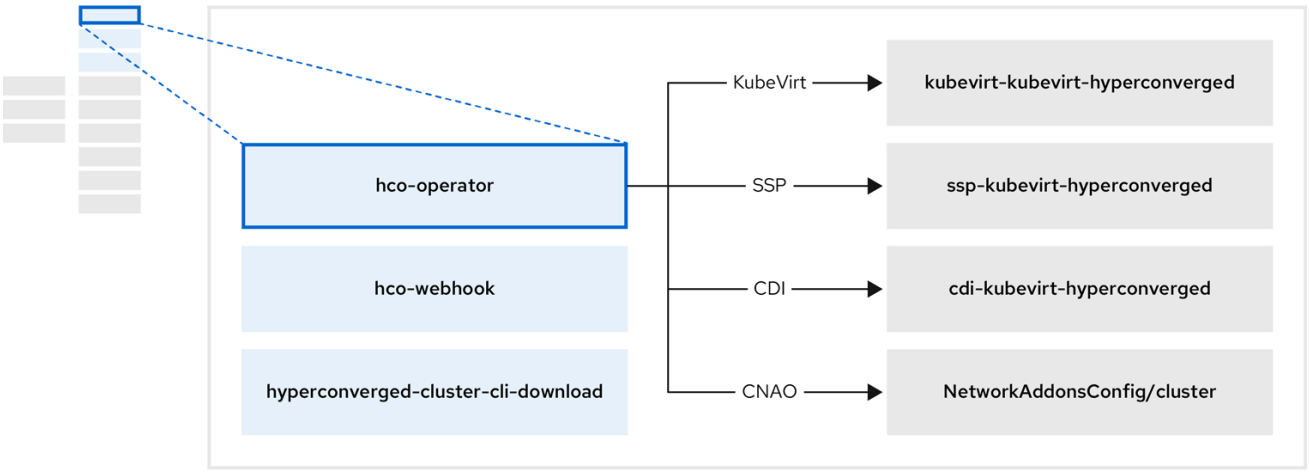


表 1.7. HyperConverged Operator 组件

组件	描述
deployment/hco-webhook	验证 **HyperConverged** 自定义资源内容。
deployment/hyperconverged-cluster-cli-download	提供 **virtctl** 工具二进制文件，以便直接从集群下载它们。
KubeVirt/kubevirt-kubevirt-hyperconverged	包含 OpenShift Virtualization 需要的所有 operator、CR 和对象。
SSP/ssp-kubevirt-hyperconverged	调度、扩展和性能 (SSP) CR。这由 HCO 自动创建。
CDI/cdi-kubevirt-hyperconverged	Containerized Data Importer (CDI) CR。这由 HCO 自动创建。
NetworkAddonsConfig/cluster	指示并由 **cluster-network-addons-operator** 管理的 CR。

1.3.2. 关于 Containerized Data Importer (CDI) Operator

CDI Operator **cdi-operator**, 管理 CDI 及其相关资源，它使用数据卷将虚拟机(VM)镜像导入到持久性卷声明(PVC)中。

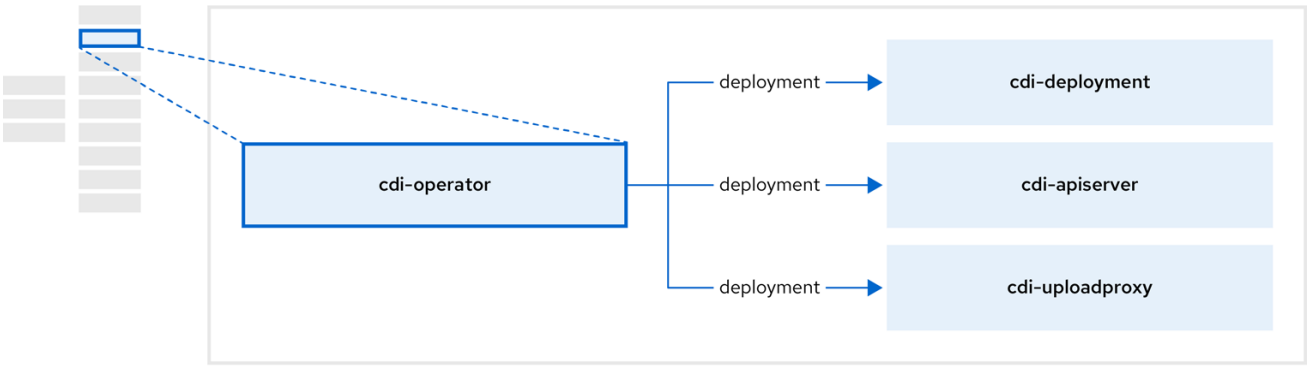


表 1.8. CDI Operator 组件

组件	描述
deployment/cdi-apiserver	通过提供安全上传令牌来管理将虚拟机磁盘上传到 PVC 的授权过程。
deployment/cdi-uploadproxy	将外部磁盘上传流量定向到适当的上传服务器 pod，以便将其写入正确的 PVC。需要有效的上传令牌。
pod/cdi-importer	helper（帮助程序）Pod，在创建数据卷时将虚拟机镜像导入到 PVC 中。

1.3.3. 关于 Cluster Network Addons Operator

Cluster Network Addons Operator **cluster-network-addons-operator** 在集群中部署网络组件，并管理扩展网络功能的相关资源。

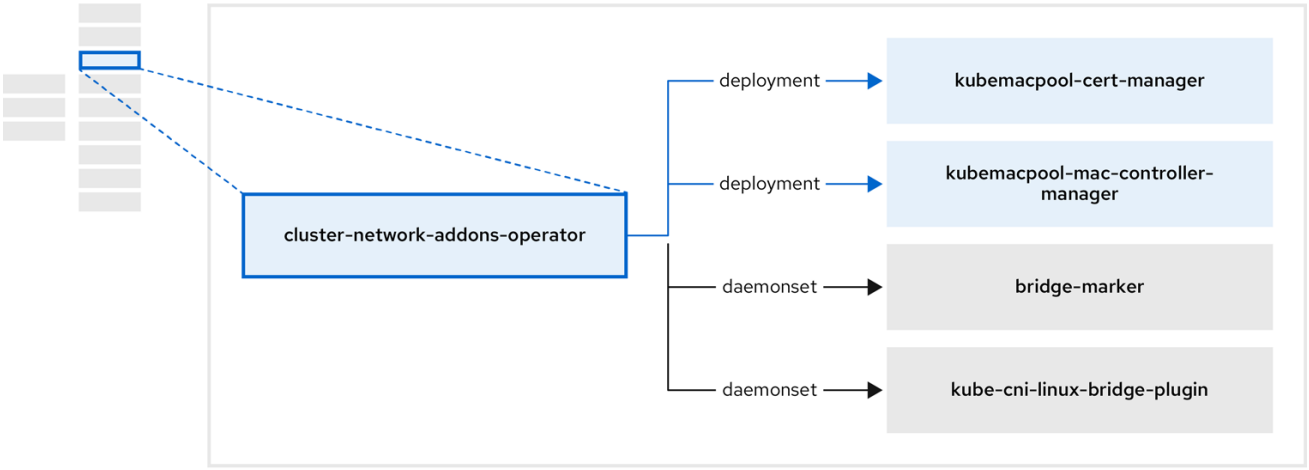


表 1.9. Cluster Network Addons Operator 组件

组件	描述
deployment/kubemacpool-cert-manager	管理 Kubemacpool 的 webhook 的 TLS 证书。
deployment/kubemacpool-mac-controller-manager	为虚拟机(VM)网络接口卡(NIC)提供 MAC 地址池服务。
daemonset/bridge-marker	将节点上可用的网络桥接标记为节点资源。
daemonset/kube-cni-linux-bridge-plugin	在集群节点上安装 Container Network Interface (CNI) 插件，通过网络附加定义将虚拟机附加到 Linux 网桥。

1.3.4. 关于 Hostpath Provisioner (HPP) Operator

HPP Operator **hostpath-provisioner-operator**，部署和管理多节点 HPP 和相关资源。

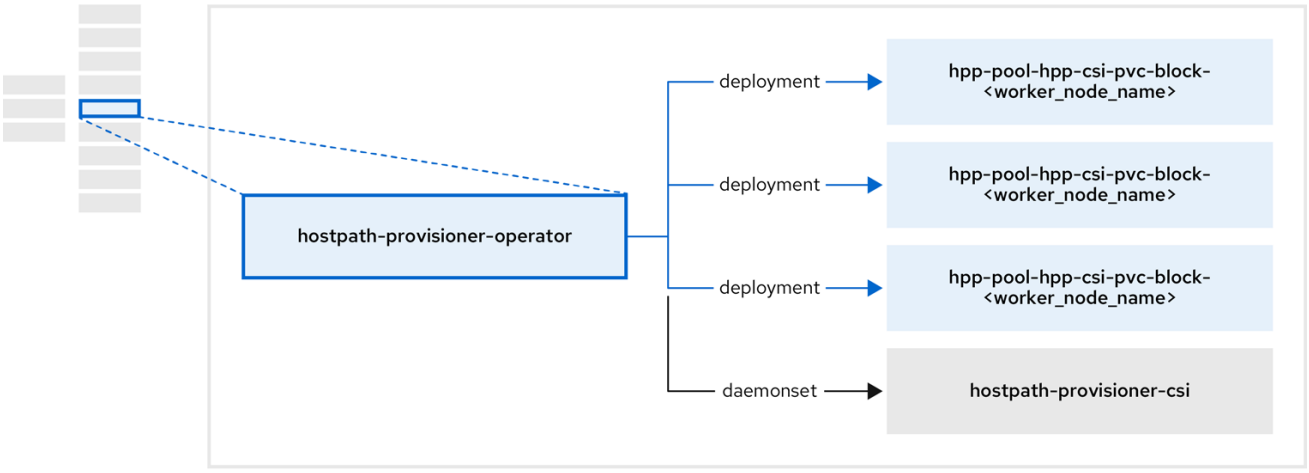


表 1.10. HPP Operator 组件

组件	描述
deployment/hpp-pool-hpp-csi-pvc-block-<worker_node_name>	为指定 HPP 的每个节点提供一个 worker。pod 在节点上挂载指定的后备存储。
daemonset/hostpath-provisioner-csi	实现 HPP 的容器存储接口(CSI)驱动程序接口。
daemonset/hostpath-provisioner	实现 HPP 的传统驱动程序接口。

1.3.5. 关于 Scheduling、Scale 和 Performance (SSP) Operator

SSP Operator、**ssp-operator**、部署通用模板、相关的默认引导源、管道任务和模板验证器。

1.3.6. 关于 OpenShift Virtualization Operator

OpenShift Virtualization Operator、**virt-operator**、部署、升级和管理 OpenShift Virtualization，而不影响当前虚拟机 (VM) 工作负载。另外，OpenShift Virtualization Operator 部署通用实例类型和通用首选项。

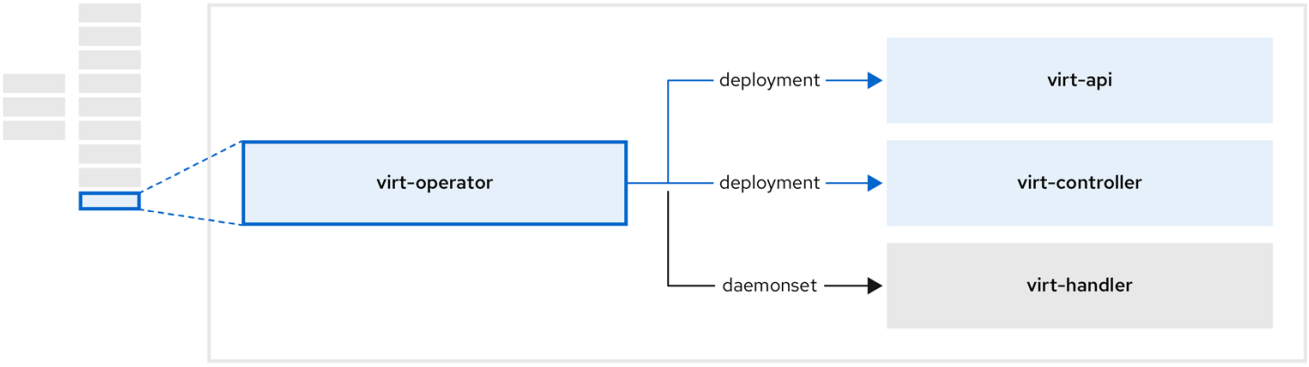


表 1.11. virt-operator 组件

组件	描述
deployment/virt-api	用作所有与虚拟化相关的流的入口点的 HTTP API 服务器。
deployment/virt-controller	观察创建新虚拟机实例对象并创建对应的 pod。当 pod 调度到某个节点上时，**virt-controller** 会使用节点名称更新虚拟机。
daemonset/virt-handler	监控对虚拟机的任何更改并指示 **virt-launcher** 执行所需操作。此组件特定于节点。
pod/virt-launcher	包含由 **libvirt** 和 **qemu** 实施的用户创建的虚拟机。

第 2 章 发行注记

2.1. OPENSHIFT VIRTUALIZATION 发行注记

2.1.1. 提供文档反馈

要报告错误或改进文档，请登录到 Red Hat JIRA 帐户 并提交 JIRA 问题。

2.1.2. 关于 Red Hat OpenShift Virtualization

使用 Red Hat OpenShift Virtualization，您可以将传统虚拟机 (VM) 带入 OpenShift Container Platform 中，并将它们与容器一同运行。在 OpenShift Virtualization 中，虚拟机是可以使用 OpenShift Container Platform Web 控制台或命令行管理的原生 Kubernetes 对象。

OpenShift Virtualization 由  图标表示。

OpenShift Virtualization 可以与 OVN-Kubernetes 或 OpenShiftSDN 默认 Container Network Interface（CNI）网络供应商一起使用。

了解更多有关 OpenShift Virtualization 的作用。

了解更多有关 OpenShift Virtualization 架构和部署 的信息。

为 OpenShift Virtualization 准备集群。

2.1.2.1. OpenShift Virtualization 支持的集群版本

支持在 OpenShift Container Platform 4.16 集群中使用 OpenShift Virtualization 4.16。要使用 OpenShift Virtualization 的最新 z-stream 版本，您必须首先升级到 OpenShift Container Platform 的最新版本。

2.1.2.2. 支持的客户端操作系统

要查看 OpenShift Virtualization 支持的客户机操作系统，请参阅 Red Hat OpenStack Platform、Red Hat Virtualization、OpenShift Virtualization 和带有 KVM 的 Red Hat Enterprise Linux 中的认证的客户机操作系统。

2.1.2.3. Microsoft Windows SVVP 认证

OpenShift Virtualization 已在 Microsoft 的 Windows Server Virtualization Validation Program (SVVP) 中认证来运行 Windows Server 的工作负载。

SVVP 认证适用于：

- Red Hat Enterprise Linux CoreOS worker。在 Microsoft SVVP Catalog 中，它们名为 *Red Hat OpenShift Container Platform 4 on RHEL CoreOS 9*。
- Intel 和 AMD CPU。

2.1.3. 快速启动

有几个 OpenShift Virtualization 功能提供快速入门导览。要查看导览，请点 OpenShift Container Platform Web 控制台标题的菜单栏中的 **Help** 图标 **?**，然后选择 **Quick Starts**。您可以通过在 **Filter** 字段中输入关键字 **virtualization** 来过滤可用的导览。

2.1.4. 新增和改变的功能

此发行版本添加了与以下组件和概念相关的新功能和增强：

2.1.4.1. 安装和更新

- 升级到 OpenShift Virtualization 4.16 后，之前通过垃圾回收移除的数据卷可能会重新创建。这是预期的行为。您可以忽略重新创建的数据卷，因为数据卷垃圾回收现在被禁用。

2.1.4.2. 虚拟化

- Windows 10 虚拟机现在使用带有 TPM 的 UEFI 引导。
- 启用 **AutoResourceLimits** 功能门自动管理虚拟机的 CPU 和内存限值。
- KubeVirt Tekton 任务现在作为 OpenShift Container Platform Pipelines 目录 的一部分提供。

2.1.4.3. 网络

- 现在，您可以使用无头服务，访问连接到一个具有稳定 FQDN 的默认内部 pod 网络的虚拟机。

2.1.4.4. Web 控制台

- 热插虚拟机现已正式发布。如果无法热插虚拟机，则 **RestartRequired** 会应用到虚拟机。您可以在 web 控制台的 **Diagnoses** 选项卡中查看此条件。
- 现在，您可以在从实例类型创建 Microsoft Windows 虚拟机时选择 **sysprep** 选项。在以前的版本中，您必须在创建后通过自定义虚拟机来设置 **sysprep** 选项。

2.1.4.5. 监控

- 作为管理员，您可以通过启用 **downwardMetrics** 功能门并配置一个 **downwardMetrics** 设备，为 OpenShift Virtualization 通过 virtio-serial 端口向客户虚拟机公开有限的主机和虚拟机 (VM) 指标。用户使用 **vm-dump-metrics** 工具或从命令行检索指标。
  在 Red Hat Enterprise Linux (RHEL) 9 中，使用命令行查看 downward 指标。Red Hat Enterprise Linux (RHEL) 9 平台不支持 **vm-dump-metrics** 工具。

2.1.4.6. 主要的技术变化

- 虚拟机至少需要 1 GiB 分配内存才能启用内存热插。如果虚拟机分配的内存小于 1 GiB，则禁用内存热插。
- OpenShift Virtualization 警报的 runbooks 现在只在 **openshift/runbooks** git 存储库中维护。现在，到 runbook 源文件的链接 可以来代替删除的 runbook。

2.1.5. 弃用和删除的功能

2.1.5.1. 已弃用的功能

弃用的功能包括在当前发行版本中，并被支持。但是，弃用的功能将在以后的发行版本中被删除，且不建议在新部署中使用。

- **tekton-tasks-operator** 已被弃用，Tekton 任务和示例管道现在由 **ssp-operator** 部署。
- **copy-template**,**modify-vm-template**, 和 **create-vm-from-template** 任务已弃用。
- 对 Windows Server 2012 R2 模板的支持已弃用。
- 警报 **KubeVirtComponentExceedsRequestedMemory** 和 **KubeVirtComponentExceedsRequestedCPU** 已被弃用。您可以安全地静默它们。

2.1.5.2. 删除的功能

当前版本不支持删除的功能。

2.1.6. 技术预览功能

这个版本中的一些功能当前还处于技术预览状态。它们并不适用于在生产环境中使用。请参阅红帽门户网站中关于对技术预览功能支持范围的信息：

技术预览功能支持范围

- 现在，您可以为 整个集群配置虚拟机驱除策略。
- 现在，您可以在 OpenShift Virtualization 主机中启用嵌套虚拟化。
- 集群管理员现在可以在 OpenShift Container Platform Web 控制台的 **Overview → Settings → Preview features** 下对命名空间启用 CPU 资源限值。
- 集群管理员现在可以使用 **wasp-agent** 工具通过过量使用内存量、在 RAM 中并将交换资源分配给虚拟机工作负载，在其 集群中配置更高的虚拟机工作负载密度。
- OpenShift Virtualization 现在支持与 Red Hat OpenShift Data Foundation (ODF) 区域灾难恢复的兼容性。

2.1.7. 已知问题

监控

- Pod Disruption Budget(PDB)可防止 pod 意外中断。如果 PDB 检测到 pod 中断，则 **openshift-monitoring** 会每 60 分钟发送 **PodDisruptionBudgetAtLimit** 警报，以使用 **LiveMigrate** 驱除策略。(CNV-33834)
  - 作为临时解决方案，静默警报。

节点

- 卸载 OpenShift Virtualization 不会删除 OpenShift Virtualization 创建的 **feature.node.kubevirt.io** 节点标签。您必须手动删除标签。(CNV-38543)
- 在具有不同计算节点的异构集群中，启用了 HyperV reenlightenment 的虚拟机无法调度到不支持时间戳扩展(TSC)或具有适当 TSC 频率的节点。(BZ#2151169)

Storage

- 如果您在 AWS 上使用 Portworx 作为存储解决方案并创建虚拟机磁盘镜像，则创建的镜像可能会小于预期，因为文件系统开销被认为是两次。(CNV-40217)
    - 作为临时解决方案，您可以手动扩展持久性卷声明 (PVC)，以便在初始置备过程完成后增加可用空间。
- 在某些情况下，多个虚拟机可以以读写模式挂载相同的 PVC，这可能会导致数据崩溃。(CNV-13500)
    - 作为临时解决方案，请避免在使用多个虚拟机的读写模式中使用单个 PVC。
- 如果您使用 **csi-clone** 克隆策略克隆超过 100 个虚拟机，则 Ceph CSI 可能无法清除克隆。手动删除克隆也可能会失败。(CNV-23501)
    - 作为临时解决方案，您可以重启 **ceph-mgr** 来清除虚拟机克隆。

虚拟化

- 在具有混合 CPU 类型的集群中虚拟机迁移可能会失败。(CNV-43195)
    - 作为临时解决方案，您可以在 VM spec 级别或集群级别设置 CPU 模型。
- 当在 Windows 虚拟机中添加虚拟受信任的平台模块 (vTPM) 设备时，BitLocker Drive Encryption 系统检查通过，即使 vTPM 设备没有持久性。这是因为 vTPM 设备不是持久性存储，并在 **virt-launcher** pod 生命周期中使用临时存储恢复加密密钥。当虚拟机迁移或关闭并重启时，vTPM 数据会丢失。(CNV-36448)
- OpenShift Virtualization 将 pod 使用的服务帐户令牌链接到该特定 pod。OpenShift Virtualization 通过创建包含令牌的磁盘镜像来实施服务帐户卷。如果您迁移虚拟机，则服务帐户卷无效。(CNV-33835)
    - 作为临时解决方案，使用用户帐户而不是服务帐户，因为用户帐户令牌没有绑定到特定 pod。
- 随着 RHSA-2023:3722 公告的发布，在启用了 FIPS 的 Red Hat Enterprise Linux (RHEL) 9 系统上，对 TLS 1.2 连接强制**Extended Master Secret** (EMS)扩展 (RFC 7627) 。这符合 FIPS-140-3 要求。TLS 1.3 不受影响。
  不支持 EMS 或 TLS 1.3 的旧的 OpenSSL 客户端现在无法连接到运行在 RHEL 9 上的 FIPS 服务器。同样，FIPS 模式下的 RHEL 9 客户端无法连接到只支持没有 EMS 的 TLS 1.2 服务器。在实践中意味着这些客户端无法连接到 RHEL 6、RHEL 7 和非 RHEL 传统操作系统上的服务器。这是因为传统的 OpenSSL 1.0.x 版本不支持 EMS 或 TLS 1.3。如需更多信息，请参阅 使用 Red Hat Enterprise Linux 9.2 强制执行的 TLS 扩展 "Extended Master Secret" 。
    - 作为临时解决方案，将旧的 OpenSSL 客户端升级到支持 TLS 1.3 的版本，并将 OpenShift Virtualization 配置为使用 TLS 1.3，为 FIPS 模式使用 **Modern** TLS 安全配置集类型。

Web 控制台

- 首次部署 OpenShift Container Platform 集群时，使用 Web 控制台从模板或实例类型创建虚拟机，如果您没有 **cluster-admin** 权限，则会失败。
    - 作为临时解决方案，集群管理员必须首先创建一个配置映射 ，以便其他用户使用模板和实例类型来创建虚拟机。(CNV-38284)
- 当您通过从 web 控制台中的 **Create PersistentVolumeClaim** 列表中选择 **With Data upload form** 来创建持久性卷声明 (PVC) 时，使用 **Upload Data** 字段将数据上传到 PVC 会失败。(CNV-37607)

第 3 章 开始使用

3.1. OPENSHIFT VIRTUALIZATION 入门

您可以通过安装和配置基本环境来探索 OpenShift Virtualization 的功能和功能。

注意

集群配置过程需要 **cluster-admin** 权限。

3.1.1. 规划和安装 OpenShift Virtualization

在 OpenShift Container Platform 集群中计划并安装 OpenShift Virtualization：

- 为 OpenShift Virtualization 规划裸机集群。
- 为 OpenShift Virtualization 准备集群。
- 安装 OpenShift Virtualization Operator。
- 安装 **virtctl** 命令行界面(CLI)工具。

规划和安装资源

- 关于虚拟机磁盘的存储卷。
- 使用启用了 CSI 的存储供应商。
- 为虚拟机配置本地存储。
- 安装 Kubernetes NMState Operator。
- 为虚拟机指定节点。
- **virtctl** 命令。

3.1.2. 创建和管理虚拟机

创建虚拟机：

- 从一个红帽镜像创建虚拟机。
  您可以使用红帽模板或实例类型创建虚拟机。
- 从自定义镜像创建虚拟机。
  您可以通过从容器 registry 或网页导入自定义镜像来创建虚拟机，方法是从本地机器上传镜像，或者克隆持久性卷声明(PVC)。

将虚拟机连接到二级网络：

- Linux 网桥网络.
- 打开虚拟网络(OVN)-Kubernetes 辅助网络。
- 单根 I/O 虚拟化(SR-IOV)网络.

注意

默认情况下，虚拟机连接到 pod 网络。

连接到虚拟机：

- 连接到虚拟机的串行控制台 或 VNC 控制台。
- 使用 SSH 连接到虚拟机。
- 连接到 Windows 虚拟机的桌面查看器。

管理虚拟机：

- 使用 Web 控制台管理虚拟机。
- 使用 **virtctl** CLI 工具管理虚拟机。
- 导出虚拟机。

3.1.3. 后续步骤

- 查看安装后配置选项。
- 配置存储选项和自动引导源更新。
- 了解监控和健康检查。
- 了解实时迁移。
- 使用 OpenShift API for Data Protection (OADP) 来备份和恢复虚拟机 。
- 调整并扩展集群。

3.2. 使用 CLI 工具

您可使用 **virtctl** 命令行工具管理 OpenShift Virtualization 资源。

您可以使用 **libguestfs** 命令行工具访问和修改虚拟机(VM)磁盘镜像。您可以使用 **virtctl libguestfs** 命令部署 **libguestfs**。

3.2.1. 安装 virtctl

要在 Red Hat Enterprise Linux (RHEL) 9、Linux、Windows 和 MacOS 操作系统上安装 **virtctl**，您可以下载并安装 **virtctl** 二进制文件。

要在 RHEL 8 上安装 **virtctl**，您可以启用 OpenShift Virtualization 仓库，然后安装 **kubevirt-virtctl** 软件包。

3.2.1.1. 在 RHEL 9、Linux、Windows 或 macOS 上安装 virtctl 二进制文件

您可以从 OpenShift Container Platform web 控制台下载适用于操作系统的 **virtctl** 二进制文件，然后安装它。

流程

1. 在 web 控制台中进入到 **Virtualization → Overview** 页面。
2. 点 **Download virtctl** 链接为您的操作系统下载 **virtctl** 二进制文件。
3. 安装 **virtctl** ：
   - 对于 RHEL 9 和其他 Linux 操作系统：
     a. 解压缩存档文件：

     ```
     $ tar -xvf <virtctl-version-distribution.arch>.tar.gz
     ```

     b. 运行以下命令使 **virtctl** 二进制可执行文件：

     ```
     $ chmod +x <path/virtctl-file-name>
     ```

     c. 将 **virtctl** 二进制文件移到 **PATH** 环境变量中的目录中。
     您可以运行以下命令来检查您的路径：

     ```
     $ echo $PATH
     ```

     d. 设置 **KUBECONFIG** 环境变量：

     ```
     $ export KUBECONFIG=/home/<user>/clusters/current/auth/kubeconfig
     ```

   - 对于 Windows:
     a. 解压缩存档文件。
     b. 进入解压的目录中，双击 **virtctl** 可执行文件来安装客户端。
     c. 将 **virtctl** 二进制文件移到 **PATH** 环境变量中的目录中。
     您可以运行以下命令来检查您的路径：

     ```
     C:\> path
     ```

   - macOS：
     a. 解压缩存档文件。
     b. 将 **virtctl** 二进制文件移到 **PATH** 环境变量中的目录中。
     您可以运行以下命令来检查您的路径：

     ```
     echo $PATH
     ```

3.2.1.2. 在 RHEL 8 上安装 virtctl RPM

您可以通过启用 OpenShift Virtualization 仓库并安装 **kubevirt-virtctl** 软件包，在 Red Hat Enterprise Linux (RHEL) 8 上安装 **virtctl** RPM 软件包。

先决条件

- 集群中的每个主机都必须通过 Red Hat Subscription Manager (RHSM) 注册，并具有有效的 OpenShift Container Platform 订阅。

流程

1. 使用 **subscription-manager** CLI 工具启用 OpenShift Virtualization 存储库，以运行以下命令：

```
# subscription-manager repos --enable cnv-4.16-for-rhel-8-x86_64-rpms
```

2. 运行以下命令安装 **kubevirt-virtctl** 软件包：

```
# yum install kubevirt-virtctl
```

3.2.2. virtctl 命令

virtctl 客户端是用于管理 OpenShift Virtualization 资源的命令行实用程序。

注意

除非另有指定，否则虚拟机(VM)命令也适用于虚拟机实例(VMI)。

3.2.2.1. virtctl 信息命令

您可使用 **virtctl** 信息命令查看 **virtctl** 客户端的信息。

表 3.1. 信息命令

命令	描述
virtctl version	查看 **virtctl** 客户端和服务器版本。
virtctl help	查看 **virtctl** 命令列表。
virtctl <command> -h|--help	查看特定命令的选项列表。
virtctl 选项	查看任何 **virtctl** 命令的全局命令选项列表。

3.2.2.2. VM 信息命令

您可使用 **virtctl** 查看有关虚拟机(VM)和虚拟机实例(VMI)的信息。

表 3.2. VM 信息命令

命令	描述
virtctl fslist <vm_name>	查看客户机机器上可用的文件系统。
virtctl guestosinfo <vm_name>	查看客户机机器上操作系统的信息。
virtctl userlist <vm_name>	查看客户机机器上的登录用户。

3.2.2.3. VM 清单创建命令

您可使用 **virtctl create** 命令为虚拟机、实例类型和首选项创建清单。

表 3.3. VM 清单创建命令

命令	描述
virtctl create vm	创建 **VirtualMachine** (VM) 清单。
virtctl create vm --name <vm_name>	创建虚拟机清单，指定虚拟机的名称。
virtctl create vm --instancetype <instancetype_name>	创建使用现有集群范围实例类型的虚拟机清单。
virtctl create vm --instancetype=virtualmachineinstancetype/<instancetype_name>	创建使用现有命名空间的实例类型的虚拟机清单。
virtctl create instancetype --cpu <cpu_value> --memory <memory_value> --name <instancetype_name>	为集群范围的实例类型创建清单。
virtctl create instancetype --cpu <cpu_value> --memory <memory_value> --name <instancetype_name> --namespace <namespace_value>	为命名空间实例类型创建清单。
virtctl create preference --name <preference_name>	为集群范围的虚拟机首选项创建清单，为首选项指定一个名称。
virtctl create preference --namespace <namespace_value>	为命名空间虚拟机首选项创建清单。

3.2.2.4. VM 管理命令

您可使用 **virtctl** 虚拟机(VM)管理命令管理和迁移虚拟机(VM)和虚拟机实例(VMI)。

表 3.4. VM 管理命令

命令	描述
virtctl start <vm_name>	启动虚拟机。
virtctl start --paused <vm_name>	以暂停状态启动虚拟机。这个选项可让您从 VNC 控制台中断引导过程。
virtctl stop <vm_name>	停止虚拟机。

命令	描述
virtctl stop <vm_name> --grace-period 0 --force	强制停止虚拟机。这个选项可能会导致数据不一致或数据丢失。
virtctl pause vm <vm_name>	暂停虚拟机。机器状态保存在内存中。
virtctl unpause vm <vm_name>	取消暂停虚拟机。
virtctl migrate <vm_name>	迁移虚拟机。
virtctl migrate-cancel <vm_name>	取消虚拟机迁移。
virtctl restart <vm_name>	重启虚拟机。

3.2.2.5. VM 连接命令

您可使用 **virtctl** connection 命令来公开端口并连接到虚拟机(VM)和虚拟机实例(VMI)。

表 3.5. VM 连接命令

命令	描述
virtctl console <vm_name>	连接到虚拟机的串行控制台。
virtctl expose vm <vm_name> --name <service_name> --type <ClusterIP|NodePort|LoadBalancer> --port <port>	创建转发虚拟机的指定端口的服务，并在节点的指定端口上公开服务。 示例：**virtctl expose vm rhel9_vm --name rhel9-ssh --type NodePort --port 22**
virtctl scp -i <ssh_key> <file_name> <user_name>@<vm_name>	将文件从机器复制到虚拟机。此命令使用 SSH 密钥对的私钥。虚拟机必须配置有公钥。
virtctl scp -i <ssh_key> <user_name@<vm_name>: <file_name> .	将文件从虚拟机复制到您的机器中。此命令使用 SSH 密钥对的私钥。虚拟机必须配置有公钥。
virtctl ssh -i <ssh_key> <user_name>@<vm_name>	与虚拟机打开 SSH 连接。此命令使用 SSH 密钥对的私钥。虚拟机必须配置有公钥。
virtctl vnc <vm_name>	连接到虚拟机的 VNC 控制台。 已安装 **virt-viewer**。

命令	描述
virtctl vnc --proxy-only=true <vm_name>	显示端口号，并使用任何查看器通过 VNC 连接手动连接到 VMI。
virtctl vnc --port=<port-number> <vm_name>	如果该端口可用，则指定端口号用于在指定端口上运行代理。 如果没有指定端口号，代理会在随机端口上运行。

3.2.2.6. VM 导出命令

使用 **virtctl vmexport** 命令来创建、下载或删除从虚拟机、虚拟机快照或持久性卷声明 (PVC) 导出的卷。某些清单还包含标头 secret，它授予对端点的访问权限，以 OpenShift Virtualization 可以使用的格式导入磁盘镜像。

表 3.6. VM 导出命令

命令	描述
virtctl vmexport create <vmexport_name> --vm|snapshot|pvc= <object_name>	创建一个 **VirtualMachineExport** 自定义资源 (CR) 来从虚拟机、虚拟机快照或 PVC 导出卷。 ● **--vm**: 导出虚拟机的 PVC。 ● **--snapshot** ：导出 **VirtualMachineSnapshot** CR 中包含的 PVC。 ● **--pvc**: 导出 PVC。 ● 可选： **--ttl=1h** 指定生存时间。默认持续时间为 2 小时。
virtctl vmexport delete <vmexport_name>	手动删除 **VirtualMachineExport** CR。
virtctl vmexport download <vmexport_name> --output= <output_file> --volume= <volume_name>	下载在 **VirtualMachineExport** CR 中定义的卷。 ● **--output** 指定文件格式。示例： **disk.img.gz**. ● **--volume** 指定要下载的卷。如果只有一个卷可用，则此标志是可选的。 可选： ● **--keep-vme** 在下载后保留 **VirtualMachineExport** CR。默认的行为是在下载后删除 **VirtualMachineExport** CR 的默认行为。 ● **--insecure** 启用不安全的 HTTP 连接。

命令	描述
virtctl vmexport download <vmexport_name> --<vm|snapshot|pvc>=<object_name> --output=<output_file> --volume=<volume_name>	创建一个 **VirtualMachineExport** CR，然后下载 CR 中定义的卷。
virtctl vmexport download export --manifest	检索一个现有导出的清单。清单不包括标头 secret。
virtctl vmexport download export --manifest --vm=example	为虚拟机创建虚拟机导出，并检索清单。清单不包括标头 secret。
virtctl vmexport download export --manifest --snap=example	为虚拟机快照示例创建虚拟机导出，并检索清单。清单不包括标头 secret。
virtctl vmexport download export --manifest --include-secret	检索一个现有导出的清单。清单包括标头 secret。
virtctl vmexport download export --manifest --manifest-output-format=json	以 json 格式检索现有导出的清单。清单不包括标头 secret。
virtctl vmexport download export --manifest --include-secret --output=manifest.yaml	检索一个现有导出的清单。清单包括标头 secret，并将其写入指定的文件中。

3.2.2.7. VM 内存转储命令

您可使用 **virtctl memory-dump** 命令在 PVC 上输出虚拟机 (VM) 内存转储。您可以指定现有的 PVC，或使用 **--create-claim** 标志来创建新 PVC。

先决条件

- PVC 卷模式必须是 **FileSystem**。
- PVC 必须足够大以保存内存转储。
  计算 PVC 大小的公式为 **(VMMemorySize + 100Mi)* FileSystemOverhead**，其中 **100Mi** 是内存转储开销。
- 您必须运行以下命令来在 **HyperConverged** 自定义资源中启用热插功能：

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op": "add", "path": "/spec/featureGates", \
  "value": "HotplugVolumes"}]'
```

下载内存转储

您必须使用 **virtctl vmexport download** 命令下载内存转储：

```
$ virtctl vmexport download <vmexport_name> --vm|pvc=<object_name> \
  --volume=<volume_name> --output=<output_file>
```

表 3.7. VM 内存转储命令

命令	描述

virtctl memory-dump get <vm_name> --claim-name= <pvc_name>	在 PVC 上保存虚拟机的内存转储。内存转储状态显示在 **VirtualMachine** 资源的 **status** 部分。
	
	可选：
	
	• **--create-claim** 会创建一个具有适当大小的新 PVC。这个标志有以下选项：
	  ◦ **--storage-class=<storage_class>**: 为 PVC 指定存储类。
	  ◦ **--access-mode=<access_mode>**: 指定 **ReadWriteOnce** 或 **ReadWriteMany**。

virtctl memory-dump get <vm_name>	使用相同的 PVC 重新运行 **virtctl memory-dump** 命令。 这个命令覆盖以前的内存转储。
virtctl memory-dump remove <vm_name>	删除内存转储。 如果要更改目标 PVC，则必须手动删除内存转储。 这个命令会删除虚拟机和 PVC 之间的关联，以便在 **VirtualMachine** 资源的 **status** 部分中不会显示内存转储。PVC 不受影响。

3.2.2.8. 热插和热拔命令

您可使用 **virtctl** 从正在运行的虚拟机(VM)和虚拟机实例(VMI)中添加或删除资源。

表 3.8. 热插和热拔命令

命令	描述
virtctl addvolume <vm_name> --volume-name= <datavolume_or_PVC> [--persist] [--serial=<label>]	热插数据卷或持久性卷声明 (PVC)。 可选： • **--persist** 在虚拟机上永久挂载虚拟磁盘。**这个标志不适用于 VMI。** • **--serial=<label>** 为虚拟机添加一个标签。如果没有指定标签，则默认标签是数据卷或 PVC 名称。

命令	描述
virtctl removevolume <vm_name> --volume-name=<virtual_disk>	热拔虚拟磁盘。
virtctl addinterface <vm_name> --network-attachment-definition-name <net_attach_def_name> --name <interface_name>	热插 Linux 网桥网络接口。
virtctl removeinterface <vm_name> --name <interface_name>	热拔 Linux 网桥网络接口。

3.2.2.9. 镜像上传命令

您可使用 **virtctl image-upload** 命令将虚拟机镜像上传到数据卷中。

表 3.9. 镜像上传命令

命令	描述
virtctl image-upload dv <datavolume_name> --image-path= </path/to/image> --no-create	将虚拟机镜像上传到已存在的数据卷中。
virtctl image-upload dv <datavolume_name> --size= <datavolume_size> --image-path=</path/to/image>	将虚拟机镜像上传到指定请求大小的新数据卷中。

3.2.3. 使用 virtctl 部署 libguestfs

您可以使用 **virtctl guestfs** 命令部署带有 **libguestfs-tools** 以及附加到它的持久性卷声明 (PVC) 的交互式容器。

流程

- 要部署一个带有 **libguestfs-tools** 的容器，挂载 PVC 并为其附加一个 shell，运行以下命令：

```
$ virtctl guestfs -n <namespace> <pvc_name> 1
```

1. PVC 名称是必需的参数。如果没有包括它，则会出现错误消息。

3.2.3.1. libguestfs 和 virtctl guestfs 命令

libguestfs 工具可帮助您访问和修改虚拟机 (VM) 磁盘镜像。您可以使用 **libguestfs** 工具查看和编辑客户机中的文件、克隆和构建虚拟机，以及格式化和调整磁盘大小。

您还可以使用 **virtctl guestfs** 命令及其子命令在 PVC 上修改、检查和调试虚拟机磁盘。要查看可能子命令的完整列表，请在命令行中输入 **virt-** 并按 Tab 键。例如：

命令	描述
virt-edit -a /dev/vda /etc/motd	在终端中以交互方式编辑文件。
virt-customize -a /dev/vda --ssh-inject root:string:<public key example>	将 ssh 密钥注入客户系统并创建登录。
virt-df -a /dev/vda -h	查看虚拟机使用了多少磁盘空间。
virt-customize -a /dev/vda --run-command 'rpm -qa > /rpm-list'	通过创建包含完整列表的输出文件，查看虚拟客户机上安装的所有 RPM 的完整列表。
virt-cat -a /dev/vda /rpm-list	在终端中使用 **virt-customize -a /dev/vda --run-command 'rpm -qa > /rpm-list'** 命令显示创建的所有 RPM 的输出文件列表。
virt-sysprep -a /dev/vda	封装要用作模板的虚拟机磁盘镜像。

默认情况下，**virtctl guestfs** 会创建一个会话，其中包含管理 VM 磁盘所需的一切内容。但是，如果想要自定义行为，该命令还支持几个标志选项：

标记选项	描述
--h 或 **--help**	为 **guestfs** 提供帮助.
带有 **<pvc_name>** 参数的 **-n <namespace>** 选项	使用特定命名空间中的 PVC。 如果不使用 **-n <namespace>** 选项，则使用您的当前项目。要更改项目，请使用 **oc project <namespace>**。 如果没有包括 **<pvc_name>** 参数，则会出现错误消息。
--image string	列出 **libguestfs-tools** 容器镜像。 您可以使用 **--image** 选项，将容器配置为使用自定义镜像。

标记选项	描述
--kvm	代表 **libguestfs-tools** 容器使用 **kvm**。 默认情况下，**virtctl guestfs** 为交互式容器设置 **kvm**，这可显著加快 **libguest-tools** 执行，因为它使用了 QEMU。 如果群集没有任何 **kvm** 支持节点，您必须通过设置 **--kvm=false** 选项来禁用 **kvm**。 如果没有设置，**libguestfs-tools** pod 将保持待处理状态，因为它无法调度到任何节点上。
--pull-policy string	显示 **libguestfs** 镜像的拉取策略。 您还可以通过设置 **pull-policy** 选项来覆盖镜像的 pull 策略。

这个命令还会检查 PVC 是否被另一个 pod 使用，这时会出现错误消息。但是，**libguestfs-tools** 进程启动后，设置无法避免使用相同的 PVC 的新 pod。在启动虚拟机访问同一 PVC 前，您必须先验证没有活跃的 **virtctl guestfs** pod。

注意

virtctl guestfs 命令只接受附加到交互式 pod 的单个 PVC。

3.2.4. 使用 Ansible

要使用 OpenShift Virtualization 的 Ansible 集合，请参阅 Red Hat Ansible Automation Hub（Red Hat Hybrid Cloud Console）。

第 4 章 安装

4.1. 为 OPENSHIFT VIRTUALIZATION 准备集群

在安装 OpenShift Virtualization 前，参阅这个部分以确保集群满足要求。

重要

安装方法注意事项

您可以使用任何安装方法（包括用户置备的、安装程序置备或辅助安装程序）来部署 OpenShift Container Platform。但是，安装方法和集群拓扑可能会影响 OpenShift Virtualization 功能，如快照或实时迁移。

Red Hat OpenShift Data Foundation

如果使用 Red Hat OpenShift Data Foundation 部署 OpenShift Virtualization，您必须为 Windows 虚拟机磁盘创建一个专用存储类。详情请参阅为 Windows 虚拟机优化 ODF PersistentVolume。

IPv6

您无法在单堆栈 IPv6 集群上运行 OpenShift Virtualization。

FIPS 模式

如果以 FIPS 模式安装集群，则 OpenShift Virtualization 不需要额外的设置。

4.1.1. 支持的平台

您可以在 OpenShift Virtualization 中使用以下平台：

- 内部裸机服务器。请参阅为 OpenShift Virtualization 规划裸机集群。
- Amazon Web Services 裸机实例。请参阅使用自定义在 AWS 上安装集群。
- IBM Cloud® 裸机服务器。请参阅在 IBM Cloud® Bare Metal 节点上部署 OpenShift Virtualization。

重要

在 IBM Cloud® 裸机服务器上安装 OpenShift Virtualization 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。

不支持由其他云供应商提供的裸机实例或服务器。

4.1.1.1. AWS 裸机上的 OpenShift Virtualization

您可以在 Amazon Web Services (AWS) 裸机 OpenShift Container Platform 集群上运行 OpenShift Virtualization。

注意

OpenShift Virtualization 也支持 Red Hat OpenShift Service on AWS (ROSA) Classic 集群，其配置要求与 AWS 裸机集群相同。

在设置集群前，请查看以下支持的功能和限制概述：

安装

- 您可以使用安装程序置备的基础架构安装集群，确保为 worker 节点指定裸机实例类型。例如，您可以对基于 x86_64 架构的机器使用 **c5n.metal** 类型值。您可以通过编辑 **install-config.yaml** 文件来指定裸机实例类型。
  如需更多信息，请参阅在 AWS 上安装 OpenShift Container Platform 文档。

访问虚拟机 (VM)

- 使用 **virtctl** CLI 工具或 OpenShift Container Platform Web 控制台没有更改您如何访问虚拟机。
- 您可以使用 **NodePort** 或 **LoadBalancer** 服务公开虚拟机。
  - 负载均衡器方法是首选的，因为 OpenShift Container Platform 会在 AWS 中自动创建负载均衡器并管理其生命周期。另外，还会为负载均衡器创建一个安全组，您可以使用注解来附加现有的安全组。删除服务时，OpenShift Container Platform 会移除负载均衡器及其关联的资源。

网络

- 您不能使用单根 I/O 虚拟化 (SR-IOV) 或桥接 Container Network Interface (CNI) 网络，包括虚拟 LAN (VLAN)。如果您的应用程序需要扁平第 2 层网络或对 IP 池进行控制，请考虑使用 OVN-Kubernetes 二级覆盖网络。

Storage

- 您可以使用存储厂商认证的任何存储解决方案与底层平台一起使用。

重要

AWS 裸机和 ROSA 集群可能有不同的存储解决方案。确保您确认支持您的存储供应商。

- 在 OpenShift Virtualization 中使用 Amazon Elastic File System (EFS)或 Amazon Elastic Block Store (EBS)可能会导致性能和功能限制，如下表所示：

表 4.1. EFS 和 EBS 性能和功能限制

功能	EBS 卷			EFS 卷	共享存储解决方案
	gp2	gp3	io2		
VM 实时迁移	不可用	不可用	Available	Available	Available

功能	EBS 卷	EFS 卷	共享存储解决方案
使用克隆快速创建虚拟机	Available	不可用	Available
使用快照进行虚拟机备份和恢复	Available	不可用	Available

考虑使用支持 ReadWriteMany (RWX)、克隆和快照的 CSI 存储来启用实时迁移、快速虚拟机创建和虚拟机快照功能。

托管 control plane (HCP)

- 目前 AWS 基础架构不支持 OpenShift Virtualization 的 HCP。

其他资源

- 将虚拟机连接到 OVN-Kubernetes 二级网络
- 使用服务公开虚拟机

4.1.2. 硬件和操作系统要求

查看 OpenShift Virtualization 的以下硬件和操作系统要求。

4.1.2.1. CPU 要求

- 由 Red Hat Enterprise Linux (RHEL) 9 支持。
参阅用于支持的 CPU 红帽生态系统目录。

注意

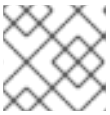
如果您的 worker 节点有不同的 CPU，则可能会出现实时迁移失败，因为不同的 CPU 具有不同的功能。您可以通过确保 worker 节点具有适当容量的 CPU，并为虚拟机配置节点关联性规则来缓解这个问题。

详情请参阅配置所需的节点关联性规则。

- 支持 AMD 和 Intel 64 位架构 (x86-64-v2)。
- 支持 Intel 64 或 AMD64 CPU 扩展。
- 启用 Intel VT 或 AMD-V 硬件虚拟化扩展。
- 启用 NX （无执行）标记。

4.1.2.2. 操作系统要求

- 在 worker 节点上安装的 Red Hat Enterprise Linux CoreOS (RHCOS)。
详情请参阅 RHCOS。



注意

不支持 RHEL worker 节点。

4.1.2.3. 存储要求

- OpenShift Container Platform 支持。请参阅优化存储。
- 您必须创建一个默认的 OpenShift Virtualization 或 OpenShift Container Platform 存储类。这样做的目的是解决虚拟机工作负载的唯一存储需求，并提供优化的性能、可靠性和用户体验。如果 OpenShift Virtualization 和 OpenShift Container Platform 默认存储类都存在，则 OpenShift Virtualization 类在创建虚拟机磁盘时具有优先权。

注意

要将存储类标记为虚拟化工作负载的默认值，请将注解 **storageclass.kubevirt.io/is-default-virt-class** 设置为 **"true"**。

- 如果存储置备程序支持快照，您必须将 **VolumeSnapshotClass** 对象与默认存储类关联。

4.1.2.3.1. 关于虚拟机磁盘的卷和访问模式

如果您将存储 API 与已知的存储供应商搭配使用，则会自动选择卷和访问模式。但是，如果您使用没有存储配置集的存储类，您必须配置卷和访问模式。

要获得最佳结果，请使用 **ReadWriteMany** (RWX)访问模式和 **Block** 卷模式。这一点非常重要：

- 实时迁移需要 **ReadWriteMany** (RWX) 访问模式。
- 块 卷模式的性能优于 **Filesystem** 卷模式。这是因为 **Filesystem** 卷模式使用更多存储层，包括文件系统层和磁盘镜像文件。虚拟机磁盘存储不需要这些层。
  例如，如果您使用 Red Hat OpenShift Data Foundation，Ceph RBD 卷优先于 CephFS 卷。

重要

您不能使用以下配置实时迁移虚拟机：

- 具有 **ReadWriteOnce** (RWO) 访问模式的存储卷
- 透传功能，比如 GPU

对于这些虚拟机，将 **evictionStrategy** 字段设置为 **None**。**None** 策略会在节点重启过程中关闭虚拟机。

4.1.3. 实时迁移要求

- 使用 **ReadWriteMany** (RWX)访问模式的共享存储.
- 足够的 RAM 和网络带宽。

注意

您必须确保集群中有足够的内存请求容量来支持节点排空会导致实时迁移。您可以使用以下计算来确定大约所需的备用内存：

> Product of (Maximum number of nodes that can drain in parallel) and (Highest total VM memory request allocations across nodes)

集群中可以并行运行的迁移数量的默认值为 5。

- 如果虚拟机使用主机型号 CPU，则节点必须支持虚拟机的主机型号 CPU。
- 强烈建议使用专用的 Multus 网络进行实时迁移。专用网络可最小化迁移期间对租户工作负载网络饱和的影响。

4.1.4. 物理资源开销要求

OpenShift Virtualization 是 OpenShift Container Platform 的一个附加组件，它会带来额外的开销。除了 OpenShift Container Platform 要求外，每个集群机器都必须满足以下开销要求。覆盖集群中的物理资源可能会影响性能。

重要

本文档中给出的数字基于红帽的测试方法和设置。这些数字会根据您自己的设置和环境而有所不同。

内存开销
使用以下因素计算 OpenShift Virtualization 的内存开销值。

集群内存开销

> Memory overhead per infrastructure node ≈ 150 MiB

> Memory overhead per worker node ≈ 360 MiB

另外，OpenShift Virtualization 环境资源需要总计 2179 MiB 的内存，分布到所有基础架构节点。

虚拟机内存开销

```
Memory overhead per virtual machine ≈ (1.002 × requested memory) \
        + 218 MiB \ 1
        + 8 MiB × (number of vCPUs) \ 2
        + 16 MiB × (number of graphics devices) \ 3
        + (additional memory overhead) 4
```

1. 在 **virt-launcher** pod 中运行的进程需要。
2. 虚拟机请求的虚拟 CPU 数量。
3. 虚拟机请求的虚拟图形卡数。
4. 额外的内存开销：

- 如果您的环境包含单一根 I/O 虚拟化（SR-IOV）网络设备或图形处理单元（GPU），请为每个设备分配 1 GiB 额外的内存开销。
- 如果启用了安全加密虚拟化 (SEV)，请添加 256 MiB。
- 如果启用了受信任的平台模块 (TPM)，请添加 53 MiB。

CPU 开销
使用以下内容计算 OpenShift Virtualization 的集群处理器开销要求。每个虚拟机的 CPU 开销取决于您的单独设置。

集群 CPU 开销

```
CPU overhead for infrastructure nodes ≈ 4 cores
```

OpenShift Virtualization 增加集群级别服务的整体使用，如日志记录、路由和监控。要考虑这个工作负载，请确保托管基础架构组件的节点分配了用于不同节点的 4 个额外内核（4000 毫秒）的容量。

```
CPU overhead for worker nodes ≈ 2 cores + CPU overhead per virtual machine
```

除了虚拟机工作负载所需的 CPU 外，每个托管虚拟机的 worker 节点都必须有 2 个额外内核（2000 毫秒）用于 OpenShift Virtualization 管理工作负载。

虚拟机 CPU 开销

如果请求专用 CPU，则会对集群 CPU 开销要求有 1:1 影响。否则，没有有关虚拟机所需 CPU 数量的具体规则。

存储开销
使用以下指南来估算 OpenShift Virtualization 环境的存储开销要求。

集群存储开销

```
Aggregated storage overhead per node ≈ 10 GiB
```

10 GiB 在安装 OpenShift Virtualization 时，集群中每个节点的磁盘存储影响估计值。

虚拟机存储开销

每个虚拟机的存储开销取决于虚拟机内的具体资源分配请求。该请求可能用于集群中其他位置托管的节点或存储资源的临时存储。OpenShift Virtualization 目前不会为正在运行的容器本身分配任何额外的临时存储。

Example

作为集群管理员，如果您计划托管集群中的 10 个虚拟机，每个虚拟机都有 1 GiB RAM 和 2 个 vCPU，集群中的内存影响为 11.68 GiB。集群中每个节点的磁盘存储影响估算为 10 GiB，托管虚拟机工作负载的 worker 节点的 CPU 影响最小 2 个内核。

4.1.5. 单节点 Openshift 的不同

您可以在单节点 OpenShift 上安装 OpenShift Virtualization。

但是，您应该注意单节点 OpenShift 不支持以下功能：

- 高可用性
- Pod 中断预算
- 实时迁移
- 配置了驱除策略的虚拟机或模板

其他资源

- OpenShift Container Platform 存储的常见术语表

4.1.6. 对象最大值

在规划集群时，您必须考虑以下测试的对象最大值：

- OpenShift Container Platform 对象最大值
- OpenShift Virtualization 对象最大值

4.1.7. 集群高可用性选项

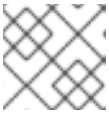
您可以为集群配置以下高可用性(HA)选项之一：

- 通过 部署机器健康检查，可以使用 安装程序置备的基础架构 (IPI)自动高可用性。

注意

在使用安装程序置备的基础架构安装的 OpenShift Container Platform 集群中，并使用正确配置的 **MachineHealthCheck** 资源，如果节点无法进行机器健康检查，且对集群不可用，则会回收它。在故障节点上运行的虚拟机之后会发生什么，这取决于一系列条件。如需有关潜在结果以及运行策略如何影响这些结果的详细信息，请参阅运行策略。

- 通过在 OpenShift Container Platform 集群上使用 **Node Health Check Operator** 来部署 **NodeHealthCheck** 控制器，可以使用 IPI 和非 IPI 自动高可用性。控制器识别不健康的节点并使用补救供应商，如 Self Node Remediation Operator 或 Fence Agents Remediation Operator 来修复不健康的节点。如需有关补救、隔离和维护节点的更多信息，请参阅 Red Hat OpenShift 文档中的工作负载可用性。
- 任何平台的高可用性可通过使用监控系统或合格的人类监控节点可用性来实现。当节点丢失时，关闭并运行 **oc delete node <lost_node>**。

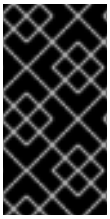


注意

如果没有外部监控系统或合格的人类监控节点运行状况，虚拟机就失去高可用性。

4.2. 安装 OPENSHIFT VIRTUALIZATION

安装 OpenShift Virtualization 以便在 OpenShift Container Platform 集群中添加虚拟化功能。



重要

如果在没有互联网连接的受限环境中安装 OpenShift Virtualization，您必须为受限网络配置 Operator Lifecycle Manager (OLM)。

如果您有有限的互联网连接，您可以在 OLM 中配置代理支持，以访问 OperatorHub。

4.2.1. 安装 OpenShift Virtualization Operator

使用 OpenShift Container Platform Web 控制台或命令行安装 OpenShift Virtualization Operator。

4.2.1.1. 使用 Web 控制台安装 OpenShift Virtualization Operator

您可以使用 OpenShift Container Platform Web 控制台部署 OpenShift Virtualization Operator。

先决条件

- 在集群上安装 OpenShift Container Platform 4.16。
- 以具有 **cluster-admin** 权限的用户身份登录到 OpenShift Container Platform web 控制台。

流程

1. 从 **Administrator** 视角中，点 **Operators → OperatorHub**。
2. 在 **Filter by keyword** 字段中，键入 **Virtualization**。
3. 选择带有 **Red Hat** source 标签的 **OpenShift Virtualization Operator** 标题。
4. 阅读 Operator 信息并单击 **Install**。
5. 在 **Install Operator** 页面中：
   a. 从可用 **Update Channel** 选项列表中选择 **stable**。这样可确保安装与 OpenShift Container Platform 版本兼容的 OpenShift Virtualization 版本。
   b. **对于安装的命名空间**，请确保选择了 **Operator 推荐的命名空间**选项。这会在 **openshift-cnv** 命名空间中安装 Operator，该命名空间在不存在时自动创建。

警告

尝试在 **openshift-cnv** 以外的命名空间中安装 OpenShift Virtualization Operator 会导致安装失败。

   c. 对于 **Approval Strategy**，强烈建议您选择 **Automatic** （默认值），以便在 **stable** 更新频道中提供新版本时 OpenShift Virtualization 会自动更新。
   虽然可以选择 **Manual** 批准策略，但这不可取，因为它会给集群提供支持和功能带来高风险。只有在您完全了解这些风险且无法使用 **Automatic** 时，才选择 **Manual**。

警告

因为 OpenShift Virtualization 只在与对应的 OpenShift Container Platform 版本搭配使用时被支持，所以缺少的 OpenShift Virtualization 更新可能会导致您的集群不被支持。

6. 点击 **Install** 使 Operator 可供 **openshift-cnv** 命名空间使用。
7. 当 Operator 成功安装时，点 **Create HyperConverged**。
8. 可选：为 OpenShift Virtualization 组件配置 **Infra** 和 **Workloads** 节点放置选项。
9. 点击 **Create** 启动 OpenShift Virtualization。

验证

- 导航到 **Workloads → Pods** 页面，并监控 OpenShift Virtualization Pod，直至全部处于 **Running** 状态。在所有 pod 都处于 **Running** 状态后，您可以使用 OpenShift Virtualization。

4.2.1.2. 使用命令行安装 OpenShift Virtualization Operator

订阅 OpenShift Virtualization 目录，并通过将清单应用到集群来安装 OpenShift Virtualization Operator。

4.2.1.2.1. 使用 CLI 订阅 OpenShift virtualization 目录

在安装 OpenShift Virtualization 前，需要订阅到 OpenShift Virtualization catalog。订阅会授予 OpenShift virtualization Operator 对 **openshift-cnv** 命名空间的访问权限。

为了订阅，在您的集群中应用一个单独的清单（manifest）来配置 **Namespace**、**OperatorGroup** 和 **Subscription** 对象。

先决条件

- 在集群上安装 OpenShift Container Platform 4.16。
- 安装 OpenShift CLI (**oc**) 。
- 以具有 **cluster-admin** 特权的用户身份登录。

流程

1. 创建一个包含以下清单的 YAML 文件：

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-cnv
---
apiVersion: operators.coreos.com/v1
```

```
kind: OperatorGroup
metadata:
  name: kubevirt-hyperconverged-group
  namespace: openshift-cnv
spec:
  targetNamespaces:
    - openshift-cnv
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.16.3
  channel: "stable" (1)
```

1 使用 **stable** 频道可确保您安装与 OpenShift Container Platform 版本兼容的 OpenShift Virtualization 版本。

2. 运行以下命令,为 OpenShift Virtualization 创建所需的 **Namespace**、**OperatorGroup** 和 **Subscription**对象：

```
$ oc apply -f <file name>.yaml
```

注意

您可以在 YAML 文件中配置证书轮转参数。

4.2.1.2.2. 使用 CLI 部署 OpenShift Virtualization Operator

您可以使用 **oc** CLI 部署 OpenShift Virtualization Operator。

先决条件

- 在 **openshift-cnv** 命名空间中订阅 OpenShift Virtualization 目录。
- 以具有 **cluster-admin** 特权的用户身份登录。

流程

1. 创建一个包含以下清单的 YAML 文件：

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
```

2. 运行以下命令来部署 OpenShift Virtualization Operator:

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

验证

- 通过观察 **openshift-cnv** 命名空间中集群服务版本（CSV）的 **PHASE** 来确保 OpenShift Virtualization 已被成功部署。运行以下命令：

```
$ watch oc get csv -n openshift-cnv
```

如果部署成功，则会显示以下输出：

输出示例

```
NAME                                       DISPLAY                    VERSION   REPLACES   PHASE
kubevirt-hyperconverged-operator.v4.16.3   OpenShift Virtualization   4.16.3
Succeeded
```

4.2.2. 后续步骤

- hostpath 置备程序是设计用于 OpenShift Virtualization 的本地存储置备程序。如果要为虚拟机配置本地存储，您必须首先启用 hostpath 置备程序。

4.3. 卸载 OPENSHIFT VIRTUALIZATION

您可以使用 Web 控制台或命令行界面 (CLI) 卸载 OpenShift Virtualization，以删除 OpenShift Virtualization 工作负载、Operator 及其资源。

4.3.1. 使用 Web 控制台卸载 OpenShift Virtualization

您可以使用 Web 控制台卸载 OpenShift Virtualization 来执行以下任务：

1. 删除 **HyperConverged** CR。
2. 删除 OpenShift Virtualization Operator。
3. 删除 **openshift-cnv** 命名空间。
4. 删除 OpenShift Virtualization 自定义资源定义 (CRD)。

重要

您必须首先删除所有 虚拟机，以及 虚拟机实例。

当其工作负载保留在集群中时，您无法卸载 OpenShift Virtualization。

4.3.1.1. 删除 HyperConverged 自定义资源

要卸载 OpenShift Virtualization，首先删除 **HyperConverged** 自定义资源 (CR)。

先决条件

- 可以使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。

流程

1. 进入到 **Operators → Installed Operators** 页面。
2. 选择 OpenShift Virtualization Operator。
3. 点 **OpenShift Virtualization Deployment** 选项卡。
4. 点 **kubevirt-hyperconverged** 旁边的 Options 菜单 ，然后选择 **Delete HyperConverged**。
5. 在确认窗口中点击 **Delete**。

4.3.1.2. 使用 Web 控制台从集群中删除 Operator

集群管理员可以使用 Web 控制台从所选命名空间中删除已安装的 Operator。

先决条件

- 您可以使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群 Web 控制台。

流程

1. 进入到 **Operators → Installed Operators** 页面。
2. 在 **Filter by name** 字段中滚动或输入关键字以查找您要删除的 Operator。然后点它。
3. 在 **Operator Details** 页面右侧，从 **Actions** 列表中选择 **Uninstall Operator**。
   此时会显示 **Uninstall Operator?** 对话框。
4. 选择 **Uninstall** 来删除 Operator、Operator 部署和 pod。按照此操作，Operator 将停止运行，不再接收更新。

注意

此操作不会删除 Operator 管理的资源，包括自定义资源定义 (CRD) 和自定义资源 (CR) 。Web 控制台和继续运行的集群资源启用的仪表板和导航项可能需要手动清理。要在卸载 Operator 后删除这些，您可能需要手动删除 Operator CRD。

4.3.1.3. 使用 web 控制台删除命令空间

您可以使用 OpenShift Container Platform web 控制台删除一个命名空间。

先决条件

- 可以使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。

流程

1. 导航至 **Administration → Namespaces**。

2. 在命名空间列表中找到您要删除的命名空间。

3. 在命名空间列表的右侧，从 Options 菜单  中选择 **Delete Namespace**。

4. 当 **Delete Namespace** 页打开时，在相关项中输入您要删除的命名空间的名称。

5. 点击 **Delete**。

4.3.1.4. 删除 OpenShift Virtualization 自定义资源定义

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

先决条件

- 可以使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。

流程

1. 进入到 **Administration → CustomResourceDefinitions**。
2. 选择 **Label** 过滤器，并在 **Search** 字段中输入 **operators.coreos.com/kubevirt-hyperconverged.openshift-cnv**，以显示 OpenShift Virtualization CRD。
3. 点每个 CRD 旁边的 Options 菜单  ，然后选择 **Delete CustomResourceDefinition**。

4.3.2. 使用 CLI 卸载 OpenShift Virtualization

您可以使用 OpenShift CLI (**oc**) 卸载 OpenShift Virtualization。

先决条件

- 可以使用具有 **cluster-admin** 权限的账户访问 OpenShift Container Platform 集群。
- 已安装 OpenShift CLI(**oc**)。
- 您已删除所有虚拟机和虚拟机实例。当其工作负载保留在集群中时，您无法卸载 OpenShift Virtualization。

流程

1. 删除 **HyperConverged** 自定义资源：

```
$ oc delete HyperConverged kubevirt-hyperconverged -n openshift-cnv
```

2. 删除 OpenShift Virtualization Operator 订阅：

```
$ oc delete subscription kubevirt-hyperconverged -n openshift-cnv
```

3. 删除 OpenShift Virtualization **ClusterServiceVersion** 资源：

```
$ oc delete csv -n openshift-cnv -l operators.coreos.com/kubevirt-hyperconverged.openshift-cnv
```

4. 删除 OpenShift Virtualization 命名空间：

```
$ oc delete namespace openshift-cnv
```

5. 使用 **dry-run** 选项运行 **oc delete crd** 命令列出 OpenShift Virtualization 自定义资源定义 (CRD)：

```
$ oc delete crd --dry-run=client -l operators.coreos.com/kubevirt-hyperconverged.openshift-cnv
```

输出示例

```
customresourcedefinition.apiextensions.k8s.io "cdis.cdi.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "hostpathprovisioners.hostpathprovisioner.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "hyperconvergeds.hco.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "kubevirts.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "networkaddonsconfigs.networkaddonsoperator.network.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "ssps.ssp.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "tektontasks.tektontasks.kubevirt.io" deleted (dry run)
```

6. 运行 **oc delete crd** 命令来删除 CRD，而无需 **dry-run** 选项：

```
$ oc delete crd -l operators.coreos.com/kubevirt-hyperconverged.openshift-cnv
```

其他资源

- 删除虚拟机
- 删除虚拟机实例

第 5 章 安装后配置

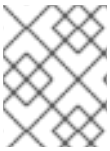
5.1. 安装后配置

以下流程通常在安装 OpenShift Virtualization 后执行。您可以配置与环境相关的组件：

- OpenShift Virtualization Operator、工作负载和控制器的节点放置规则
- 网络配置 ：
  - 安装 Kubernetes NMState 和 SR-IOV Operator
  - 配置 Linux 网桥网络以从外部访问虚拟机(VM)
  - 为实时迁移配置专用的二级网络
  - 配置 SR-IOV 网络
  - 使用 OpenShift Container Platform Web 控制台启用创建负载均衡器服务
- 存储配置 ：
  - 为 Container Storage Interface (CSI) 定义默认存储类
  - 使用 Hostpath Provisioner (HPP) 配置本地存储

5.2. 为 OPENSHIFT VIRTUALIZATION 组件指定节点

裸机节点上虚拟机(VM)的默认调度是适当的。另外，您可以通过配置节点放置规则来指定您要部署 OpenShift Virtualization Operator、工作负载和控制器的节点。



注意

在安装 OpenShift Virtualization 后，您可以为一些组件配置节点放置规则，但如果要为工作负载配置节点放置规则，则虚拟机将无法被存在。

5.2.1. 关于 OpenShift Virtualization 组件的节点放置规则

您可以将节点放置规则用于以下任务：

- 仅在用于虚拟化工作负载的节点上部署虚拟机。
- 仅在基础架构节点上部署 Operator。
- 在工作负载之间保持隔离。

根据对象，您可以使用以下一个或多个规则类型：

nodeSelector

允许将 Pod 调度到使用您在此字段中指定的键值对标记的节点上。节点必须具有与所有列出的对完全匹配的标签。

关联性

可让您使用更宽松的语法来设置与 pod 匹配的规则。关联性也允许在规则应用方面更加精细。例如，您可以指定规则是首选项，而不是要求。如果规则是首选项的，则在不满足规则时仍然会调度 pod。

容限（tolerations）

允许将 pod 调度到具有匹配污点的节点。如果某个节点有污点（taint），则该节点只接受容许该污点的 pod。

5.2.2. 应用节点放置规则

您可以使用命令行编辑 **Subscription**、**HyperConverged** 或 **HostPathProvisioner** 对象来应用节点放置规则。

先决条件

- 已安装 **oc** CLI 工具。
- 使用集群管理员权限登录。

流程

1. 运行以下命令，在默认编辑器中编辑对象：

```
$ oc edit <resource_type> <resource_name> -n {CNVNamespace}
```

2. 保存文件以使改变生效。

5.2.3. 节点放置规则示例

您可以通过编辑 **Subscription**、**HyperConverged** 或 **HostPathProvisioner** 对象来为 OpenShift Virtualization 组件指定节点放置规则。

5.2.3.1. 订阅对象节点放置规则示例

要指定 OLM 部署 OpenShift Virtualization Operator 的节点，在 OpenShift Virtualization 安装过程中编辑 **Subscription** 对象。

目前，您无法使用 Web 控制台为 **Subscription** 对象配置节点放置规则。

Subscription 对象不支持 **关联性** 节点放置规则。

使用 nodeSelector 规则的 Subscription 对象示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.16.3
  channel: "stable"
  config:
    nodeSelector:
      example.io/example-infra-key: example-infra-value 1
```

1 OLM 在带有 **example.io/example-infra-key = example-infra-value** 的节点上部署 OpenShift Virtualization Operator。

带有 容限 规则的 Subscription 对象示例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source:  redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.16.3
  channel: "stable"
  config:
    tolerations:
    - key: "key"
      operator: "Equal"
      value: "virtualization" (1)
      effect: "NoSchedule"
```

1 OLM 在带有 **key = virtualization:NoSchedule** 污点的节点上部署 OpenShift Virtualization Operator。只有具有匹配容限的 pod 才会调度到这些节点上。

5.2.3.2. HyperConverged 对象节点放置规则示例

要指定 OpenShift Virtualization 部署其组件的节点，您可以在 OpenShift Virtualization 安装过程中创建的 HyperConverged 自定义资源(CR)文件中编辑 **nodePlacement** 对象。

使用 nodeSelector 规则的 HyperConverged 对象示例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  infra:
    nodePlacement:
      nodeSelector:
        example.io/example-infra-key: example-infra-value (1)
  workloads:
    nodePlacement:
      nodeSelector:
        example.io/example-workloads-key: example-workloads-value (2)
```

1 基础架构资源放置在带有 **example.io/example-infra-key = example-infra-value** 的节点上。

2 工作负载放置在带有 **example.io/example-workloads-key = example-workloads-value** 的节点上。

使用 关联性规则的 HyperConverged 对象示例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  infra:
    nodePlacement:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: example.io/example-infra-key
                operator: In
                values:
                - example-infra-value (1)
  workloads:
    nodePlacement:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: example.io/example-workloads-key (2)
                operator: In
                values:
                - example-workloads-value
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 1
            preference:
              matchExpressions:
              - key: example.io/num-cpus
                operator: Gt
                values:
                - 8 (3)
```

1. 基础架构资源放置在标记为 **example.io/example-infra-key = example-value** 的节点上。
2. 工作负载放置在带有 **example.io/example-workloads-key = example-workloads-value** 的节点上。
3. 对于工作负载，最好使用八个以上 CPU 的节点，但如果它们不可用，仍可调度 pod。

带有 容限 规则的 HyperConverged 对象示例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
```

```
workloads:
  nodePlacement:
    tolerations: 1
    - key: "key"
      operator: "Equal"
      value: "virtualization"
      effect: "NoSchedule"
```

1 为 OpenShift Virtualization 组件保留的节点使用 **key = virtualization:NoSchedule** 污点标记。只有具有匹配容限的 pod 才会调度到保留节点上。

5.2.3.3. HostPathProvisioner 对象节点放置规则示例

您可以直接编辑 **HostPathProvisioner** 对象，或使用 Web 控制台。

> **警告**
>
> 您必须将 hostpath 置备程序和 OpenShift Virtualization 组件调度到同一节点上。否则，使用 hostpath 置备程序的虚拟化 pod 无法运行。您无法运行虚拟机。

使用 hostpath 置备程序(HPP)存储类部署虚拟机(VM)后，您可以使用节点选择器从同一节点中删除 hostpath 置备程序 pod。但是，您必须首先恢复该更改，至少针对该特定节点，并在尝试删除虚拟机前等待 pod 运行。

您可以通过为安装 hostpath 置备程序时创建的 **HostPathProvisioner** 对象的 **spec.workload** 字段指定 **nodeSelector**、**affinity** 或 **tolerations** 来配置节点放置规则。

带有 nodeSelector 规则的 HostPathProvisioner 对象示例

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
  pathConfig:
    path: "</path/to/backing/directory>"
    useNamingPrefix: false
  workload:
    nodeSelector:
      example.io/example-workloads-key: example-workloads-value 1
```

1 工作负载放置在带有 **example.io/example-workloads-key = example-workloads-value** 的节点上。

5.2.4. 其他资源

- 为虚拟机指定节点
- 使用节点选择器将 pod 放置到特定节点
- 使用节点关联性规则控制节点上的 pod 放置
- 使用节点污点控制 pod 放置

5.3. 安装后的网络配置

默认情况下，OpenShift Virtualization 安装了一个内部 pod 网络。

安装 OpenShift Virtualization 后，您可以安装网络 Operator 并配置额外网络。

5.3.1. 安装网络 Operator

您必须安装 Kubernetes NMState Operator 来为实时迁移或外部访问虚拟机(VM)配置 Linux 网桥网络。有关安装说明，请参阅使用 Web 控制台安装 Kubernetes NMState Operator 。

您可以安装 SR-IOV Operator 来管理 SR-IOV 网络设备和网络附加。有关安装说明，请参阅 安装 SR-IOV Network Operator。

您可以添加 MetalLB Operator 来管理集群中的 MetalLB 实例的生命周期。有关安装说明，请参阅 使用 Web 控制台从 OperatorHub 安装 MetalLB Operator。

5.3.2. 配置 Linux 网桥网络

安装 Kubernetes NMState Operator 后，您可以为实时迁移或外部访问虚拟机(VM)配置 Linux 网桥网络。

5.3.2.1. 创建 Linux 网桥 NNCP

您可以为 Linux 网桥网络创建一个 **NodeNetworkConfigurationPolicy** (NNCP) 清单。

先决条件

- 已安装 Kubernetes NMState Operator。

流程

- 创建 **NodeNetworkConfigurationPolicy** 清单。本例包含示例值，您必须替换为您自己的信息。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy (1)
spec:
  desiredState:
    interfaces:
      - name: br1 (2)
        description: Linux bridge with eth1 as a port (3)
        type: linux-bridge (4)
        state: up (5)
        ipv4:
```

```
    enabled: false 6
  bridge:
    options:
      stp:
        enabled: false 7
    port:
      - name: eth1 8
```

1. 策略的名称。
2. 接口的名称。
3. 可选：接口人类可读的接口描述。
4. 接口的类型。这个示例会创建一个桥接。
5. 创建后接口的请求状态。
6. 在这个示例中禁用 IPv4。
7. 在这个示例中禁用 STP。
8. 网桥附加到的节点 NIC。

5.3.2.2. 使用 Web 控制台创建 Linux 网桥 NAD

您可以创建一个网络附加定义(NAD)来使用 OpenShift Container Platform web 控制台为 Pod 和虚拟机提供第 2 层网络。

Linux 网桥网络附加定义是将虚拟机连接至 VLAN 的最有效方法。

警告

不支持在虚拟机的网络附加定义中配置 IP 地址管理(IPAM)。

流程

1. 在 Web 控制台中，点 **Networking → NetworkAttachmentDefinitions**。
2. 点 **Create Network Attachment Definition**。

注意

网络附加定义必须与 pod 或虚拟机位于同一个命名空间中。

3. 输入唯一 **Name** 和可选 **Description**。
4. 从 **Network Type** 列表中选择 **CNV Linux 网桥**。

5. 在 **Bridge Name** 字段输入网桥名称。
6. 可选：如果资源配置了 VLAN ID，请在 **VLAN Tag Number** 字段中输入 ID 号。
7. 可选： 选择 **MAC Spoof Check**来启用 MAC spoof 过滤。此功能只允许单个 MAC 地址退出 pod，从而可以防止使用 MAC 欺骗进行的安全攻击。
8. 点 **Create**。

后续步骤

- 将虚拟机(VM)附加到 Linux 网桥网络

5.3.3. 配置网络以进行实时迁移

配置了 Linux 网桥网络后，您可以为实时迁移配置专用网络。专用的网络可最小化实时迁移期间对租户工作负载的网络饱和影响。

5.3.3.1. 为实时迁移配置专用的二级网络

要为实时迁移配置专用的二级网络，您必须首先使用 CLI 创建桥接网络附加定义(NAD)。然后，您可以将 **NetworkAttachmentDefinition** 对象的名称添加到 **HyperConverged** 自定义资源(CR)。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 您以具有 **cluster-admin** 角色的用户身份登录到集群。
- 每个节点至少有两个网络接口卡 (NIC)。
- 用于实时迁移的 NIC 连接到同一 VLAN。

流程

1. 根据以下示例创建 **NetworkAttachmentDefinition** 清单：

配置文件示例

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: my-secondary-network 1
  namespace: openshift-cnv 2
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "migration-bridge",
    "type": "macvlan",
    "master": "eth1", 3
    "mode": "bridge",
    "ipam": {
      "type": "whereabouts", 4
```

```
      "range": "10.200.5.0/24" 5
    }
  }'
```

1 指定 **NetworkAttachmentDefinition** 对象的名称。

2 3 指定要用于实时迁移的 NIC 名称。

4 指定为 NAD 提供网络的 CNI 插件名称。

5 为二级网络指定一个 IP 地址范围。这个范围不得与主网络的 IP 地址重叠。

2. 运行以下命令，在默认编辑器中打开 **HyperConverged** CR：

```
oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

3. 将 **NetworkAttachmentDefinition** 对象的名称添加到 **HyperConverged** CR 的 **spec.liveMigrationConfig** 小节中：

HyperConverged 清单示例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  liveMigrationConfig:
    completionTimeoutPerGiB: 800
    network: <network> 1
    parallelMigrationsPerCluster: 5
    parallelOutboundMigrationsPerNode: 2
    progressTimeout: 150
# ...
```

1 指定要用于实时迁移的 Multus **NetworkAttachmentDefinition** 对象的名称。

4. 保存更改并退出编辑器。**virt-handler** Pod 会重启并连接到二级网络。

验证

- 当运行虚拟机的节点置于维护模式时，虚拟机会自动迁移到集群中的另一个节点。您可以通过检查虚拟机实例(VMI)元数据中的目标 IP 地址，验证迁移是否在二级网络中发生，而不是默认 pod 网络。

```
$ oc get vmi <vmi_name> -o jsonpath='{.status.migrationState.targetNodeAddress}'
```

5.3.3.2. 使用 Web 控制台选择专用网络

您可以使用 OpenShift Container Platform Web 控制台为实时迁移选择一个专用网络。

先决条件

- 为实时迁移配置了 Multus 网络。

流程

1. 在 OpenShift Container Platform web 控制台中进入到 **Virtualization > Overview**。
2. 点 **Settings** 选项卡，然后点 **Live migration**。
3. 从 **Live migration network** 列表中选择网络。

5.3.4. 配置 SR-IOV 网络

安装 SR-IOV Operator 后，您可以配置 SR-IOV 网络。

5.3.4.1. 配置 SR-IOV 网络设备

SR-IOV Network Operator 把 **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CRD 添加到 OpenShift Container Platform。您可以通过创建一个 SriovNetworkNodePolicy 自定义资源 (CR) 来配置 SR-IOV 网络设备。

注意

在应用由 **SriovNetworkNodePolicy** 对象中指定的配置时，SR-IOV Operator 可能会排空节点，并在某些情况下会重启节点。仅在以下情况下重启：

- 使用 Mellanox NIC (**mlx5** 驱动程序)时，当虚拟功能(VF)数量增加时，节点重启都会在物理功能(PF)上增加。
- 使用 Intel NIC 时，只有在内核参数不包含 **intel_iommu=on** 和 **iommu=pt** 时，才会重启。

它可能需要几分钟时间来应用配置更改。

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。
- 已安装 SR-IOV Network Operator。
- 集群中有足够的可用节点，用于处理从排空节点中驱除的工作负载。
- 您还没有为 SR-IOV 网络设备配置选择任何 control plane 节点。

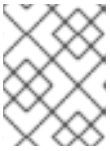
流程

1. 创建一个 **SriovNetworkNodePolicy** 对象，然后在 **<name>-sriov-node-network.yaml** 文件中保存 YAML。使用配置的实际名称替换 **<name>**。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
```

```
spec:
  resourceName: <sriov_resource_name> 3
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" 4
  priority: <priority> 5
  mtu: <mtu> 6
  numVfs: <num> 7
  nicSelector: 8
    vendor: "<vendor_code>" 9
    deviceID: "<device_id>" 10
    pfNames: ["<pf_name>", ...] 11
    rootDevices: ["<pci_bus_id>", "..."] 12
  deviceType: vfio-pci 13
  isRdma: false 14
```

1. 为 CR 对象指定一个名称。
2. 指定 SR-IOV Operator 安装到的命名空间。
3. 指定 SR-IOV 设备插件的资源名称。您可以为一个资源名称创建多个 **SriovNetworkNodePolicy** 对象。
4. 指定节点选择器来选择要配置哪些节点。只有所选节点上的 SR-IOV 网络设备才会被配置。SR-IOV Container Network Interface（CNI）插件和设备插件仅在所选节点上部署。
5. 可选：指定一个 **0** 到 **99** 之间的整数。较小的数值具有较高的优先权，优先级 **10** 高于优先级 **99**。默认值为 **99**。
6. 可选：为虚拟功能（VF）的最大传输单位 (MTU) 指定一个值。最大 MTU 值可能因不同的 NIC 型号而有所不同。
7. 为 SR-IOV 物理网络设备指定要创建的虚拟功能 (VF) 的数量。对于 Intel 网络接口控制器（NIC），VF 的数量不能超过该设备支持的 VF 总数。对于 Mellanox NIC，VF 的数量不能超过 **127**。
8. **nicSelector** 映射为 Operator 选择要配置的以太网设备。您不需要为所有参数指定值。建议您以足够的准确度来识别以太网适配器，以便尽量减小意外选择其他以太网设备的可能性。如果指定了**rootDevices**，则必须同时为 **vendor**、 **deviceID** 或 **pfNames** 指定一个值。如果同时指定了 **pfNames** 和 **rootDevices**，请确保它们指向同一个设备。
9. 可选：指定 SR-IOV 网络设备的厂商十六进制代码。允许的值只能是 **8086** 或 **15b3**。
10. 可选：指定 SR-IOV 网络设备的设备十六进制代码。允许的值只能是 **158b**、**1015**、**1017**。
11. 可选：参数接受包括以太网设备的一个或多个物理功能 (PF) 的数组。
12. 参数接受一个包括一个或多个 PCI 总线地址，用于以太网设备的物理功能的数组。使用以下格式提供地址: **0000:02:00.1**。
13. OpenShift Virtualization 中的虚拟功能需要 **vfio-pci** 驱动程序类型。
14. 可选：指定是否启用远程直接访问（RDMA）模式。对于 Mellanox 卡，请将 **isRdma** 设置为 **false**。默认值为 **false**。



注意

如果将 **RDMA** 标记设定为 **true**，您可以继续使用启用了 RDMA 的 VF 作为普通网络设备。设备可在其中的一个模式中使用。

2. 可选：将 SR-IOV 功能的集群节点标记为 **SriovNetworkNodePolicy.Spec.NodeSelector**（如果它们还没有标记）。有关标记节点的更多信息，请参阅"了解如何更新节点上的标签"。

3. 创建 **SriovNetworkNodePolicy** 对象：

```
$ oc create -f <name>-sriov-node-network.yaml
```

其中 **<name>** 指定这个配置的名称。

在应用配置更新后，**sriov-network-operator** 命名空间中的所有 Pod 都会变为 **Running** 状态。

4. 要验证是否已配置了 SR-IOV 网络设备，请输入以下命令。将 **<node_name>** 替换为带有您刚才配置的 SR-IOV 网络设备的节点名称。

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o jsonpath='{.status.syncStatus}'
```

后续步骤

- 将虚拟机(VM)附加到 SR-IOV 网络

5.3.5. 使用 Web 控制台启用负载均衡器服务创建

您可以使用 OpenShift Container Platform web 控制台为虚拟机(VM)创建负载均衡器服务。

先决条件

- 已为集群配置负载均衡器。
- 以具有 **cluster-admin** 角色的用户身份登录。

流程

1. 进入到 **Virtualization → Overview**。
2. 在 **Settings** 选项卡中，点 **Cluster**。
3. 展开 **General settings** 和 **SSH 配置**。
4. 将 **SSH over LoadBalancer 服务**设置为 on。

5.4. 安装后存储配置

以下存储配置任务是必需的：

- 您必须为集群配置默认存储类。否则，集群无法接收自动引导源更新。
- 如果您的存储供应商没有被 CDI 识别，您必须配置存储配置集。存储配置集根据关联的存储类提供推荐的存储设置。

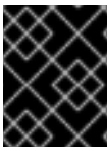
可选： 您可以使用 hostpath 置备程序(HPP)配置本地存储。

如需了解更多选项，请参阅 存储配置概述，包括配置 Containerized Data Importer (CDI)、数据卷和自动引导源更新。

5.4.1. 使用 HPP 配置本地存储

安装 OpenShift Virtualization Operator 时，会自动安装 Hostpath Provisioner(HPP)Operator。HPP Operator 创建 HPP 置备程序。

HPP 是为 OpenShift Virtualization 设计的本地存储置备程序。要使用 HPP，您必须创建一个 HPP 自定义资源(CR)。



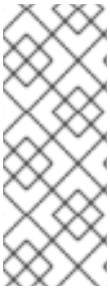
重要

HPP 存储池不能与操作系统位于同一个分区。否则，存储池可能会填满操作系统分区。如果操作系统分区已满，则性能可能会生效，或者节点可能会不稳定或不可用。

5.4.1.1. 使用 storagePools 小节为 CSI 驱动程序创建存储类

要使用 hostpath 置备程序 (HPP)，您必须为 Container Storage Interface (CSI) 驱动程序创建关联的存储类。

当您创建存储类时，您将设置参数，它们会影响属于该存储类的持久性卷(PV)的动态置备。您不能在创建 **StorageClass** 对象后更新其参数。



注意

虚拟机使用基于本地 PV 的数据卷。本地 PV 与特定节点绑定。虽然磁盘镜像准备供虚拟机消耗，但可能不会将虚拟机调度到之前固定本地存储 PV 的节点。

要解决这个问题，使用 Kubernetes pod 调度程序将持久性卷声明(PVC)绑定到正确的节点上的 PV。通过使用 **volumeBindingMode** 参数设置为 **WaitForFirstConsumer** 的 **StorageClass** 值，PV 的绑定和置备会延迟到 pod 使用 PVC。

流程

1. 创建 **storageclass_csi.yaml** 文件来定义存储类：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: hostpath-csi
provisioner: kubevirt.io.hostpath-provisioner
reclaimPolicy: Delete (1)
volumeBindingMode: WaitForFirstConsumer (2)
parameters:
  storagePool: my-storage-pool (3)
```

(1) 两个可能的 **reclaimPolicy** 值为 **Delete** 和 **Retain**。如果没有指定值，则默认值为 **Delete**。

(2) **volumeBindingMode** 参数决定何时发生动态置备和卷绑定。指定 **WaitForFirstConsumer**，将持久性卷(PV)的绑定和置备延迟到创建使用持久性卷声明(PVC)的 pod 后。这样可确保 PV 满足 pod 的调度要求。

3 指定 HPP CR 中定义的存储池名称。

2. 保存文件并退出。
3. 运行以下命令来创建 **StorageClass** 对象：

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

5.5. 配置更高的虚拟机工作负载密度

要增加虚拟机 (VM) 的数量，您可以通过过量使用内存 (RAM) 来在集群中配置更高的虚拟机工作负载密度。

重要

配置更高的工作负载密度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。

以下工作负载特别适用于更高的工作负载密度：

- 许多类似的工作负载
- 低使用的工作负载

注意

虽然过度使用的内存可能会导致工作负载密度更高，但也可能会降低高使用系统的工作负载性能。

5.5.1. 使用 wasp-agent 来配置更高的虚拟机工作负载密度

wasp-agent 组件允许 OpenShift Container Platform 集群为虚拟机(VM) 工作负载分配交换资源。只有在 worker 节点上才支持交换使用量。

重要

交换资源只能分配给 **Burstable** 服务质量 (QoS) 类的虚拟机工作负载 (VM pod)。属于虚拟机的 **Guaranteed** QoS 类和任何 QoS 类的 pod 不能交换资源。

有关 QoS 类的描述，请参阅为 Pod 配置服务质量 (Kubernetes 文档)。

先决条件

- **oc** 工具可用。
- 使用 cluster-admin 角色登录到集群。
- 定义内存 over-commit 比率。
- 节点属于 worker 池。

流程

1. 输入以下命令来创建特权服务帐户：

```
$ oc adm new-project wasp
```

```
$ oc create sa -n wasp wasp
```

```
$ oc create clusterrolebinding wasp --clusterrole=cluster-admin --serviceaccount=wasp:wasp
```

```
$ oc adm policy add-scc-to-user -n wasp privileged -z wasp
```

注意

wasp-agent 组件部署 OCI hook，以便在节点级别上为容器启用交换使用情况。低级性质要求 **DaemonSet** 对象具有特权。

2. 通过创建 **DaemonSet** 对象来部署 **wasp-agent**，如下所示：

```
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: wasp-agent
  namespace: wasp
  labels:
    app: wasp
    tier: node
spec:
  selector:
    matchLabels:
      name: wasp
  template:
    metadata:
      annotations:
        description: >-
          Configures swap for workloads
      labels:
        name: wasp
    spec:
      serviceAccountName: wasp
      hostPID: true
      hostUsers: true
      terminationGracePeriodSeconds: 5
      containers:
        - name: wasp-agent
          image: >-
            registry.redhat.io/container-native-virtualization/wasp-agent-rhel9:v4.16
          imagePullPolicy: Always
          env:
          - name: "FSROOT"
            value: "/host"
          resources:
            requests:
```

```
            cpu: 100m
            memory: 50M
          securityContext:
            privileged: true
          volumeMounts:
          - name: host
            mountPath: "/host"
      volumes:
      - name: host
        hostPath:
          path: "/"
      priorityClassName: system-node-critical
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 10%
      maxSurge: 0
status: {}
```

3. 将 **kubelet** 服务配置为允许交换：

   a. 如示例所示，创建一个 **KubeletConfiguration** 文件：

   KubeletConfiguration 文件示例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-config
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "  # MCP
      #machine.openshift.io/cluster-api-machine-role: worker # machine
      #node-role.kubernetes.io/worker: " # node
  kubeletConfig:
    failSwapOn: false
    evictionSoft:
      memory.available: "1Gi"
    evictionSoftGracePeriod:
      memory.available: "10s"
```

   如果集群已使用现有的 **KubeletConfiguration** 文件，请将以下内容添加到 **spec** 部分：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-config
# ...
spec
# ...
    kubeletConfig:
      evictionSoft:
        memory.available: 1Gi
```

```
    evictionSoftGracePeriod:
      memory.available: 1m30s
    failSwapOn: false
```

b. 运行以下命令：

```
$ oc wait mcp worker --for condition=Updated=True
```

4. 创建 **MachineConfig** 对象以置备交换，如下所示：

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 90-worker-swap
spec:
  config:
    ignition:
      version: 3.4.0
    systemd:
      units:
      - contents: |
          [Unit]
          Description=Provision and enable swap
          ConditionFirstBoot=no

          [Service]
          Type=oneshot
          Environment=SWAP_SIZE_MB=5000
          ExecStart=/bin/sh -c "sudo dd if=/dev/zero of=/var/tmp/swapfile
count=${SWAP_SIZE_MB} bs=1M && \
          sudo chmod 600 /var/tmp/swapfile && \
          sudo mkswap /var/tmp/swapfile && \
          sudo swapon /var/tmp/swapfile && \
          free -h && \
          sudo systemctl set-property --runtime system.slice MemorySwapMax=0
IODeviceLatencyTargetSec=\"/ 50ms\""

          [Install]
          RequiredBy=kubelet-dependencies.target
        enabled: true
        name: swap-provision.service
```

为了在出现最糟糕的情况时有足够的交换空间，请确保置备的交换空间的数量最少与过量使用 RAM 相同。使用以下公式计算节点上置备的交换空间量：

```
NODE_SWAP_SPACE = NODE_RAM * (MEMORY_OVER_COMMIT_PERCENT / 100% -
1)
```

Example:

```
NODE_SWAP_SPACE = 16 GB * (150% / 100% - 1)
                = 16 GB * (1.5 - 1)
```

```
= 16 GB * (0.5)
= 8 GB
```

5. 按如下方式部署警报规则：

```
apiVersion: monitoring.openshift.io/v1
kind: AlertingRule
metadata:
  name: wasp-alerts
  namespace: openshift-monitoring
spec:
  groups:
  - name: wasp.rules
    rules:
    - alert: NodeSwapping
      annotations:
        description: Node {{ $labels.instance }} is swapping at a rate of {{ printf "%.2f" $value }}
MB/s
        runbook_url: https://github.com/openshift-virtualization/wasp-
agent/tree/main/runbooks/alerts/NodeSwapping.md
        summary: A node is swapping memory pages
      expr: |
        # In MB/s
        irate(node_memory_SwapFree_bytes{job="node-exporter"}[5m]) / 1024^2 > 0
      for: 1m
      labels:
        severity: critical
```

6. 使用 OpenShift Container Platform Web 控制台或编辑 HyperConverged 自定义资源 (CR) 文件，将 OpenShift Virtualization 配置为使用内存过量使用，如下例所示。
Example:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  higherWorkloadDensity:
    memoryOvercommitPercentage: 150
```

7. 输入以下命令将所有配置应用到集群中的计算节点：

```
$ oc patch --type=merge \
  -f <../manifests/hco-set-memory-overcommit.yaml> \
  --patch-file <../manifests/hco-set-memory-overcommit.yaml>
```

注意

应用所有配置后，交换功能仅在所有 **MachineConfigPool** rollouts 完成后完全可用。

验证

1. 要验证 **wasp-agent** 的部署，请运行以下命令：

```
$  oc rollout status ds wasp-agent -n wasp
```

如果部署成功，则会显示以下信息：

```
daemon set "wasp-agent" successfully rolled out
```

2. 要验证是否正确置备了 swap，请执行以下操作：

   a. 运行以下命令：

```
$ oc get nodes -l node-role.kubernetes.io/worker
```

   b. 从提供的列表中选择节点并运行以下命令：

```
$ oc debug node/<selected-node> -- free -m
```

如果正确置备了 swap，则会显示大于零的数量，如下所示：

	total	使用的	free	shared	buff/cache	可用
Mem:	31846	23155	1044	6014	14483	8690
Swap:	8191	2337	5854			

3. 运行以下命令验证 OpenShift Virtualization 内存过量使用配置：

```
$ oc get -n openshift-cnv HyperConverged kubevirt-hyperconverged -o jsonpath="
{.spec.higherWorkloadDensity.memoryOvercommitPercentage}"
150
```

返回的值（如 **150**）必须与之前配置的值匹配。

第 6 章 更新

6.1. 更新 OPENSHIFT VIRTUALIZATION

了解 Operator Lifecycle Manager (OLM) 如何为 OpenShift Virtualization 提供 z-stream 和次要版本更新。

6.1.1. RHEL 9 上的 OpenShift Virtualization

OpenShift Virtualization 4.16 基于 Red Hat Enterprise Linux (RHEL) 9。您可以按照标准 OpenShift Virtualization 更新过程，从一个基于 RHEL 8 的版本升级到 OpenShift Virtualization 4.16。不需要额外的步骤。

与之前的版本一样，您可以在不中断运行工作负载的情况下执行更新。OpenShift Virtualization 4.16 支持从 RHEL 8 节点实时迁移到 RHEL 9 节点。

6.1.1.1. RHEL 9 机器类型

OpenShift Virtualization 中包含的所有虚拟机模板现在默认使用 RHEL 9 机器类型： **machineType: pc-q35-rhel9.<y>.0**，其中 **<y>** 是与最新 RHEL 9 次版本对应的数字。例如，值 **pc-q35-rhel9.2.0** 用于 RHEL 9.2。

更新 OpenShift Virtualization 不会更改任何现有虚拟机的 **machineType** 值。这些虚拟机在更新前继续正常工作。您可以选择更改虚拟机的机器类型，使其可从 RHEL 9 改进中受益。

重要

在更改虚拟机的 **machineType** 值前，您必须关闭虚拟机。

6.1.2. 关于更新 OpenShift Virtualization

- Operator Lifecycle Manager (OLM) 管理 OpenShift Virtualization Operator 的生命周期。Marketplace Operator 在 OpenShift Container Platform 安装过程中部署，使外部 Operator 可供集群使用。
- OLM 为 OpenShift Virtualization 提供 z-stream 和次要版本更新。在将 OpenShift Container Platform 更新至下一个次版本时，次版本更新将变为可用。在不先更新 OpenShift Container Platform 的情况下，您无法将 OpenShift Virtualization 更新至下一个次版本。
- OpenShift Virtualization 订阅使用一个名为 **stable** 的单一更新频道。**stable** 频道确保 OpenShift Virtualization 和 OpenShift Container Platform 版本兼容。
- 如果您的订阅的批准策略被设置为 **Automatic**，则当 **stable** 频道中提供新版本的 Operator 时，更新过程就会马上启动。强烈建议您使用 **Automatic（自动）** 批准策略来维护可支持的环境。只有在运行对应的 OpenShift Container Platform 版本时，才会支持 OpenShift Virtualization 的每个次要版本。例如，您必须在 OpenShift Container Platform 4.16 上运行 OpenShift Virtualization 4.16。
    - 虽然可以选择 **Manual（手工）** 批准策略，但并不建议这样做，因为它存在集群的支持性和功能风险。使用 **Manual** 批准策略时，您必须手动批准每个待处理的更新。如果 OpenShift Container Platform 和 OpenShift Virtualization 更新不同步，您的集群将无法被支持。
- 更新完成所需时间取决于您的网络连接情况。大部分自动更新可在十五分钟内完成。

- 更新 OpenShift Virtualization 不会中断网络连接。
- 数据卷及其关联的持久性卷声明会在更新过程中保留。

重要

如果您的虚拟机正在运行，使用 hostpath 置备程序存储，则无法实时迁移，并可能会阻止 OpenShift Container Platform 集群更新。

作为临时解决方案，您可以重新配置虚拟机以便在集群更新过程中自动关闭它们。将 **evictionStrategy** 字段设置为 **None**，将 **runStrategy** 字段设置为 **Always**。

6.1.2.1. 关于工作负载更新

更新 OpenShift Virtualization 时，虚拟机工作负载（包括 **libvirt**、**virt-launcher** ）和 **qemu** （如果支持实时迁移）会自动更新。

注意

每个虚拟机均有一个 **virt-launcher** pod，用于运行虚拟机实例(VMI)。**virt-launcher** pod 运行一个 **libvirt** 实例，用于管理虚拟机(VM)进程。

您可以通过编辑 **HyperConverged** 自定义资源 (CR) 的 **spec.workloadUpdateStrategy** 小节来配置工作负载的更新方式。可用的工作负载更新方法有两种： **LiveMigrate** 和 **Evict**。

因为 **Evict** 方法关闭 VMI pod，所以只启用 **LiveMigrate** 更新策略。

当 **LiveMigrate** 是唯一启用的更新策略时：

- 支持实时迁移的 VMI 会在更新过程中进行迁移。VM 客户机会进入启用了更新组件的新 pod。
- 不支持实时迁移的 VMI 不会中断或更新。
    - 如果 VMI 有 **LiveMigrate** 驱除策略，但没有支持实时迁移。

如果您同时启用 **LiveMigrate** 和 **Evict** ：

- 支持实时迁移的 VMI 使用 **LiveMigrate** 更新策略。
- 不支持实时迁移的 VMI 使用 **Evict** 更新策略。如果 VMI 由带有 **runStrategy: Always** 设置的 **VirtualMachine** 对象控制，则会在带有更新组件的新 pod 中创建一个新的 VMI。

迁移尝试和超时
更新工作负载时，如果 pod 在以下时间段内处于 **Pending** 状态，实时迁移会失败：

5 分钟

如果 pod 因为是 **Unschedulable** 而处于 pending 状态。

15 分钟

如果 pod 因任何原因处于 pending 状态。

当 VMI 无法迁移时， **virt-controller** 会尝试再次迁移它。它会重复这个过程，直到所有可可迁移的 VMI 在新的 **virt-launcher** Pod 上运行。如果 VMI 没有被正确配置，这些尝试可能会无限期重复。

注意

每次尝试都会对应于一个迁移对象。只有最近五个尝试才在缓冲区中。这可防止迁移对象在系统上进行积累，同时保留用于调试的信息。

6.1.2.2. 关于 Control Plane Only 更新

每个 OpenShift Container Platform 的次版本号为偶数（包括 4.10 和 4.12）都是延长更新支持(EUS)版本。但是，由于 Kubernetes 设计了串行次版本更新，所以您无法直接从一个 EUS 版本更新到下一个版本。

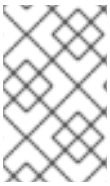
从源 EUS 版本升级到下一个奇数次版本后，您必须按顺序将 OpenShift Virtualization 更新至更新路径中的所有次版本的 z-stream 版本。当您升级到最新的适用 z-stream 版本时，您可以将 OpenShift Container Platform 更新至目标 EUS 次版本。

当 OpenShift Container Platform 更新成功时，OpenShift Virtualization 的对应更新将变为可用。现在，您可以将 OpenShift Virtualization 更新至目标 EUS 版本。

6.1.2.2.1. 准备更新

在开始 Control Plane Only 更新前，您必须：

- 在启动 Control Plane Only 更新前暂停 worker 节点的机器配置池，以便 worker 不会重启两次。
- 在开始更新过程前禁用自动工作负载更新。这是为了防止 OpenShift Virtualization 迁移或驱除虚拟机(VM)，直到您升级到目标 EUS 版本。



注意

默认情况下，当您更新 OpenShift Virtualization Operator 时，OpenShift Virtualization 会自动更新工作负载，如 **virt-launcher** pod。您可以在 **HyperConverged** 自定义资源的 **spec.workloadUpdateStrategy** 小节中配置此行为。

了解有关 执行 Control Plane 仅更新 的更多信息。

6.1.3. 防止在 Control Plane Only 更新过程中进行工作负载更新

当您从一个延长更新支持(EUS)版本升级到下一个版本时，您必须手动禁用自动工作负载更新，以防止 OpenShift Virtualization 在更新过程中迁移或驱除工作负载。

先决条件

- 您正在运行 EUS 版本 OpenShift Container Platform，并希望升级到下一个 EUS 版本。还没有同时更新至奇数版本。
- 您可以阅读"准备执行 Control Plane only update"，并了解与 OpenShift Container Platform 集群相关的注意事项和要求。
- 按照 OpenShift Container Platform 文档的指示暂停 worker 节点的机器配置池。
- 建议您使用默认的 **Automatic** 批准策略。如果使用 **Manual** 批准策略，您必须批准 web 控制台中的所有待处理的更新。如需了解更多详细信息，请参阅"需要批准待处理的 Operator 更新"部分。

流程

1. 运行以下命令并记录 **workloadUpdateMethods** 配置：

```
$ oc get kv kubevirt-kubevirt-hyperconverged \
  -n openshift-cnv -o jsonpath='{.spec.workloadUpdateStrategy.workloadUpdateMethods}'
```

2. 运行以下命令关闭所有工作负载更新方法：

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  --type json -p
'[{"op":"replace","path":"/spec/workloadUpdateStrategy/workloadUpdateMethods", "value":[]}]'
```

输出示例

```
hyperconverged.hco.kubevirt.io/kubevirt-hyperconverged patched
```

3. 在继续操作前，请确保 **HyperConverged** Operator 为 **Upgradeable**。输入以下命令并监控输出：

```
$ oc get hyperconverged kubevirt-hyperconverged -n openshift-cnv -o json | jq
".status.conditions"
```

例 6.1. 输出示例


```
[
  {
    "lastTransitionTime": "2022-12-09T16:29:11Z",
    "message": "Reconcile completed successfully",
    "observedGeneration": 3,
    "reason": "ReconcileCompleted",
    "status": "True",
    "type": "ReconcileComplete"
  },
  {
    "lastTransitionTime": "2022-12-09T20:30:10Z",
    "message": "Reconcile completed successfully",
    "observedGeneration": 3,
    "reason": "ReconcileCompleted",
    "status": "True",
    "type": "Available"
  },
  {
    "lastTransitionTime": "2022-12-09T20:30:10Z",
    "message": "Reconcile completed successfully",
    "observedGeneration": 3,
    "reason": "ReconcileCompleted",
    "status": "False",
    "type": "Progressing"
  },
  {
    "lastTransitionTime": "2022-12-09T16:39:11Z",
    "message": "Reconcile completed successfully",
    "observedGeneration": 3,
```

```
      "reason": "ReconcileCompleted",
      "status": "False",
      "type": "Degraded"
    },
    {
      "lastTransitionTime": "2022-12-09T20:30:10Z",
      "message": "Reconcile completed successfully",
      "observedGeneration": 3,
      "reason": "ReconcileCompleted",
      "status": "True",
      "type": "Upgradeable" 1
    }
  ]
```

1 OpenShift Virtualization Operator 具有 **Upgradeable** 状态。

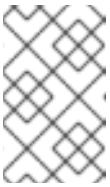
4. 手动将集群从源 EUS 版本升级到下一个 OpenShift Container Platform 次要版本：

```
$ oc adm upgrade
```

验证

- 运行以下命令检查当前版本：

```
$ oc get clusterversion
```



注意

将 OpenShift Container Platform 更新至下一版本是更新 OpenShift Virtualization 的先决条件。如需了解更多详细信息，请参阅 OpenShift Container Platform 文档中的"更新集群"部分。

5. 更新 OpenShift Virtualization。
   - 使用默认的 **Automatic** 批准策略，OpenShift Virtualization 会在更新 OpenShift Container Platform 后自动更新到对应的版本。
   - 如果使用 **Manual** 批准策略，请使用 Web 控制台批准待处理的更新。
6. 运行以下命令监控 OpenShift Virtualization 更新：

```
$ oc get csv -n openshift-cnv
```

7. 将 OpenShift Virtualization 更新至可用于非 EUS 次版本的每个 z-stream 版本，通过运行上一步中显示的命令来监控每个更新。
8. 运行以下命令，确认 OpenShift Virtualization 已成功更新至非 EUS 版本的最新 z-stream 版本：

```
$ oc get hyperconverged kubevirt-hyperconverged -n openshift-cnv -o json | jq
".status.versions"
```

输出示例

```
[
  {
    "name": "operator",
    "version": "4.16.3"
  }
]
```

9. 等待 **HyperConverged** Operator 在执行下一次更新前具有 **Upgradeable** 状态。输入以下命令并监控输出：

```
$ oc get hyperconverged kubevirt-hyperconverged -n openshift-cnv -o json | jq
".status.conditions"
```

10. 将 OpenShift Container Platform 更新至目标 EUS 版本。

11. 通过检查集群版本确认更新是否成功：

```
$ oc get clusterversion
```

12. 将 OpenShift Virtualization 更新至目标 EUS 版本。

    - 使用默认的 **Automatic** 批准策略，OpenShift Virtualization 会在更新 OpenShift Container Platform 后自动更新到对应的版本。
    - 如果使用 **Manual** 批准策略，请使用 Web 控制台批准待处理的更新。

13. 运行以下命令监控 OpenShift Virtualization 更新：

```
$ oc get csv -n openshift-cnv
```

当 **VERSION** 字段与目标 EUS 版本匹配并且 **PHASE** 字段显示为 **Succeeded** 时，更新已完成。

14. 使用以下命令恢复您从第 1 步中记录的 **workloadUpdateMethods** 配置：

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv --type json -p \
  "[{\"op\":\"add\",\"path\":\"/spec/workloadUpdateStrategy/workloadUpdateMethods\",
\"value\":{WorkloadUpdateMethodConfig}}]"
```

输出示例

```
hyperconverged.hco.kubevirt.io/kubevirt-hyperconverged patched
```

验证

- 运行以下命令检查虚拟机迁移的状态：

```
$ oc get vmim -A
```

后续步骤

- 现在，您可以取消暂停 worker 节点的机器配置池。

6.1.4. 配置工作负载更新方法

您可以通过编辑 **HyperConverged** 自定义资源(CR)来配置工作负载更新方法。

先决条件

- 要使用实时迁移作为更新方法，您必须首先在集群中启用实时迁移。

注意

如果 **VirtualMachineInstance** CR 包含 **evictionStrategy: LiveMigrate**，且虚拟机实例(VMI)不支持实时迁移，则 VMI 将不会更新。

流程

1. 要在默认编辑器中打开 **HyperConverged** CR，请运行以下命令：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 编辑 **HyperConverged** CR 的 **workloadUpdateStrategy** 小节。例如：

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  workloadUpdateStrategy:
    workloadUpdateMethods: 1
    - LiveMigrate 2
    - Evict 3
    batchEvictionSize: 10 4
    batchEvictionInterval: "1m0s" 5
# ...
```

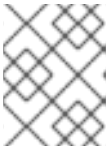
1. 可用于执行自动化工作负载更新的方法。可用值为 **LiveMigrate** 和 **Evict**。如果您如本例所示启用这两个选项，则更新会为不支持实时迁移的 VMI 使用 **LiveMigrate**，对于不支持实时迁移的 VMI 使用 **Evict**。要禁用自动工作负载更新，您可以删除 **workloadUpdateStrategy** 小节，或设置 **workloadUpdateMethods: []** 将数组留空。

2. 具有最低破坏性的更新方法。支持实时迁移的 VMI 通过将虚拟机 (VM) 客户机迁移到启用了更新组件的新 pod 中来更新。如果 **LiveMigrate** 是唯一列出的工作负载更新方法，不支持实时迁移的 VMI 不会中断或更新。

3. 在升级过程中关闭 VMI pod 是一个有破坏性的方法。如果在集群中没有启用实时迁移，**Evict** 是唯一可用的更新方法。如果 VMI 由带有 **runStrategy: Always** 配置的 **VirtualMachine** 对象控制，则会在带有更新组件的新 pod 中创建一个新的 VMI。

4. 使用 **Evict** 方法每次可以强制更新的 VMI 数量。这不适用于 **LiveMigrate** 方法。

5. 驱除下一批工作负载前等待的时间间隔。这不适用于 **LiveMigrate** 方法。



注意

您可以通过编辑 **HyperConverged** CR 的 **spec.liveMigrationConfig** 小节来配置实时迁移限制和超时。

3. 若要应用您的更改，请保存并退出编辑器。

6.1.5. 批准待处理的 Operator 更新

6.1.5.1. 手动批准待处理的 Operator 更新

如果已安装的 Operator 的订阅被设置为 **Manual**，则当其当前更新频道中发布新更新时，在开始安装前必须手动批准更新。

先决条件

- 之前使用 Operator Lifecycle Manager（OLM）安装的 Operator。

流程

1. 在 OpenShift Container Platform Web 控制台的 **Administrator** 视角中，进入 **Operators → Installed Operators**。
2. 处于待定更新的 Operator 会显示 **Upgrade available** 状态。点您要更新的 Operator 的名称。
3. 点 **Subscription** 标签页。任何需要批准的更新都会在 **Upgrade status** 旁边显示。例如：它可能会显示 **1 requires approval**。
4. 点 **1 requires approval**，然后点 **Preview Install Plan**。
5. 检查列出可用于更新的资源。在满意后，点 **Approve**。
6. 返回到 **Operators → Installed Operators** 页面，以监控更新的进度。完成后，状态会变为 **Succeeded** 和 **Up to date**。

6.1.6. 监控更新状态

6.1.6.1. 监控 OpenShift Virtualization 升级状态

要监控 OpenShift Virtualization Operator 升级的状态，请观察集群服务版本 (CSV) **PHASE**。此外您还可在 web 控制台中，或运行此处提供的命令来监控 CSV 状况。

注意

PHASE 和状况值均是基于可用信息的近似值。

先决条件

- 以具有 **cluster-admin** 角色的用户身份登录集群。
- 安装 OpenShift CLI（**oc**）。

流程

1. 运行以下命令：

```
$ oc get csv -n openshift-cnv
```

2. 查看输出，检查 **PHASE** 字段。例如：

输出示例

```
VERSION  REPLACES                                   PHASE
4.9.0    kubevirt-hyperconverged-operator.v4.8.2    Installing
4.9.0    kubevirt-hyperconverged-operator.v4.9.0    Replacing
```

3. 可选：运行以下命令来监控所有 OpenShift Virtualization 组件状况的聚合状态：

```
$ oc get hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  -o=jsonpath='{range .status.conditions[*]}{.type}{"\t"}{.status}{"\t"}{.message}{"\n"}{end}'
```

成功升级后会输出以下内容：

输出示例

```
ReconcileComplete  True  Reconcile completed successfully
Available          True  Reconcile completed successfully
Progressing        False Reconcile completed successfully
Degraded           False Reconcile completed successfully
Upgradeable        True  Reconcile completed successfully
```

6.1.6.2. 查看过时的 OpenShift Virtualization 工作负载

您可以使用 CLI 查看过时的工作负载列表。

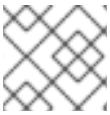
注意

如果集群中存在过时的虚拟化 pod，**OutdatedVirtualMachineInstanceWorkloads** 警报会触发。

流程

- 要查看过时的虚拟机实例 (VMI) 列表，请运行以下命令：

```
$ oc get vmi -l kubevirt.io/outdatedLauncherImage --all-namespaces
```



注意

配置工作负载更新以确保 VMI 自动更新。

6.1.7. 其他资源

- 仅执行 Control Plane 更新
- 什么是 Operator?

- Operator Lifecycle Manager 概念和资源
- 集群服务版本（CSV）
- 关于实时迁移
- 配置驱除策略
- 配置实时迁移限制和超时

第 7 章 虚拟机

7.1. 从红帽镜像创建虚拟机

7.1.1. 从红帽镜像创建虚拟机概述

红帽镜像是金级镜像。它们作为容器磁盘在安全 registry 中发布。Containerized Data Importer (CDI) 轮询容器磁盘并将其导入到集群中，并将其存储在 **openshift-virtualization-os-images** 项目中作为快照或持久性卷声明(PVC)。

红帽镜像会自动更新。您可以为这些镜像禁用和重新启用自动更新。请参阅管理红帽引导源更新。

集群管理员现在可以在 OpenShift Virtualization web 控制台中为 Red Hat Enterprise Linux (RHEL)虚拟机启用自动订阅。

您可以使用以下方法之一从红帽提供的操作系统镜像创建虚拟机(VM)：

- 使用 Web 控制台从模板创建虚拟机
- 使用 Web 控制台从实例类型创建虚拟机
- 使用命令行从 **VirtualMachine** 清单创建虚拟机

重要

不要在默认的 **openshift** Galaxy 命名空间中创建虚拟机。相反，创建一个新命名空间或使用没有 **openshift** 前缀的现有命名空间。

7.1.1.1. 关于金级镜像

金级镜像是虚拟机(VM)的预配置快照，您可以用作部署新虚拟机的资源。例如，您可以使用金级镜像来更加一致地置备相同的系统环境，并更快高效地部署系统。

7.1.1.1.1. 金级镜像如何工作？

通过在参考机器或虚拟机上安装和配置操作系统和软件应用程序来创建金级镜像。这包括设置系统、安装所需的驱动程序、应用补丁和更新以及配置特定选项和首选项。

创建金级镜像后，它会保存为模板或镜像文件，可在多个集群中复制和部署。金级镜像可以通过维护人员定期更新，以纳入必要的软件更新和补丁，确保镜像保持最新且安全，并且新创建的虚拟机基于这个更新的镜像。

7.1.1.1.2. 红帽对金级镜像的实施

对于 Red Hat Enterprise Linux (RHEL) 版本，红帽发布金级镜像作为 registry 中的容器磁盘。容器磁盘是虚拟机镜像，它作为容器镜像 registry 存储在容器镜像 registry 中。安装 OpenShift Virtualization 后，任何发布的镜像将自动在连接的集群中提供。镜像在集群中可用后，可以使用它们创建虚拟机。

7.1.1.2. 关于虚拟机引导源

虚拟机 (VM) 由虚拟机定义以及由数据卷支持的一个或多个磁盘组成。VM 模板允许您使用预定义的规格创建虚拟机。

每个模板都需要一个引导源，它是一个完全配置的磁盘镜像，包括配置的驱动程序。每个模板都包含一个虚拟机定义，其中包含指向引导源的指针。每个引导源都有一个预定义的名称和命名空间。对于某些操作系统，会自动提供一个引导源。如果没有提供，管理员必须准备自定义引导源。

提供的引导源会自动更新至操作系统的最新版本。对于自动更新的引导源，持久性卷声明 (PVC) 和卷快照会使用集群的默认存储类创建。如果在配置后选择了不同的默认存储类，您必须删除使用之前默认存储类配置的集群命名空间中的现有引导源。

7.1.2. 从实例类型创建虚拟机

您可以使用实例类型（无论 OpenShift Container Platform web 控制台或 CLI 创建虚拟机）来简化虚拟机 (VM) 创建。

7.1.2.1. 关于实例类型

实例类型是一种可重复使用的对象，您可以定义应用到新虚拟机的资源和特征。您可以定义自定义实例类型，或使用安装 OpenShift Virtualization 时包括的各种类型。

要创建新实例类型，您必须首先手动创建清单，也可以使用 **virtctl** CLI 工具创建清单。然后，您可以通过将清单应用到集群来创建实例类型对象。

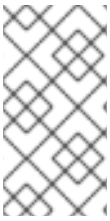
OpenShift Virtualization 为配置实例类型提供两个 CRD：

- 命名空间范围对象：**VirtualMachineInstancetype**
- 集群范围的对象： **VirtualMachineClusterInstancetype**

这些对象使用相同的 **VirtualMachineInstancetypeSpec**。

7.1.2.1.1. 所需属性

配置实例类型时，您必须定义 **cpu** 和 **memory** 属性。其他属性是可选的。



注意

从实例类型创建虚拟机时，您无法覆盖实例类型中定义的任何参数。

因为实例类型需要定义的 CPU 和内存属性，所以 OpenShift Virtualization 始终会在从实例类型创建虚拟机时拒绝这些资源的额外请求。

您可以手动创建实例类型清单。例如：

带有必填字段的 YAML 文件示例

```
apiVersion: instancetype.kubevirt.io/v1beta1
kind: VirtualMachineInstancetype
metadata:
  name: example-instancetype
spec:
  cpu:
    guest: 1 (1)
  memory:
    guest: 128Mi (2)
```

1 必需。指定要分配给客户机的 vCPU 数量。

2 必需。指定要分配给客户机的内存量。

您可以使用 **virtctl** CLI 实用程序创建实例类型清单。例如：

带有必填字段的 virtctl 命令示例

```
$ virtctl create instancetype --cpu 2 --memory 256Mi
```

其中：

--cpu <value>
指定要分配给客户机的 vCPU 数量。必需。

--memory <value>
指定要分配给客户机的内存量。必需。

提示

您可以运行以下命令来立即从新清单中创建对象：

```
$ virtctl create instancetype --cpu 2 --memory 256Mi | oc apply -f -
```

7.1.2.1.2. 可选属性

除了所需的 **cpu** 和 **memory** 属性外，您还可以在 **VirtualMachineInstancetypeSpec** 中包含以下可选属性：

annotations
列出应用到虚拟机的注解。

gpus
列出用于 passthrough 的 vGPU。

hostDevices
列出用于透传的主机设备。

ioThreadsPolicy
定义用于管理专用磁盘访问的 IO 线程策略。

launchSecurity
配置安全加密虚拟化 (SEV)。

nodeSelector
指定节点选择器来控制调度此虚拟机的节点。

schedulerName
定义用于此虚拟机的自定义调度程序，而不是默认的调度程序。

7.1.2.2. 预定义的实例类型

OpenShift Virtualization 包括一组预定义的实例类型，称为 **common-instancetypes**。一些会针对特定工作负载进行定制，另一些则与工作负载无关。

这些实例类型资源根据其系列、版本和大小命名。大小值使用 **.** 分隔符，范围从 **nano** 到 **8xlarge**。

表 7.1. common-instancetypes 系列比较

使用案例	系列	特性	vCPU 与内存的比率	资源示例
Universal	U	• Burstable CPU 性能	1:4	**u1.medium** • 1 个 vCPU • 4 Gi 内存
过量使用	O	• 过量使用的内存 • Burstable CPU 性能	1:4	**o1.small** • 1 vCPU • 2Gi 内存
compute-exclusive	CX	• Hugepages • 专用 CPU • 隔离的仿真程序线程 • vNUMA	1:2	**cx1.2xlarge** • 8 个 vCPU • 16Gi 内存
NVIDIA GPU	GN	• 对于使用 NVIDIA GPU Operator 提供的 GPU 的虚拟机 • 具有预定义的 GPU • Burstable CPU 性能	1:4	**gn1.8xlarge** • 32 个 vCPU • 128Gi 内存
内存密集型	M	• Hugepages • Burstable CPU 性能	1:8	**m1.large** • 2 个 vCPU • 16Gi 内存

使用案例	系列	特性	vCPU 与内存的比率	资源示例
Network-intensive	N	• Hugepages • 专用 CPU • 隔离的仿真程序线程 • 需要能够运行 DPDK 工作负载的节点	1:2	**n1.medium** • 4 个 vCPU • 4Gi 内存

7.1.2.3. 使用 virtctl 工具创建清单

您可使用 **virtctl** CLI 实用程序简化为虚拟机、虚拟机实例类型和虚拟机首选项创建清单。如需更多信息，请参阅虚拟机清单创建命令。

如果您有 **VirtualMachine** 清单，可以从命令行创建虚拟机。

7.1.2.4. 使用 Web 控制台从实例类型创建虚拟机

您可以使用 OpenShift Container Platform web 控制台从实例类型创建虚拟机 (VM)。您还可以通过复制现有快照或克隆虚拟机，来使用 Web 控制台创建虚拟机。

您可以从可用可引导卷列表创建虚拟机。您可以在列表中添加基于 Linux 或 Windows 的卷。

流程

1. 在 Web 控制台中，进入到 **Virtualization → Catalog**。
   InstanceTypes 选项卡默认为打开。
2. 选择以下选项之一：
   - 从列表中选择合适的可引导卷。如果列表已被截断，请点 **Show all** 按钮来显示整个列表。

     注意

     可引导的卷表仅列出 **openshift-virtualization-os-images** 命名空间中具有 **instancetype.kubevirt.io/default-preference** 标签的卷。

     - 可选：点星号图标将可引导卷指定为热门卷。不足的可引导卷首先出现在卷列表中。
   - 点 **Add volume** 上传新卷，或使用现有的持久性卷声明(PVC)、卷快照或 **containerDisk** 卷。点击 **Save**。
     集群中不可用的操作系统的徽标显示在列表的底部。您可以点 **Add volume** 链接为所需的操作系统添加卷。

     另外，还有 **创建 Windows 引导源**快速启动的链接。如果您将鼠标悬停在 *Select volume to boot from* 行旁边的问号图标上，则同一链接会出现在弹出窗口中。

安装环境或环境断开连接后，从中引导的卷列表为空。在这种情况下，会显示三个操作系统徽标：Windows、RHEL 和 Linux。您可以点 **Add volume** 按钮添加新卷来满足您的要求。

3. 点实例类型标题，然后选择适合您的工作负载的资源大小。
4. 可选：选择虚拟机详情，包括虚拟机的名称，适用于您要从其引导的卷：
   - 对于基于 Linux 的卷，请按照以下步骤配置 SSH：
     a. 如果您还没有在项目中添加公共 SSH 密钥，点 **VirtualMachine details** 部分中的 **Authorized SSH key** 旁边的编辑图标。
     b. 选择以下选项之一：
        - **使用现有** ：从 secrets 列表中选择一个 secret。
        - **Add new**: 遵循以下步骤：
          i. 浏览到公共 SSH 密钥文件，或在 key 字段中粘贴文件。
          ii. 输入 secret 名称。
          iii. 可选： 选择 **Automatically apply this key to any new VirtualMachine you create in this project**。
     c. 点击 **Save**。
   - 对于 Windows 卷，请按照以下步骤配置 sysprep 选项：
     - 如果您还没有为 Windows 卷添加 sysprep 选项，请按照以下步骤执行：
       i. 点 **VirtualMachine 详情** 部分中的 **Sysprep** 的编辑图标。
       ii. 添加 **Autoattend.xml** 回答文件。
       iii. 添加 **Unattend.xml** 回答文件。
       iv. 点击 **Save**。
     - 如果要将现有的 sysprep 选项用于 Windows 卷，请按照以下步骤执行：
       i. 点 **Attach existing sysprep**。
       ii. 输入现有 sysprep **Unattend.xml** 回答文件的名称。
       iii. 点击 **Save**。
5. 可选： 如果要创建 Windows 虚拟机，您可以挂载 Windows 驱动程序磁盘：
   a. 点 **Customize VirtualMachine** 按钮。
   b. 在 **VirtualMachine 详情**页中，点 **Storage**。
   c. 选择 **Mount Windows 驱动程序磁盘**复选框。
6. 可选：点 **View YAML & CLI** 查看 YAML 文件。点 **CLI** 查看 CLI 命令。您还可以下载或复制 YAML 文件内容或 CLI 命令。
7. 点 **Create VirtualMachine**。

创建虚拟机后，您可以在 **VirtualMachine 详情**页中监控状态。

7.1.3. 从模板创建虚拟机

您可以使用 OpenShift Container Platform web 控制台从红帽模板创建虚拟机 (VM)。

7.1.3.1. 关于虚拟机模板

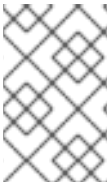
引导源

您可以使用有可用引导源的模板加快虚拟机创建。如果带有引导源的模板没有自定义标签，则会被标记为 **Available boot source**。
没有引导源的模板被标记为 **Boot source required**。请参阅从自定义镜像创建虚拟机。

自定义

在启动虚拟机前，您可以自定义磁盘源和虚拟机参数。

有关磁盘源设置的详情，请参阅存储卷类型和存储字段。



注意

如果您使用所有标签和注解复制虚拟机模板，则当部署新版本的 Scheduling、Scale 和 Performance (SSP) Operator 时，您的模板版本将被标记为已弃用。您可以删除此设计。请参阅使用 Web 控制台自定义虚拟机模板。

单节点 OpenShift

由于存储行为的区别，一些模板与单节点 OpenShift 不兼容。为确保兼容性，请不要为使用数据卷或存储配置集的模板或虚拟机设置 **evictionStrategy** 字段。

7.1.3.2. 从模板创建虚拟机

您可以使用 OpenShift Container Platform web 控制台从带有可用引导源的模板创建虚拟机 (VM)。

可选：在启动虚拟机前，您可以自定义模板或虚拟机参数，如数据源、cloud-init 或 SSH 密钥。

流程

1. 在 web 控制台中进入到 **Virtualization → Catalog**。
2. 点 **Boot source available** 来使用引导源过滤模板。
   目录显示默认模板。点 **All Items** 查看您的过滤器的所有可用模板。
3. 点模板标题查看其详情。
4. 可选：如果您使用 Windows 模板，可以通过选择 **Mount Windows 驱动程序磁盘**复选框来挂载 Windows 驱动程序磁盘。
5. 如果您不需要自定义模板或虚拟机参数，点 **Quick create VirtualMachine** 从模板创建虚拟机。
   如果您需要自定义模板或虚拟机参数，请执行以下操作：
   a. 点 **Customize VirtualMachine**。
   b. 展开 **Storage** 或 **Optional 参数**，以编辑数据源设置。

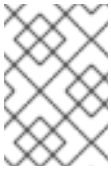
c. 点 **Customize VirtualMachine 参数**。
Customize and create VirtualMachine 窗格显示 **Overview**, **YAML**, **Scheduling**, **Environment**, **Network interfaces**, **Disks**, **Scripts**, 和 **Metadata** 标签页。

d. 编辑在虚拟机引导前必须设置的参数，如 cloud-init 或静态 SSH 密钥。

e. 点 **Create VirtualMachine**。
VirtualMachine 详情页面会显示 provisioning 状态。

7.1.3.2.1. 存储卷类型

表 7.2. 存储卷类型

Type	描述
ephemeral	将网络卷用作只读后备存储的本地写时复制 (COW) 镜像。后备卷必须为 **PersistentVolumeClaim**。当虚拟机启动并在本地存储所有写入数据时，便会创建临时镜像。当虚拟机停止、重启或删除时，便会丢弃临时镜像。其底层的卷 (PVC) 不会以任何方式发生变化。
persistentVolumeClaim	将可用 PV 附加到虚拟机。附加 PV 可确保虚拟机数据在会话之间保持。 将现有虚拟机导入到 OpenShift Container Platform 中的建议方法是，使用 CDI 将现有虚拟机磁盘导入到 PVC 中，然后将 PVC 附加到虚拟机实例。在 PVC 中使用磁盘需要满足一些要求。
dataVolume	通过导入、克隆或上传操作来管理虚拟机磁盘的准备过程，以此在 **persistentVolumeClaim** 磁盘类型基础上构建数据卷。使用此卷类型的虚拟机可保证在卷就绪前不会启动。 指定 **type: dataVolume** 或 **type: ""**。如果您为 **type** 指定任何其他值，如 **persistentVolumeClaim**，则会显示警告信息，虚拟机也不会启动。
cloudInitNoCloud	附加包含所引用的 cloud-init NoCloud 数据源的磁盘，从而向虚拟机提供用户数据和元数据。虚拟机磁盘内部需要安装 cloud-init。
containerDisk	引用容器镜像 registry 中存储的镜像，如虚拟机磁盘。镜像从 registry 中拉取，并在虚拟机启动时作为磁盘附加到虚拟机。 **containerDisk** 卷不仅限于一个虚拟机，对于要创建大量无需持久性存储的虚拟机克隆来说也非常有用。 容器镜像 registry 仅支持 RAW 和 QCOW2 格式的磁盘类型。建议使用 QCOW2 格式以减小镜像的大小。  **注意** **containerDisk** 卷是临时的。将在虚拟机停止、重启或删除时丢弃。**containerDisk** 卷对于只读文件系统（如 CD-ROM）或可处理的虚拟机很有用。

Type	描述
emptyDisk	创建额外的稀疏 QCOW2 磁盘，与虚拟机接口的生命周期相关联。当虚拟机中的客户端初始化重启后，数据保留下来，但当虚拟机停止或从 web 控制台重启时，数据将被丢弃。空磁盘用于存储应用程序依赖项和数据，否则这些依赖项和数据会超出临时磁盘有限的临时文件系统。 此外还必须提供磁盘容量大小。

7.1.3.2.2. 存储字段

字段	描述
空白（创建 PVC）	创建一个空磁盘。
通过 URL 导入（创建 PVC）	通过 URL（HTTP 或 HTTPS 端点）导入内容。
使用现有的 PVC	使用集群中已可用的 PVC。
克隆现有的 PVC（创建 PVC）	选择集群中可用的现有 PVC 并克隆它。
通过 Registry 导入（创建 PVC）	通过容器 registry 导入内容。
容器（临时）	从集群可以访问的 registry 中的容器上传内容。容器磁盘应只用于只读文件系统，如 CD-ROM 或临时虚拟机。
名称	磁盘的名称。名称可包含小写字母 (**a-z**)、数字 (**0-9**)、连字符 (**-**) 和句点 (**.**)，最多 253 个字符。第一个和最后一个字符必须为字母数字。名称不得包含大写字母、空格或特殊字符。
Size	GiB 中磁盘的大小。
类型	磁盘类型。示例：磁盘或光盘
Interface	磁盘设备的类型。支持的接口包括 **virtIO**、**SATA** 和 **SCSI**。
Storage class	用于创建磁盘的存储类。

高级存储设置
以下高级存储设置是可选的，对 **Blank**, **Import via URL**, and **Clone existing PVC** 磁盘可用。

如果没有指定这些参数，系统将使用默认存储配置集值。

参数	选项	参数描述
卷模式	Filesystem	在基于文件系统的卷中保存虚拟磁盘。
	Block	直接将虚拟磁盘存储在块卷中。只有底层存储支持时才使用 **Block**。
访问模式	ReadWriteOnce (RWO)	卷可以被一个节点以读写模式挂载。
	ReadWriteMany (RWX)	卷可以被多个节点以读写模式挂载。  **注意** 实时迁移需要此模式。

7.1.3.2.3. 使用 Web 控制台自定义虚拟机模板

在启动虚拟机前，您可以通过修改 VM 或模板参数（如数据源、cloud-init 或 SSH 密钥）来自定义现有虚拟机(VM)模板。如果您通过复制模板并包含其所有标签和注解，则部署新版本的 Scheduling、Scale 和 Performance (SSP) Operator 时，自定义模板将标记为已弃用。

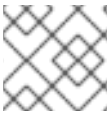
您可以从自定义模板中删除已弃用的设计。

流程

1. 在 web 控制台中进入到 **Virtualization → Templates**。
2. 从虚拟机模板列表中，点标记为已弃用的模板。
3. 点 **Labels** 旁边的铅笔图标旁的 **Edit**。
4. 删除以下两个标签：
    - **template.kubevirt.io/type: "base"**
    - **template.kubevirt.io/version: "version"**
5. 点击 **Save**。
6. 点现有 **Annotations** 数旁边的铅笔图标。
7. 删除以下注解：
    - **template.kubevirt.io/deprecated**
8. 点击 **Save**。

7.1.4. 从命令行创建虚拟机

您可以通过编辑或创建 **VirtualMachine** 清单来从命令行创建虚拟机 (VM)。您可以使用虚拟机清单中的实例类型来简化虚拟机配置。



注意

您还可以使用 Web 控制台从实例类型创建虚拟机。

7.1.4.1. 使用 virtctl 工具创建清单

您可使用 **virtctl** CLI 实用程序简化为虚拟机、虚拟机实例类型和虚拟机首选项创建清单。如需更多信息，请参阅虚拟机清单创建命令。

7.1.4.2. 从 VirtualMachine 清单创建虚拟机

您可以从 **VirtualMachine** 清单创建虚拟机(VM)。

流程

1. 编辑虚拟机的 **VirtualMachine** 清单。以下示例配置 Red Hat Enterprise Linux (RHEL) 虚拟机：

注意

这个示例清单没有配置虚拟机身份验证。

RHEL 虚拟机的清单示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
 name: rhel-9-minimal
spec:
 dataVolumeTemplates:
  - metadata:
      name: rhel-9-minimal-volume
    spec:
      sourceRef:
        kind: DataSource
        name: rhel9 (1)
        namespace: openshift-virtualization-os-images (2)
      storage: {}
 instancetype:
  name: u1.medium (3)
 preference:
  name: rhel.9 (4)
 running: true
 template:
  spec:
    domain:
      devices: {}
    volumes:
      - dataVolume:
          name: rhel-9-minimal-volume
        name: rootdisk
```

(1) **rhel9** 金级镜像用于安装 RHEL 9 作为客户机操作系统。

2 金级镜像存储在 **openshift-virtualization-os-images** 命名空间中。

3 **u1.medium** 实例类型为虚拟机请求 1 个 vCPU 和 4Gi 内存。这些资源值不能在虚拟机中覆盖。

4 **rhel.9** 首选项指定支持 RHEL 9 客户机操作系统的额外属性。

2. 使用清单文件创建虚拟机：

```
$ oc create -f <vm_manifest_file>.yaml
```

3. 可选：启动虚拟机：

```
$ virtctl start <vm_name> -n <namespace>
```

后续步骤

- 配置对虚拟机的 SSH 访问

7.2. 从自定义镜像创建虚拟机

7.2.1. 从自定义镜像创建虚拟机概述

您可以使用以下方法之一从自定义操作系统镜像创建虚拟机(VM)：

- 从 registry 将镜像导入为容器磁盘。
  可选：您可以为容器磁盘启用自动更新。详情请参阅管理自动引导源更新。
- 从网页导入镜像。
- 从本地计算机上传镜像。
- 克隆包含镜像的持久性卷声明(PVC)。

Containerized Data Importer (CDI) 使用数据卷将镜像导入到 PVC 中。您可以使用 OpenShift Container Platform Web 控制台或命令行将 PVC 添加到虚拟机。

重要

您必须在从红帽提供的操作系统镜像创建的虚拟机上安装 QEMU 客户机代理。

您还必须在 Windows 虚拟机上安装 VirtIO 驱动程序。

QEMU 客户机代理包含在红帽镜像中。

7.2.2. 使用容器磁盘创建虚拟机

您可以使用从操作系统镜像构建的容器磁盘创建虚拟机 (VM)。

您可以为容器磁盘启用自动更新。详情请参阅管理自动引导源更新。

重要

如果容器磁盘较大，I/O 流量可能会增加，并导致 worker 节点不可用。您可以执行以下任务来解决这个问题：

- 修剪 **DeploymentConfig** 对象。
- 配置垃圾回收。

您可以通过执行以下步骤从容器磁盘创建虚拟机：

1. 将操作系统镜像构建到容器磁盘中，并将其上传到容器注册表。
2. 如果您的容器 registry 没有 TLS，请将您的环境配置为为您的 registry 禁用 TLS。
3. 使用 Web 控制台或命令行使用容器磁盘创建虚拟机作为磁盘源。

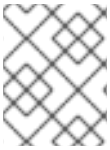
重要

您必须在从红帽提供的操作系统镜像创建的虚拟机上安装 QEMU 客户机代理。

7.2.2.1. 构建和上传容器磁盘

您可以将虚拟机(VM)镜像构建到容器磁盘中，并将其上传到 registry。

容器磁盘的大小受托管容器磁盘的 registry 的最大层大小的限制。



注意

对于 Red Hat Quay，您可以通过编辑首次部署 Red Hat Quay 时创建的 YAML 配置文件来更改最大层大小。

先决条件

- 必须安装 **podman** 。
- 您必须具有 QCOW2 或 RAW 镜像文件。

流程

1. 创建 Dockerfile 以将虚拟机镜像构建到容器镜像中。虚拟机镜像必须由 QEMU 所有，其 UID 为 **107**，并放置在容器内的 **/disk/** 目录中。**/disk/** 目录的权限必须设为 **0440**。
   以下示例在第一阶段使用 Red Hat Universal Base Image（UBI）来处理这些配置更改，并使用第二阶段中的最小 **scratch** 镜像存储结果：

```
$ cat > Dockerfile << EOF
FROM registry.access.redhat.com/ubi8/ubi:latest AS builder
ADD --chown=107:107 <vm_image>.qcow2 /disk/ 1
RUN chmod 0440 /disk/*

FROM scratch
COPY --from=builder /disk/* /disk/
EOF
```

1 其中 **<vm_image>** 是 QCOW2 或 RAW 格式的镜像。如果使用远程镜像，请将 **<vm_image>.qcow2** 替换为完整的 URL。

2. 构建和标记容器：

```
$ podman build -t <registry>/<container_disk_name>:latest .
```

3. 将容器镜像推送到 registry:

```
$ podman push <registry>/<container_disk_name>:latest
```

7.2.2.2. 为容器 registry 禁用 TLS

您可以通过编辑 **HyperConverged** 自定义资源的 **insecureRegistries** 字段来禁用一个或多个容器 registry 的 TLS（传输层安全）。

先决条件

1. 运行以下命令，在默认编辑器中打开 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 将不安全的 registry 列表添加到 **spec.storageImport.insecureRegistries** 字段中。

HyperConverged 自定义资源示例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  storageImport:
    insecureRegistries: 1
      - "private-registry-example-1:5000"
      - "private-registry-example-2:5000"
```

1 将此列表中的示例替换为有效的 registry 主机名。

7.2.2.3. 使用 Web 控制台从容器磁盘创建虚拟机

您可以使用 OpenShift Container Platform web 控制台从容器 registry 中导入容器磁盘来创建虚拟机 (VM)。

流程

1. 在 web 控制台中进入到 **Virtualization → Catalog**。
2. 点没有可用引导源的模板标题。
3. 点 **Customize VirtualMachine**。

4. 在 **Customize template parameters** 页面中，展开 **Storage**，然后从 **Disk source** 列表中选择 **Registry (creates PVC)**。
5. 输入容器镜像 URL。示例：**https://mirror.arizona.edu/fedora/linux/releases/38/Cloud/x86_64/images/Fedora-Cloud-Base-38-1.6.x86_64.qcow2**
6. 设置磁盘大小。
7. 点击 **Next**。
8. 点 **Create VirtualMachine**。

7.2.2.4. 使用命令行从容器磁盘创建虚拟机

您可以使用命令行从容器磁盘创建虚拟机 (VM)。

创建虚拟机 (VM) 时，容器磁盘的数据卷将导入到持久性存储中。

先决条件

- 您必须具有包含容器磁盘的容器 registry 的访问凭证。

流程

1. 如果容器 registry 需要身份验证，请创建一个 **Secret** 清单，指定凭证，并将其保存为 **data-source-secret.yaml** 文件：

```
apiVersion: v1
kind: Secret
metadata:
  name: data-source-secret
  labels:
    app: containerized-data-importer
type: Opaque
data:
  accessKeyId: "" 1
  secretKey:   "" 2
```

1. 指定以 Base64 编码的密钥 ID 或用户名。
2. 指定以 Base64 编码的 secret 密钥或密码。

2. 运行以下命令来应用 **Secret** 清单：

```
$ oc apply -f data-source-secret.yaml
```

3. 如果虚拟机必须与没有由系统 CA 捆绑包签名的证书的服务器通信，请在与虚拟机相同的命名空间中创建一个配置映射：

```
$ oc create configmap tls-certs 1
  --from-file=</path/to/file/ca.pem> 2
```

1. 指定配置映射名称。
2. 指定 CA 证书的路径。

4. 编辑 **VirtualMachine** 清单，并将它保存为 **vm-fedora-datavolume.yaml** 文件：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  creationTimestamp: null
  labels:
    kubevirt.io/vm: vm-fedora-datavolume
  name: vm-fedora-datavolume 1
spec:
  dataVolumeTemplates:
  - metadata:
      creationTimestamp: null
      name: fedora-dv 2
    spec:
      storage:
        resources:
          requests:
            storage: 10Gi 3
        storageClassName: <storage_class> 4
      source:
        registry:
          url: "docker://kubevirt/fedora-cloud-container-disk-demo:latest" 5
          secretRef: data-source-secret 6
          certConfigMap: tls-certs 7
    status: {}
  running: true
  template:
    metadata:
      creationTimestamp: null
      labels:
        kubevirt.io/vm: vm-fedora-datavolume
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: datavolumedisk1
        machine:
          type: ""
        resources:
          requests:
            memory: 1.5Gi
      terminationGracePeriodSeconds: 180
      volumes:
      - dataVolume:
          name: fedora-dv
        name: datavolumedisk1
status: {}
```

1. 指定虚拟机的名称。
2. 指定数据卷的名称。
3. 指定为数据卷请求的存储大小。
4. 可选：如果您没有指定存储类，则会使用默认存储类。
5. 指定容器 registry 的 URL。
6. 可选：如果您为容器 registry 访问凭证创建 secret，请指定 secret 名称。
7. 可选：指定一个 CA 证书配置映射。

5. 运行以下命令来创建虚拟机：

```
$ oc create -f vm-fedora-datavolume.yaml
```

oc create 命令创建数据卷和虚拟机。CDI 控制器创建一个带有正确注解和导入过程的底层 PVC。导入完成后，数据卷状态变为 **Succeeded**。您可以启动虚拟机。

数据卷置备在后台进行，因此无需监控进程。

验证

1. importer pod 从指定的 URL 下载容器磁盘，并将其存储在置备的持久性卷中。运行以下命令，查看 importer pod 的状态：

```
$ oc get pods
```

2. 运行以下命令监控数据卷，直到其状态为 **Succeeded** ：

```
$ oc describe dv fedora-dv 1
```

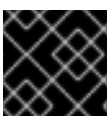
1. 指定您在 **VirtualMachine** 清单中定义的数据卷名称。

3. 通过访问其串行控制台来验证置备是否已完成，以及虚拟机是否已启动：

```
$ virtctl console vm-fedora-datavolume
```

7.2.3. 通过从网页导入镜像来创建虚拟机

您可以通过从 web 页面导入操作系统镜像来创建虚拟机 (VM)。



重要

您必须在从红帽提供的操作系统镜像创建的虚拟机上安装 QEMU 客户机代理。

7.2.3.1. 使用 Web 控制台从网页上的镜像创建虚拟机

您可以使用 OpenShift Container Platform Web 控制台从网页导入镜像来创建虚拟机 (VM)。

先决条件

- 您必须有权访问包含镜像的网页。

流程

1. 在 web 控制台中进入到 **Virtualization → Catalog**。
2. 点没有可用引导源的模板标题。
3. 点 **Customize VirtualMachine**。
4. 在 **Customize template parameters** 页面中，展开 **Storage**，然后从 **Disk source** 列表中选择 **URL (creates PVC)**。
5. 输入镜像 URL。示例：**https://access.redhat.com/downloads/content/69/ver=/rhel---7/7.9/x86_64/product-software**
6. 输入容器镜像 URL。示例：**https://mirror.arizona.edu/fedora/linux/releases/38/Cloud/x86_64/images/Fedora-Cloud-Base-38-1.6.x86_64.qcow2**
7. 设置磁盘大小。
8. 点击 **Next**。
9. 点 **Create VirtualMachine**。

7.2.3.2. 使用命令行从网页上的镜像创建虚拟机

您可以使用命令行从网页中的镜像创建虚拟机 (VM)。

创建虚拟机 (VM) 时，带有镜像的数据卷将导入到持久性存储中。

先决条件

- 您必须有包含镜像的网页的访问凭证。

流程

1. 如果网页需要身份验证，请创建一个 **Secret** 清单，指定凭证，并将其保存为 **data-source-secret.yaml** 文件：

```
apiVersion: v1
kind: Secret
metadata:
  name: data-source-secret
  labels:
    app: containerized-data-importer
type: Opaque
data:
  accessKeyId: "" 1
  secretKey:   "" 2
```

1 指定以 Base64 编码的密钥 ID 或用户名。

2 指定以 Base64 编码的 secret 密钥或密码。

2. 运行以下命令来应用 **Secret** 清单：

```
$ oc apply -f data-source-secret.yaml
```

3. 如果虚拟机必须与没有由系统 CA 捆绑包签名的证书的服务器通信，请在与虚拟机相同的命名空间中创建一个配置映射：

```
$ oc create configmap tls-certs 1
  --from-file=</path/to/file/ca.pem> 2
```

1 指定配置映射名称。

2 指定 CA 证书的路径。

4. 编辑 **VirtualMachine** 清单，并将它保存为 **vm-fedora-datavolume.yaml** 文件：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  creationTimestamp: null
  labels:
    kubevirt.io/vm: vm-fedora-datavolume
  name: vm-fedora-datavolume 1
spec:
  dataVolumeTemplates:
  - metadata:
      creationTimestamp: null
      name: fedora-dv 2
    spec:
      storage:
        resources:
          requests:
            storage: 10Gi 3
        storageClassName: <storage_class> 4
      source:
        http:
          url: "https://mirror.arizona.edu/fedora/linux/releases/35/Cloud/x86_64/images/Fedora-Cloud-Base-35-1.2.x86_64.qcow2" 5
        registry:
          url: "docker://kubevirt/fedora-cloud-container-disk-demo:latest" 6
          secretRef: data-source-secret 7
          certConfigMap: tls-certs 8
    status: {}
  running: true
  template:
    metadata:
      creationTimestamp: null
      labels:
        kubevirt.io/vm: vm-fedora-datavolume
    spec:
```

```
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: datavolumedisk1
        machine:
          type: ""
        resources:
          requests:
            memory: 1.5Gi
      terminationGracePeriodSeconds: 180
      volumes:
      - dataVolume:
          name: fedora-dv
        name: datavolumedisk1
status: {}
```

1. 指定虚拟机的名称。
2. 指定数据卷的名称。
3. 指定为数据卷请求的存储大小。
4. 可选：如果您没有指定存储类，则会使用默认存储类。
5. 6. 指定网页的 URL。
7. 可选：如果您为 Web 页面访问凭证创建了 secret，请指定 secret 名称。
8. 可选：指定一个 CA 证书配置映射。

5. 运行以下命令来创建虚拟机：

```
$ oc create -f vm-fedora-datavolume.yaml
```

oc create 命令创建数据卷和虚拟机。CDI 控制器创建一个带有正确注解和导入过程的底层 PVC。导入完成后，数据卷状态变为 **Succeeded**。您可以启动虚拟机。

数据卷置备在后台进行，因此无需监控进程。

验证

1. importer pod 从指定的 URL 下载镜像，并将其存储在置备的持久性卷上。运行以下命令，查看 importer pod 的状态：

```
$ oc get pods
```

2. 运行以下命令监控数据卷，直到其状态为 **Succeeded** ：

```
$ oc describe dv fedora-dv 1
```

1. 指定您在 **VirtualMachine** 清单中定义的数据卷名称。

3. 通过访问其串行控制台来验证置备是否已完成，以及虚拟机是否已启动：

```
$ virtctl console vm-fedora-datavolume
```

7.2.4. 通过上传镜像来创建虚拟机

您可以通过从本地机器上传操作系统镜像来创建虚拟机 (VM)。

您可以通过上传 Windows 镜像到 PVC 来创建 Windows 虚拟机。然后，在创建虚拟机时克隆 PVC。

重要

您必须在从红帽提供的操作系统镜像创建的虚拟机上安装 QEMU 客户机代理。

您还必须在 Windows 虚拟机上安装 VirtIO 驱动程序。

7.2.4.1. 使用 Web 控制台从上传的镜像创建虚拟机

您可以使用 OpenShift Container Platform web 控制台从上传的操作系统镜像创建虚拟机 (VM)。

先决条件

- 您必须有一个 **IMG**、**ISO** 或 **QCOW2** 镜像文件。

流程

1. 在 web 控制台中进入到 **Virtualization → Catalog**。
2. 点没有可用引导源的模板标题。
3. 点 **Customize VirtualMachine**。
4. 在 **Customize template parameters** 页面中，展开 **Storage**，然后从 **Disk source** 列表中选择 **Upload (Upload a new file to a PVC)**。
5. 浏览到本地机器上的镜像并设置磁盘大小。
6. 点 **Customize VirtualMachine**。
7. 点 **Create VirtualMachine**。

7.2.4.2. 创建 Windows 虚拟机

您可以通过上传 Windows 镜像到持久性卷声明 (PVC) 来创建 Windows 虚拟机，然后使用 OpenShift Container Platform web 控制台创建虚拟机时克隆 PVC。

先决条件

- 已使用 Windows Media Creation Tool 创建 Windows 安装 DVD 或者 USB。请参阅 Microsoft 文档中的创建 Windows 10 安装介质 。
- 您创建了 **autounattend.xml** 回答文件。请参阅 Microsoft 文档中的回答文件(unattend.xml)。

流程

1. 将 Windows 镜像上传为新 PVC：
   a. 在 web 控制台中进入到 **Storage → PersistentVolumeClaims**。
   b. 点 **Create PersistentVolumeClaim → With Data upload form**。
   c. 浏览 Windows 镜像并选择它。
   d. 输入 PVC 名称，选择存储类和大小，然后点 **Upload**。
   Windows 镜像上传到 PVC。
2. 通过克隆上传的 PVC 来配置新虚拟机：
   a. 进入到 **Virtualization → Catalog**。
   b. 选择 Windows 模板标题并点 **Customize VirtualMachine**。
   c. 从 **Disk source** 列表中选择 **Clone (clone PVC)**。
   d. 选择 PVC 项目、Windows 镜像 PVC 和磁盘大小。
3. 将回答文件应用到虚拟机：
   a. 点 **Customize VirtualMachine 参数**。
   b. 在 **Scripts** 选项卡的 **Sysprep** 部分，点 **Edit**。
   c. 浏览到 **autounattend.xml** 回答文件，然后点**保存**。
4. 设置虚拟机的 run 策略：
   a. 清除 **Start this VirtualMachine after creation**，以便虚拟机不会立即启动。
   b. 点 **Create VirtualMachine**。
   c. 在 **YAML** 标签页中，将 **running:false** 替换为 **runStrategy: RerunOnFailure**，点 Save。
5. 点选项菜单  并选择 **Start**。
   虚拟机从包含 **autounattend.xml** 回答文件的 **sysprep** 磁盘引导。

7.2.4.2.1. 常规化 Windows 虚拟机镜像

在使用镜像创建新虚拟机前，您可以常规化 Windows 操作系统镜像删除所有特定于系统的配置数据。

在常规调整虚拟机前，您必须确保 **sysprep** 工具在无人值守的 Windows 安装后无法检测到应答文件。

先决条件

- 正在运行的 Windows 虚拟机安装有 QEMU 客户机代理。

流程

1. 在 OpenShift Container Platform 控制台中点 **Virtualization → VirtualMachines**。
2. 选择 Windows 虚拟机以打开 **VirtualMachine 详情**页。

3. 点 **Configuration** → **Disks**。
4. 点 **sysprep** 磁盘  旁边的 Options 菜单并选择 **Detach**。
5. 单击 **Detach**。
6. 重命名 **C:\Windows\Panther\unattend.xml** 以避免 **sysprep** 工具对其进行检测。
7. 运行以下命令启动 **sysprep** 程序：

```
%WINDIR%\System32\Sysprep\sysprep.exe /generalize /shutdown /oobe /mode:vm
```

8. **sysprep** 工具完成后，Windows 虚拟机将关闭。VM 的磁盘镜像现在可作为 Windows 虚拟机的安装镜像使用。

现在，您可以对虚拟机进行特殊化。

7.2.4.2.2. 特殊化 Windows 虚拟机镜像

特殊化 Windows 虚拟机 (VM) 配置从常规化 Windows 镜像到虚拟机中的计算机特定信息。

先决条件

- 您必须有一个通用的 Windows 磁盘镜像。
- 您必须创建一个 **unattend.xml** 回答文件。详情请查看 Microsoft 文档。

流程

1. 在 OpenShift Container Platform 控制台中点 **Virtualization** → **Catalog**。
2. 选择 Windows 模板并点 **Customize VirtualMachine**。
3. 从 **Disk source** 列表中选择 **PVC(clone PVC)**。
4. 选择通用 Windows 镜像的 PVC 项目和 PVC 名称。
5. 点 **Customize VirtualMachine 参数**。
6. 点 **Scripts** 选项卡。
7. 在 **Sysprep** 部分中，点 **Edit**，浏览到 **unattend.xml** 回答文件，然后点**保存**。
8. 点 **Create VirtualMachine**。

在初次启动过程中，Windows 使用 **unattend.xml** 回答文件来专注于虚拟机。虚拟机现在可供使用。

创建 Windows 虚拟机的其他资源

- Microsoft, Sysprep(Generalize)Windows 安装
- Microsoft，常规化
- Microsoft, specialize

7.2.4.3. 使用命令行从上传的镜像创建虚拟机

您可使用 **virtctl** 命令行工具上传操作系统镜像。您可以使用现有数据卷，或为镜像创建新数据卷。

先决条件

- 您必须有一个 **ISO**、**IMG** 或 **QCOW2** 操作系统镜像文件。
- 为获得最佳性能，请使用 virt-sparsify 工具或 **xz** 或 **gzip** 工具压缩镜像文件。
- 已安装 **virtctl**。
- 客户端机器必须配置为信任 OpenShift Container Platform 路由器的证书。

流程

1. 运行 **virtctl image-upload** 命令上传镜像：

```
$ virtctl image-upload dv <datavolume_name> \ 1
  --size=<datavolume_size> \ 2
  --image-path=</path/to/image> \ 3
```

1. 数据卷的名称。
2. 数据卷的大小。例如： **--size=500Mi**, **--size=1G**
3. 镜像的文件路径。

注意

- 如果您不想创建新数据卷，请省略 **--size** 参数，并包含 **--no-create** 标志。
- 将磁盘镜像上传到 PVC 时，PVC 大小必须大于未压缩的虚拟磁盘的大小。
- 若要在使用 HTTPS 时允许不安全的服务器连接，请使用 **--insecure** 参数。当您使用 **--insecure** 标志时，**不会**验证上传端点的真实性。

2. 可选。要验证数据卷是否已创建，运行以下命令来查看所有数据卷：

```
$ oc get dvs
```

7.2.5. 安装 QEMU 客户机代理和 VirtIO 驱动程序

QEMU 客户机代理是在虚拟机 (VM) 上运行的守护进程，并将信息传递给有关虚拟机、用户、文件系统和从属网络的信息。

您必须在从红帽提供的操作系统镜像创建的虚拟机上安装 QEMU 客户机代理。

7.2.5.1. 安装 QEMU 客户机代理

7.2.5.1.1. 在 Linux 虚拟机上安装 QEMU 客户机代理

qemu-guest-agent 广泛可用，默认在 Red Hat Enterprise Linux (RHEL) 虚拟机 (VM) 中可用。安装代理并启动服务。

注意

要为具有最高完整性的在线（Running 状态）虚拟机创建快照，请安装 QEMU 客户机代理。

QEMU 客户机代理通过尝试静止虚拟机的文件系统来尽可能取一个一致的快照，具体取决于系统工作负载。这样可确保在进行快照前将 in-flight I/O 写入磁盘。如果没有客户机代理，则无法静止并生成最佳快照。执行快照的条件反映在 web 控制台或 CLI 中显示的快照声明中。

流程

1. 使用控制台或 SSH 登录虚拟机。
2. 运行以下命令来安装 QEMU 客户机代理：

```
$ yum install -y qemu-guest-agent
```

3. 确保服务持久并启动它：

```
$ systemctl enable --now qemu-guest-agent
```

验证

- 运行以下命令，以验证 **AgentConnected** 是否列在 VM spec 中：

```
$ oc get vm <vm_name>
```

7.2.5.1.2. 在 Windows 虚拟机上安装 QEMU 客户机代理

对于 Windows 虚拟机，QEMU 客户机代理包含在 VirtIO 驱动程序中。您可以在 Windows 安装过程中或现有 Windows 虚拟机上安装驱动程序。

注意

要为具有最高完整性的在线（Running 状态）虚拟机创建快照，请安装 QEMU 客户机代理。

QEMU 客户机代理通过尝试静止虚拟机的文件系统来尽可能取一个一致的快照，具体取决于系统工作负载。这样可确保在进行快照前将 in-flight I/O 写入磁盘。如果没有客户机代理，则无法静止并生成最佳快照。执行快照的条件反映在 web 控制台或 CLI 中显示的快照声明中。

流程

1. 在 Windows 客户机操作系统中，使用 **File Explorer** 进入到 **virtio-win** CD 驱动器中的 **guest-agent** 目录。
2. 运行 **qemu-ga-x86_64.msi** 安装程序。

验证

1. 运行以下命令来获取网络服务列表：

```
$ net start
```

2. 验证输出是否包含 **QEMU 客户机代理**。

7.2.5.2. 在 Windows 虚拟机上安装 VirtIO 驱动程序

VirtIO 驱动程序是 Microsoft Windows 虚拟机在 OpenShift Virtualization 中运行时所需的半虚拟化设备驱动程序。驱动程序由其余镜像提供，不需要单独下载。

必须将 **container-native-virtualization/virtio-win** 容器磁盘作为 SATA CD 驱动器附加到虚拟机，以启用驱动程序安装。您可以在安装过程中安装 VirtIO 驱动程序，或添加到现有 Windows 安装中。

安装驱动程序后，可从虚拟机中移除 **container-native-virtualization/virtio-win** 容器磁盘。

表 7.3. 支持的驱动程序

驱动程序名称	硬件 ID	描述
viostor	VEN_1AF4&DEV_1001 VEN_1AF4&DEV_1042	块驱动程序。有时在 **Other devices** 组中被标记为 **SCSI Controller**。
viorng	VEN_1AF4&DEV_1005 VEN_1AF4&DEV_1044	熵源（entropy）驱动程序。有时在 **Other devices** 组中被标记为 **PCI** 设备。
NetKVM	VEN_1AF4&DEV_1000 VEN_1AF4&DEV_1041	网络驱动程序。有时，在 **Other devices** 组中被标记为 **Ethernet Controller**。仅在配置了 VirtIO NIC 时可用。

7.2.5.2.1. 在安装过程中将 VirtIO 容器磁盘附加到 Windows 虚拟机

您必须将 VirtIO 容器磁盘附加到 Windows 虚拟机，以安装必要的 Windows 驱动程序。这可以在创建虚拟机时完成。

流程

1. 从模板创建 Windows 虚拟机时，点 **Customize VirtualMachine**。
2. 选择 **Mount Windows 驱动程序磁盘**。
3. 点 **Customize VirtualMachine 参数**。
4. 点 **Create VirtualMachine**。

创建虚拟机后，**virtio-win** SATA CD 磁盘将附加到虚拟机。

7.2.5.2.2. 将 VirtIO 容器磁盘附加到现有的 Windows 虚拟机

您必须将 VirtIO 容器磁盘附加到 Windows 虚拟机，以安装必要的 Windows 驱动程序。这可以对现有的虚拟机完成。

流程

1. 导航到现有的 Windows 虚拟机，然后点 **Actions → Stop**。
2. 进入 **VM Details → Configuration → Disks**，然后点 **Add disk**。
3. 从容器源添加 **windows-driver-disk**，将 **Type** 设置为 **CD-ROM**，然后将 **Interface** 设置为 **SATA**。
4. 点击 **Save**。
5. 启动虚拟机并连接到图形控制台。

7.2.5.2.3. 在 Windows 安装过程中安装 VirtIO 驱动程序

您可以在虚拟机 (VM) 上安装 Windows 时安装 VirtIO 驱动程序。

注意

该流程使用通用方法安装 Windows，且安装方法可能因 Windows 版本而异。有关您要安装的 Windows 版本，请参阅相关文档。

先决条件

- 包含 **virtio** 驱动程序的存储设备必须附加到虚拟机。

流程

1. 在 Windows 操作系统中，使用 **File Explorer** 进入到 **virtio-win** CD 驱动器。
2. 双击该驱动器为您的虚拟机运行适当的安装程序。
   对于 64 位 vCPU，请选择 **virtio-win-gt-x64** 安装程序。不再支持 32 位 vCPU。
3. 可选：在安装程序的 **Custom Setup** 步骤中，选择您要安装的设备驱动程序。推荐的驱动程序集会被默认选择。
4. 安装完成后，选择 **Finish**。
5. 重启虚拟机。

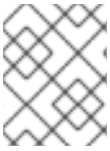
验证

1. 在 PC 上打开系统磁盘。这通常是 **C:**。
2. 进入到 **Program Files → Virtio-Win**。

如果 **Virtio-Win** 目录存在并包含每个驱动程序的子目录，则安装可以成功。

7.2.5.2.4. 在现有 Windows 虚拟机上从 SATA CD 驱动器安装 VirtIO 驱动程序

您可以从现有 Windows 虚拟机(VM)上的 SATA CD 驱动器安装 VirtIO 驱动程序。



注意

该流程使用通用方法为 Windows 添加驱动。有关具体安装步骤，请参阅您的 Windows 版本安装文档。

先决条件

- 包含 virtio 驱动程序的存储设备必须作为 SATA CD 驱动器附加到虚拟机。

流程

1. 启动虚拟机并连接到图形控制台。
2. 登录 Windows 用户会话。
3. 打开 **Device Manager** 并展开 **Other devices** 以列出所有 **Unknown device**。
   a. 打开 **Device Properties** 以识别未知设备。
   b. 右击设备并选择 **Properties**。
   c. 单击 **Details** 选项卡，并在 **Property** 列表中选择 **Hardware Ids**。
   d. 将 **Hardware Ids** 的 **Value** 与受支持的 VirtIO 驱动程序相比较。
4. 右击设备并选择 **Update Driver Software**。
5. 点击 **Browse my computer for driver software**并浏览所附加的 VirtIO 驱动程序所在 SATA CD 驱动器。驱动程序将按照其驱动程序类型、操作系统和 CPU 架构分层排列。
6. 点击 **Next** 以安装驱动程序。
7. 对所有必要 VirtIO 驱动程序重复这一过程。
8. 安装完驱动程序后，点击 **Close** 关闭窗口。
9. 重启虚拟机以完成驱动程序安装。

7.2.5.2.5. 从添加为 SATA CD 驱动器的容器磁盘安装 VirtIO 驱动程序

您可以从作为 SATA CD 驱动器添加到 Windows 虚拟机(VM)的容器磁盘中安装 VirtIO 驱动程序。

提示

从红帽生态系统目录 下载 **container-native-virtualization/virtio-win** 容器磁盘不是必须的，因为如果集群中不存在容器磁盘，则会从红帽 registry 下载容器磁盘。但是，下载可减少安装时间。

先决条件

- 您必须在受限环境中访问红帽 registry 或下载的 **container-native-virtualization/virtio-win** 容器磁盘。

流程

1. 通过编辑 **VirtualMachine** 清单将 **container-native-virtualization/virtio-win** 容器磁盘添加为 CD 驱动器：

```
# ...
spec:
  domain:
    devices:
      disks:
        - name: virtiocontainerdisk
          bootOrder: 2 1
          cdrom:
            bus: sata
volumes:
  - containerDisk:
      image: container-native-virtualization/virtio-win
    name: virtiocontainerdisk
```

1 OpenShift Virtualization 按照 **VirtualMachine** 清单中定义的顺序引导虚拟机磁盘。您可以定义在 **container-native-virtualization/virtio-win** 容器磁盘前引导的其他虚拟机磁盘，或使用可选的 **bootOrder** 参数来确保虚拟机从正确的磁盘启动。如果为磁盘配置引导顺序，您必须为其他磁盘配置引导顺序。

2. 应用更改：
   - 如果虚拟机没有运行，请运行以下命令：

     ```
     $ virtctl start <vm> -n <namespace>
     ```

   - 如果虚拟机正在运行，重启虚拟机或运行以下命令：

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

3. 虚拟机启动后，从 SATA CD 驱动器安装 VirtIO 驱动程序。

7.2.5.3. 更新 VirtIO 驱动程序

7.2.5.3.1. 更新 Windows 虚拟机上的 VirtIO 驱动程序

使用 Windows Update 服务更新 Windows 虚拟机(VM)上的 **virtio** 驱动程序。

先决条件

- 集群必须连接到互联网。断开连接的集群无法访问 Windows Update 服务。

流程

1. 在 Windows Guest 操作系统中，点 **Windows** 密钥并选择 **Settings**。
2. 进入到 **Windows Update → Advanced Options → Optional Updates**。
3. 安装 **Red Hat, Inc.** 的所有更新。
4. 重启虚拟机。

验证

1. 在 Windows 虚拟机上，进入到 **设备管理器**。
2. 选择一个设备。
3. 选择 **Driver** 选项卡。
4. 点 **Driver Details**，并确认 **virtio** 驱动程序详情显示了正确的版本。

7.2.6. 克隆虚拟机

您可以克隆虚拟机(VM)或从快照创建新虚拟机。

7.2.6.1. 使用 Web 控制台克隆虚拟机

您可以使用 Web 控制台克隆现有虚拟机。

流程

1. 在 web 控制台中进入到 **Virtualization → VirtualMachines**。
2. 选择一个虚拟机以打开 **VirtualMachine 详情**页。
3. 点 **Actions**。
4. 选择 **Clone**。
5. 在 **Clone VirtualMachine** 页面中，输入新虚拟机的名称。
6. （可选）选择 **Start cloned VM**复选框来启动克隆的虚拟机。
7. 单击 **Clone**。

7.2.6.2. 使用 Web 控制台从现有快照创建虚拟机

您可以通过复制现有快照来创建新虚拟机。

流程

1. 在 web 控制台中进入到 **Virtualization → VirtualMachines**。
2. 选择一个虚拟机以打开 **VirtualMachine 详情**页。
3. 点 **Snapshots** 标签页。
4. 对于您要复制的快照，点操作菜单  。
5. 选择 **Create VirtualMachine**。
6. 输入虚拟机的名称。
7. （可选）选择 **Start this VirtualMachine after creation** 来启动新的虚拟机。
8. 点 **Create**。

7.2.6.3. 其他资源

- 通过克隆 PVC 创建虚拟机

7.2.7. 通过克隆 PVC 创建虚拟机

您可以通过使用自定义镜像克隆现有持久性卷声明 (PVC) 来创建虚拟机 (VM)。

您必须在从红帽提供的操作系统镜像创建的虚拟机上安装 QEMU 客户机代理。

您可以通过创建一个引用源 PVC 的数据卷来克隆 PVC。

7.2.7.1. 关于克隆

在克隆数据卷时，Containerized Data Importer (CDI)选择以下 Container Storage Interface (CSI)克隆方法之一：

- CSI 卷克隆
- 智能克隆

CSI 卷克隆和智能克隆方法都非常高效，但使用它们会有一定的要求。如果没有满足要求，CDI 将使用主机辅助克隆。主机辅助克隆是最慢且效率最低的克隆方法，但使用它的要求比其它两种克隆方法要少。

7.2.7.1.1. CSI 卷克隆

Container Storage Interface (CSI) 克隆使用 CSI 驱动程序功能更有效地克隆源数据卷。

CSI 卷克隆有以下要求：

- 支持持久性卷声明(PVC)的存储类的 CSI 驱动程序必须支持卷克隆。
- 对于 CDI 无法识别的置备程序，对应的存储配置集必须将 **cloneStrategy** 设置为 CSI Volume Cloning。
- 源和目标 PVC 必须具有相同的存储类和卷模式。
- 如果创建数据卷，则必须有在源命名空间中创建 **datavolumes/source** 资源的权限。
- 源卷不能在使用中。

7.2.7.1.2. 智能克隆

当有快照功能的 Container Storage Interface (CSI) 插件时，Containerized Data Importer (CDI) 会从快照创建一个持久性卷声明 (PVC)，然后允许有效地克隆额外的 PVC。

智能克隆有以下要求：

- 与存储类关联的快照类必须存在。
- 源和目标 PVC 必须具有相同的存储类和卷模式。
- 如果创建数据卷，则必须有在源命名空间中创建 **datavolumes/source** 资源的权限。
- 源卷不能在使用中。

7.2.7.1.3. 主机辅助克隆

当没有满足 Container Storage Interface (CSI) 卷克隆或智能克隆的要求时，主机辅助克隆将用作回退方法。主机辅助克隆比其他两个克隆方法之一更高效。

主机辅助克隆使用源 pod 和目标 pod 将数据从源卷复制到目标卷。目标持久性卷声明 (PVC) 使用回退原因标注，该原因解释了使用主机辅助克隆的原因，并创建事件。

PVC 目标注解示例

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    cdi.kubevirt.io/cloneFallbackReason: The volume modes of source and target are incompatible
    cdi.kubevirt.io/clonePhase: Succeeded
    cdi.kubevirt.io/cloneType: copy
```

事件示例

```
NAMESPACE   LAST SEEN   TYPE      REASON                  OBJECT                                MESSAGE
test-ns     0s          Warning   IncompatibleVolumeModes persistentvolumeclaim/test-target     The volume modes of source and target are incompatible
```

7.2.7.2. 使用 Web 控制台从 PVC 创建虚拟机

您可以使用 OpenShift Container Platform Web 控制台从网页导入镜像来创建虚拟机 (VM)。您可以使用 OpenShift Container Platform web 控制台克隆持久性卷声明 (PVC) 来创建虚拟机 (VM)。

先决条件

- 您必须有权访问包含镜像的网页。
- 您必须有权访问包含源 PVC 的命名空间。

流程

1. 在 web 控制台中进入到 **Virtualization → Catalog**。
2. 点没有可用引导源的模板标题。
3. 点 **Customize VirtualMachine**。
4. 在 **Customize template parameters** 页面中，展开 **Storage**，然后从 **Disk source** 列表中选择 **PVC (clone PVC)**。
5. 输入镜像 URL。示例：**https://access.redhat.com/downloads/content/69/ver=/rhel---7/7.9/x86_64/product-software**
6. 输入容器镜像 URL。示例：**https://mirror.arizona.edu/fedora/linux/releases/38/Cloud/x86_64/images/Fedora-Cloud-Base-38-1.6.x86_64.qcow2**
7. 选择 PVC 项目和 PVC 名称。

8. 设置磁盘大小。
9. 点击 **Next**。
10. 点 **Create VirtualMachine**。

7.2.7.3. 使用命令行从 PVC 创建虚拟机

您可以使用命令行克隆现有虚拟机的持久性卷声明 (PVC) 来创建虚拟机 (VM)。

您可以使用以下选项之一克隆 PVC：

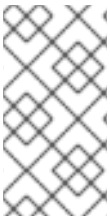
- 将 PVC 克隆到新数据卷中。
  这个方法会创建一个独立于原始虚拟机的数据卷。删除原始虚拟机不会影响新数据卷或者关联的 PVC。
- 通过使用 **dataVolumeTemplates** 小节创建 **VirtualMachine** 清单来克隆 PVC。
  这个方法会创建一个数据卷，其生命周期取决于原始虚拟机。删除原始虚拟机会删除克隆的数据卷及其关联的 PVC。

7.2.7.3.1. 将 PVC 克隆到数据卷中

您可以使用命令行将现有虚拟机 (VM) 磁盘的持久性卷声明 (PVC) 克隆到数据卷中。

您可以创建一个引用原始源 PVC 的数据卷。新数据卷的生命周期独立于原始虚拟机。删除原始虚拟机不会影响新数据卷或者关联的 PVC。

主机辅助克隆支持在不同卷模式间进行克隆，如从块持久性卷 (PV) 克隆到文件系统 PV，只要源和目标 PV 属于 **kubevirt** 内容类型。



注意

智能克隆比主机辅助克隆更快、效率更高，因为它使用快照克隆 PVC。支持快照的存储供应商支持智能克隆，如 Red Hat OpenShift Data Foundation。

对于智能克隆，不支持在不同卷模式间进行克隆。

先决条件

- 带有源 PVC 的虚拟机必须被关闭。
- 如果将 PVC 克隆到不同的命名空间中，则必须具有在目标命名空间中创建资源的权限。
- smart-cloning 的额外先决条件：
  - 您的存储供应商必须支持快照。
  - 源和目标 PVC 必须具有相同的存储供应商和卷模式。
  - **VolumeSnapshotClass** 对象的 **driver** 键的值必须与 **StorageClass** 对象的 **provisioner** 键的值匹配，如下例所示：

    VolumeSnapshotClass 对象示例

    ```
    kind: VolumeSnapshotClass
    apiVersion: snapshot.storage.k8s.io/v1
    ```

```
driver: openshift-storage.rbd.csi.ceph.com
# ...
```

StorageClass 对象示例

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
# ...
provisioner: openshift-storage.rbd.csi.ceph.com
```

流程

1. 如以下示例所示创建 **DataVolume** 清单：

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <datavolume> 1
spec:
  source:
    pvc:
      namespace: "<source_namespace>" 2
      name: "<my_vm_disk>" 3
  storage: {}
```

1. 指定新数据卷的名称。
2. 指定源 PVC 的命名空间。
3. 指定源 PVC 的名称。

2. 运行以下命令来创建数据卷：

```
$ oc create -f <datavolume>.yaml
```

注意

数据卷可防止虚拟机在 PVC 准备好前启动。您可以创建一个在克隆 PVC 时引用新数据卷的虚拟机。

7.2.7.3.2. 使用数据卷模板从克隆的 PVC 创建虚拟机

您可以创建一个虚拟机 (VM) 来使用数据卷模板克隆现有虚拟机的持久性卷声明 (PVC)。

这个方法会创建一个数据卷，其生命周期取决于原始虚拟机。删除原始虚拟机会删除克隆的数据卷及其关联的 PVC。

先决条件

- 带有源 PVC 的虚拟机必须被关闭。

流程

1. 如以下示例所示，创建一个 **VirtualMachine** 清单：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm-dv-clone
  name: vm-dv-clone (1)
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm-dv-clone
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: root-disk
        resources:
          requests:
            memory: 64M
      volumes:
      - dataVolume:
          name: favorite-clone
        name: root-disk
  dataVolumeTemplates:
  - metadata:
      name: favorite-clone
    spec:
      storage:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 2Gi
      source:
        pvc:
          namespace: <source_namespace> (2)
          name: "<source_pvc>" (3)
```

(1) 指定虚拟机的名称。

(2) 指定源 PVC 的命名空间。

(3) 指定源 PVC 的名称。

2. 使用 PVC 克隆的数据卷创建虚拟机：

```
$ oc create -f <vm-clone-datavolumetemplate>.yaml
```

7.3. 连接到虚拟机控制台

您可以连接到以下控制台来访问正在运行的虚拟机 (VM)：

- VNC 控制台
- 串行控制台
- Windows 虚拟机的桌面视图

7.3.1. 连接至 VNC 控制台

您可以使用 OpenShift Container Platform web 控制台或 **virtctl** 命令行工具连接到虚拟机的 VNC 控制台。

7.3.1.1. 使用 Web 控制台连接到 VNC 控制台

您可以使用 OpenShift Container Platform Web 控制台连接至虚拟机的 VNC 控制台。

注意

如果您使用分配了介质设备的 vGPU 连接到 Windows 虚拟机，您可以在默认显示和 vGPU 显示间切换。

流程

1. 在 **Virtualization → VirtualMachines** 页面中，点虚拟机打开 **VirtualMachine 详情**页。
2. 点击 **Console** 选项卡。VNC 控制台会话会自动启动。
3. 可选： 要切换到 Windows 虚拟机的 vGPU 显示，请从 **Send key** 列表中选择 **Ctl + Alt + 2**。
   - 从 **Send key** 列表中选择 **Ctl + Alt + 1**以恢复默认显示。
4. 要结束控制台会话，请点控制台窗格外，然后点 **Disconnect**。

7.3.1.2. 使用 virtctl 连接到 VNC 控制台

您可使用 **virtctl** 命令行工具连接到正在运行的虚拟机的 VNC 控制台。

注意

如果您通过 SSH 连接在远程机器上运行 **virtctl vnc** 命令，则必须使用 **-X** 或 **-Y** 标志运行 **ssh** 命令，将 X 会话转发到本地机器。

先决条件

- 您必须安装 **virt-viewer** 软件包。

流程

1. 运行以下命令以启动控制台会话：

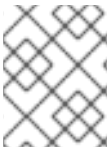
```
$ virtctl vnc <vm_name>
```

2. 如果连接失败，请运行以下命令来收集故障排除信息：

```
$ virtctl vnc <vm_name> -v 4
```

7.3.1.3. 为 VNC 控制台生成临时令牌

要访问虚拟机的 VNC，请为 Kubernetes API 生成临时身份验证 bearer 令牌。



注意

Kubernetes 还支持通过修改 curl 命令来使用客户端证书进行身份验证，而不是 bearer 令牌。

先决条件

- 一个运行的虚拟机，带有 OpenShift Virtualization 4.14 或更高版本，**ssp-operator** 4.14 或更高版本

流程

1. 在 HyperConverged (**HCO**)自定义资源(CR)中启用功能门：

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv --type json -p '[{"op":
"replace", "path": "/spec/featureGates/deployVmConsoleProxy", "value": true}]'
```

2. 输入以下命令生成令牌：

```
$ curl --header "Authorization: Bearer ${TOKEN}" \
    "https://api.
<cluster_fqdn>/apis/token.kubevirt.io/v1alpha1/namespaces/<namespace>/virtualmachines/<vm
_name>/vnc?duration=<duration>"
```

<duration> 参数可以以小时和分钟为单位设置，最小持续时间为 10 分钟。例如： **5h30m**。如果没有设置此参数，令牌默认为 10 分钟有效。

输出示例：

```
{ "token": "eyJhb..." }
```

3. 可选：使用输出中提供的令牌来创建变量：

```
$ export VNC_TOKEN="<token>"
```

现在，您可以使用令牌来访问虚拟机的 VNC 控制台。

验证

1. 输入以下命令登录到集群：

```
$ oc login --token ${VNC_TOKEN}
```

2. 使用 **virtctl** 命令测试对虚拟机的 VNC 控制台的访问：

```
$ virtctl vnc <vm_name> -n <namespace>
```

警告

目前无法撤销特定的令牌。

要撤销令牌，您必须删除用于创建令牌的服务帐户。但是，这也会撤销使用服务帐户创建的所有令牌。请谨慎使用以下命令：

```
$ virtctl delete serviceaccount --namespace "<namespace>" "<vm_name>-vnc-access"
```

7.3.1.3.1. 使用集群角色为 VNC 控制台授予令牌生成权限

作为集群管理员，您可以安装集群角色并将其绑定到用户或服务帐户，以允许访问为 VNC 控制台生成令牌的端点。

流程

- 选择将集群角色绑定到用户或服务帐户。
  - 运行以下命令，将集群角色绑定到用户：

```
$ kubectl create rolebinding "${ROLE_BINDING_NAME}" --
clusterrole="token.kubevirt.io:generate" --user="${USER_NAME}"
```

  - 运行以下命令，将集群角色绑定到服务帐户：

```
$ kubectl create rolebinding "${ROLE_BINDING_NAME}" --
clusterrole="token.kubevirt.io:generate" --
serviceaccount="${SERVICE_ACCOUNT_NAME}"
```

7.3.2. 连接至串行控制台

您可以使用 OpenShift Container Platform web 控制台或 **virtctl** 命令行工具连接到虚拟机的串行控制台。

注意

目前不支持运行到单个虚拟机的并发 VNC 连接。

7.3.2.1. 使用 Web 控制台连接到串行控制台

您可以使用 OpenShift Container Platform web 控制台连接至虚拟机的串行控制台。

流程

1. 在 **Virtualization → VirtualMachines** 页面中，点虚拟机打开 **VirtualMachine 详情**页。

2. 点击 **Console** 选项卡。VNC 控制台会话会自动启动。
3. 点 **Disconnect** 结束 VNC 控制台会话。否则，VNC 控制台会话会在后台继续运行。
4. 从控制台列表中选择 **Serial console**。
5. 要结束控制台会话，请点控制台窗格外，然后点 **Disconnect**。

7.3.2.2. 使用 virtctl 连接到串行控制台

您可使用 **virtctl** 命令行工具连接到正在运行的虚拟机的串行控制台。

流程

1. 运行以下命令以启动控制台会话：

```
$ virtctl console <vm_name>
```

2. 按 **Ctrl+]** 结束控制台会话。

7.3.3. 连接到桌面视图

您可以使用 desktop viewer 和 Remote Desktop Protocol (RDP) 连接到 Windows 虚拟机。

7.3.3.1. 使用 Web 控制台连接到桌面查看器

您可以使用 OpenShift Container Platform Web 控制台连接到 Windows 虚拟机的桌面视图。

先决条件

- 您已在 Windows 虚拟机上安装了 QEMU 客户机代理。
- 已安装 RDP 客户端。

流程

1. 在 **Virtualization → VirtualMachines** 页面中，点虚拟机打开 **VirtualMachine 详情**页。
2. 点击 **Console** 选项卡。VNC 控制台会话会自动启动。
3. 点 **Disconnect** 结束 VNC 控制台会话。否则，VNC 控制台会话会在后台继续运行。
4. 从控制台列表中选择 **Desktop viewer**。
5. 点 **Create RDP Service** 打开 **RDP Service** 对话框。
6. 选择 **Expose RDP Service** 并点 **Save** 创建节点端口服务。
7. 点 **Launch Remote Desktop** 以下载 **.rdp** 文件并启动桌面查看器。

7.4. 指定实例类型或首选项

您可以指定实例类型、首选项或两者来定义一组工作负载大小和运行时特征，以便在多个虚拟机间重复使用。

7.4.1. 使用标志来指定实例类型和首选项

使用标志指定实例类型和首选项。

先决条件

- 集群中必须具有实例类型、首选或两者。

流程

1. 要在创建虚拟机时指定实例类型，请使用 the- **instancetype** 标志。要指定首选项，请使用 the- **preference** 标志。以下示例包括这两个标记：

```
$ virtctl create vm --instancetype <my_instancetype> --preference <my_preference>
```

2. 可选： 要指定命名空间的实例类型或首选项，请在传递给- **instancetype** or-- **preference** 标志命令的值中包含 **kind**。命名空间实例类型或首选项必须位于您要在其中创建虚拟机的同一命名空间中。以下示例包括命名空间实例类型和命名空间首选项的标记：

```
$ virtctl create vm --instancetype virtualmachineinstancetype/<my_instancetype> --
preference virtualmachinepreference/<my_preference>
```

7.4.2. 推断实例类型或首选项

推断实例类型、首选项或两者都默认启用，并且 **inferFromVolume** 属性的 **inferFromVolumeFailure** 策略被设置为 **Ignore**。当引导卷的推断时，会忽略错误，并使用实例类型创建虚拟机并取消设置虚拟机。

但是，当应用标记时，in **ferFromVolumeFailure** 策略默认为 **Reject**。当从引导卷推断出时，错误会导致创建该虚拟机的拒绝。

您可以使用-- **infer-instancetype** and **--infer-preference** 标志来推断哪个实例类型、首选项或两者用来定义虚拟机的工作负载大小和运行时特征。

先决条件

- 已安装 **virtctl** 工具。

流程

- 要从用于启动虚拟机的卷明确推断实例类型，请使用-- **infer-instancetype** 标志。要显式推断首选项，请使用-- **infer-preference** 标志。以下命令包括两个标记：

```
$ virtctl create vm --volume-import type:pvc,src:my-ns/my-pvc --infer-instancetype --infer-
preference
```

7.4.3. 设置 inferFromVolume 标签

在 PVC、数据源或数据卷中使用以下标签来指示在尝试从卷引导时使用哪些实例类型、首选或两者。

- 集群范围的实例类型： **instancetype.kubevirt.io/default-instancetype** 标签。
- namespaced 实例类型： **instancetype.kubevirt.io/default-instancetype-kind** 标签。如果留空，则默认为 **VirtualMachineClusterInstancetype** 标签。

- 集群范围的首选项： **instancetype.kubevirt.io/default-preference** 标签。
- namespaced preference: **instancetype.kubevirt.io/default-preference-kind** 标签。如果留空，则默认为 **VirtualMachineClusterPreference** 标签。

先决条件

- 集群中必须具有实例类型、首选或两者。

流程

- 要将标签应用到数据源，请使用 **oc label**。以下命令应用指向集群范围实例类型的标签：

```
$ oc label DataSource foo instancetype.kubevirt.io/default-instancetype=<my_instancetype>
```

7.5. 配置对虚拟机的 SSH 访问

您可以使用以下方法配置对虚拟机的 SSH 访问：

- **virtctl ssh 命令**
  您可以创建一个 SSH 密钥对，将公钥添加到虚拟机，并使用私钥运行 **virtctl ssh** 命令连接到虚拟机。

  您可以在运行时将公共 SSH 密钥添加到 Red Hat Enterprise Linux (RHEL) 9 虚拟机，或第一次引导到使用 cloud-init 数据源配置的客户机操作系统的虚拟机。
- **virtctl port-forward 命令**
  您可以将 **virtctl port-foward** 命令添加到 **.ssh/config** 文件中，并使用 OpenSSH 连接到虚拟机。
- 服务
  您可以创建一个服务，将服务与虚拟机关联，并连接到该服务公开的 IP 地址和端口。
- 二级网络
  您可以配置二级网络，将虚拟机(VM)附加到二级网络接口，并连接到 DHCP 分配的 IP 地址。

7.5.1. 访问配置注意事项

根据流量负载和客户端要求，配置对虚拟机(VM)的访问的每个方法都有优点和限制。

服务为从集群外部访问的应用程序提供出色的性能，并推荐使用。

如果内部集群网络无法处理流量负载，您可以配置二级网络。

virtctl ssh 和 virtctl port-forwarding 命令

- 易于配置。
- 建议对虚拟机进行故障排除。
- 推荐使用 Ansible 自动配置虚拟机的 **virtctl port-forwarding**。
- 动态公共 SSH 密钥可用于使用 Ansible 调配虚拟机。

- 因为 API 服务器的负担，不建议用于 Rsync 或 Remote Desktop Protocol 等高流量应用程序。
- API 服务器必须能够处理流量负载。
- 客户端必须能够访问 API 服务器。
- 客户端必须具有集群的访问凭证。

集群 IP 服务

- 内部集群网络必须能够处理流量负载。
- 客户端必须能够访问内部集群 IP 地址。

节点端口服务

- 内部集群网络必须能够处理流量负载。
- 客户端必须能够访问至少一个节点。

负载均衡器服务

- 必须配置负载均衡器。
- 每个节点必须能够处理一个或多个负载均衡器服务的流量负载。

二级网络

- 卓越的性能，因为流量不会通过内部集群网络。
- 允许灵活的网络拓扑方法。
- 客户机操作系统必须配置适当的安全性，因为虚拟机直接公开给二级网络。如果虚拟机被破坏，入侵者可能会获得对二级网络的访问权限。

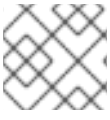
7.5.2. 使用 virtctl ssh

您可以将公共 SSH 密钥添加到虚拟机 (VM)，并通过运行 **virtctl ssh** 命令连接到虚拟机。

这个方法易于配置。但是，不建议在有高流量负载的环境中使用，因为它会对 API 服务器造成负担。

7.5.2.1. 关于静态和动态 SSH 密钥管理

您可以在首次引导时或在运行时动态向虚拟机 (VM) 静态添加公共 SSH 密钥。



注意

只有 Red Hat Enterprise Linux (RHEL) 9 支持动态密钥注入。

静态 SSH 密钥管理
您可以使用 cloud-init 数据源支持配置的客户机操作系统向虚拟机添加静态管理的 SSH 密钥。密钥会在第一次引导时添加到虚拟机 (VM) 中。

您可以使用以下方法之一添加密钥：

- 在使用 Web 控制台或命令行创建时，向单个虚拟机添加密钥。
- 使用 Web 控制台向项目添加密钥。之后，密钥会自动添加到您在这个项目中创建的虚拟机。

使用案例

- 作为虚拟机所有者，您可以使用单个密钥置备所有新创建的虚拟机。

动态 SSH 密钥管理
您可以为安装了 Red Hat Enterprise Linux (RHEL) 9 的虚拟机启用动态 SSH 密钥管理。之后，您可以在运行时更新密钥。密钥由 QEMU 客户机代理添加，该代理使用 Red Hat 引导源安装。

出于安全原因，您可以禁用动态密钥管理。然后，虚拟机会继承创建它的镜像的密钥管理设置。

使用案例

- 授予或撤销对虚拟机的访问权限：作为集群管理员，您可以通过从应用到命名空间中的所有虚拟机的 **Secret** 对象添加或删除单个用户的密钥来授予或撤销远程虚拟机访问。
- 用户访问：您可以将访问凭证添加到您创建和管理的所有虚拟机。
- Ansible 置备：
  - 作为操作团队成员，您可以创建一个单一 secret，其中包含用于 Ansible 置备的所有密钥。
  - 作为虚拟机所有者，您可以创建虚拟机并附加用于 Ansible 置备的密钥。
- 密钥轮转：
  - 作为集群管理员，您可以轮转命名空间中虚拟机使用的 Ansible 置备程序密钥。
  - 作为工作负载所有者，您可以轮转您管理的虚拟机的密钥。

7.5.2.2. 静态密钥管理

当使用 OpenShift Container Platform web 控制台或命令行创建虚拟机 (VM) 时，您可以添加静态管理的公共 SSH 密钥。当虚拟机第一次引导时，密钥会添加为 cloud-init 数据源。

在使用 Web 控制台创建虚拟机时，您还可以将公共 SSH 密钥添加到项目中。密钥保存为 secret，并自动添加到您创建的所有虚拟机中。

注意

如果您在项目中添加 secret，然后删除虚拟机，则 secret 会被保留，因为它是一个命名空间资源。您必须手动删除 secret。

7.5.2.2.1. 从模板创建虚拟机时添加密钥

在使用 OpenShift Container Platform web 控制台创建虚拟机时，您可以添加静态管理的公共 SSH 密钥。密钥会在第一次引导时作为 cloud-init 数据源添加到虚拟机。这个方法不会影响 cloud-init 用户数据。

可选：您可以在项目中添加密钥。之后，此密钥会自动添加到您在项目中创建的虚拟机。

先决条件

- 您可以通过运行 **ssh-keygen** 命令生成 SSH 密钥对。

流程

1. 在 web 控制台中进入到 **Virtualization → Catalog**。
2. 点模板标题。
   客户机操作系统必须支持 cloud-init 数据源的配置。
3. 点 **Customize VirtualMachine**。
4. 点击 **Next**。
5. 点 **Scripts** 选项卡。
6. 如果您还没有在项目中添加公共 SSH 密钥，点 **Authorized SSH key** 旁边的编辑图标，然后选择以下选项之一：
   - **使用现有** ：从 secrets 列表中选择一个 secret。
   - **添加新** ：
     a. 浏览到 SSH 密钥文件或在 key 字段中粘贴文件。
     b. 输入 secret 名称。
     c. 可选： 选择 **Automatically apply this key to any new VirtualMachine you create in this project**。
7. 点击 **Save**。
8. 点 **Create VirtualMachine**。
   VirtualMachine 详情页显示创建虚拟机的进度。

验证

- 点 **Configuration** 选项卡上的 **Scripts** 选项卡。
  secret 名称显示在 **Authorized SSH key** 部分中。

7.5.2.2.2. 使用 Web 控制台从实例类型创建虚拟机时添加密钥

您可以使用 OpenShift Container Platform web 控制台从实例类型创建虚拟机 (VM)。您还可以通过复制现有快照或克隆虚拟机，来使用 Web 控制台创建虚拟机。

您可以从可用可引导卷列表创建虚拟机。您可以在列表中添加基于 Linux 或 Windows 的卷。

在使用 OpenShift Container Platform web 控制台从实例类型创建虚拟机(VM)时，您可以添加静态管理的 SSH 密钥。密钥会在第一次引导时作为 cloud-init 数据源添加到虚拟机。这个方法不会影响 cloud-init 用户数据。

流程

1. 在 Web 控制台中，进入到 **Virtualization → Catalog**。
   InstanceTypes 选项卡默认为打开。

2. 选择以下选项之一：

   - 从列表中选择合适的可引导卷。如果列表已被截断，请点 **Show all** 按钮来显示整个列表。

     注意

     可引导的卷表仅列出 **openshift-virtualization-os-images** 命名空间中具有 **instancetype.kubevirt.io/default-preference** 标签的卷。

     - 可选：点星号图标将可引导卷指定为热门卷。不足的可引导卷首先出现在卷列表中。

   - 点 **Add volume** 上传新卷，或使用现有的持久性卷声明(PVC)、卷快照或 **containerDisk** 卷。点击 **Save**。
     集群中不可用的操作系统的徽标显示在列表的底部。您可以点 **Add volume** 链接为所需的操作系统添加卷。

     另外，还有 **创建 Windows 引导源**快速启动的链接。如果您将鼠标悬停在 *Select volume to boot from* 行旁边的问号图标上，则同一链接会出现在弹出窗口中。

     安装环境或环境断开连接后，从中引导的卷列表为空。在这种情况下，会显示三个操作系统徽标：Windows、RHEL 和 Linux。您可以点 **Add volume** 按钮添加新卷来满足您的要求。

3. 点实例类型标题，然后选择适合您的工作负载的资源大小。

4. 可选：选择虚拟机详情，包括虚拟机的名称，适用于您要从其引导的卷：

   - 对于基于 Linux 的卷，请按照以下步骤配置 SSH：

     a. 如果您还没有在项目中添加公共 SSH 密钥，点 **VirtualMachine details** 部分中的 **Authorized SSH key** 旁边的编辑图标。

     b. 选择以下选项之一：

        - **使用现有** ：从 secrets 列表中选择一个 secret。
        - **Add new**: 遵循以下步骤：
          i. 浏览到公共 SSH 密钥文件，或在 key 字段中粘贴文件。
          ii. 输入 secret 名称。
          iii. 可选： 选择 **Automatically apply this key to any new VirtualMachine you create in this project**。

     c. 点击 **Save**。

   - 对于 Windows 卷，请按照以下步骤配置 sysprep 选项：

     - 如果您还没有为 Windows 卷添加 sysprep 选项，请按照以下步骤执行：
       i. 点 **VirtualMachine 详情** 部分中的 **Sysprep** 的编辑图标。
       ii. 添加 **Autoattend.xml** 回答文件。
       iii. 添加 **Unattend.xml** 回答文件。
       iv. 点击 **Save**。

- 如果要将现有的 sysprep 选项用于 Windows 卷，请按照以下步骤执行：
  i. 点 **Attach existing sysprep**。
  ii. 输入现有 sysprep **Unattend.xml** 回答文件的名称。
  iii. 点击 **Save**。

5. 可选： 如果要创建 Windows 虚拟机，您可以挂载 Windows 驱动程序磁盘：
   a. 点 **Customize VirtualMachine** 按钮。
   b. 在 **VirtualMachine 详情**页中，点 **Storage**。
   c. 选择 **Mount Windows 驱动程序磁盘**复选框。
6. 可选：点 **View YAML & CLI** 查看 YAML 文件。点 **CLI** 查看 CLI 命令。您还可以下载或复制 YAML 文件内容或 CLI 命令。
7. 点 **Create VirtualMachine**。

创建虚拟机后，您可以在 **VirtualMachine 详情**页中监控状态。

7.5.2.2.3. 使用命令行在创建虚拟机时添加密钥

当使用命令行创建虚拟机(VM)时，您可以添加静态管理的公共 SSH 密钥。密钥会在第一次引导时添加到虚拟机。

密钥作为 cloud-init 数据源添加到虚拟机中。此方法将访问凭据与 cloud-init 用户数据中的应用数据分隔开。这个方法不会影响 cloud-init 用户数据。

先决条件

- 您可以通过运行 **ssh-keygen** 命令生成 SSH 密钥对。

流程

1. 为 **VirtualMachine** 对象和 **Secret** 对象创建清单文件：

   清单示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  dataVolumeTemplates:
    - metadata:
        name: example-vm-volume
      spec:
        sourceRef:
          kind: DataSource
          name: rhel9
          namespace: openshift-virtualization-os-images
        storage:
```

```
        resources: {}
  instancetype:
    name: u1.medium
  preference:
    name: rhel.9
  running: true
  template:
    spec:
      domain:
        devices: {}
      volumes:
        - dataVolume:
            name: example-vm-volume
          name: rootdisk
        - cloudInitNoCloud: 1
            userData: |-
              #cloud-config
              user: cloud-user
          name: cloudinitdisk
      accessCredentials:
        - sshPublicKey:
            propagationMethod:
              noCloud: {}
            source:
              secret:
                secretName: authorized-keys 2
---
apiVersion: v1
kind: Secret
metadata:
  name: authorized-keys
data:
  key: c3NoLXJzYSB... 3
```

1. 指定 **cloudInitNoCloud** 数据源。
2. 指定 **Secret** 对象名称。
3. 粘贴公共 SSH 密钥。

2. 运行以下命令来创建 **VirtualMachine** 和 **Secret** 对象：

```
$ oc create -f <manifest_file>.yaml
```

3. 运行以下命令来启动虚拟机：

```
$ virtctl start vm example-vm -n example-namespace
```

验证

- 获取虚拟机配置：

```
$ oc describe vm example-vm -n example-namespace
```

输出示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  template:
    spec:
      accessCredentials:
        - sshPublicKey:
            propagationMethod:
              noCloud: {}
            source:
              secret:
                secretName: authorized-keys
# ...
```

7.5.2.3. 动态密钥管理

您可以使用 OpenShift Container Platform Web 控制台或命令行为虚拟机(VM)启用动态密钥注入。然后，您可以在运行时更新密钥。

注意

只有 Red Hat Enterprise Linux (RHEL) 9 支持动态密钥注入。

如果您禁用动态密钥注入，则虚拟机会继承创建它的镜像的密钥管理方法。

7.5.2.3.1. 从模板创建虚拟机时启用动态密钥注入

在使用 OpenShift Container Platform web 控制台从模板创建虚拟机时，您可以启用动态公共 SSH 密钥注入。然后，您可以在运行时更新密钥。

注意

只有 Red Hat Enterprise Linux (RHEL) 9 支持动态密钥注入。

密钥由 QEMU 客户机代理添加到虚拟机，该代理使用 RHEL 9 安装。

先决条件

- 您可以通过运行 **ssh-keygen** 命令生成 SSH 密钥对。

流程

1. 在 web 控制台中进入到 **Virtualization → Catalog**。
2. 点 **Red Hat Enterprise Linux 9 虚拟机**标题。
3. 点 **Customize VirtualMachine**。

4. 点击 **Next**。
5. 点 **Scripts** 选项卡。
6. 如果您还没有在项目中添加公共 SSH 密钥，点 **Authorized SSH key** 旁边的编辑图标，然后选择以下选项之一：
   - **使用现有** ：从 secrets 列表中选择一个 secret。
   - **添加新** ：
     a. 浏览到 SSH 密钥文件或在 key 字段中粘贴文件。
     b. 输入 secret 名称。
     c. 可选： 选择 **Automatically apply this key to any new VirtualMachine you create in this project**。
7. 将 **Dynamic SSH 密钥注入** 设置为 on。
8. 点击 **Save**。
9. 点 **Create VirtualMachine**。
   VirtualMachine 详情页显示创建虚拟机的进度。

验证

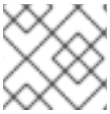
- 点 **Configuration** 选项卡上的 **Scripts** 选项卡。
  secret 名称显示在 **Authorized SSH key** 部分中。

7.5.2.3.2. 使用 Web 控制台在从实例类型创建虚拟机时启用动态密钥注入

您可以使用 OpenShift Container Platform web 控制台从实例类型创建虚拟机 (VM)。您还可以通过复制现有快照或克隆虚拟机，来使用 Web 控制台创建虚拟机。

您可以从可用可引导卷列表创建虚拟机。您可以在列表中添加基于 Linux 或 Windows 的卷。

在使用 OpenShift Container Platform web 控制台从实例类型创建虚拟机(VM)时，您可以启用动态 SSH 密钥注入。然后，您可以在运行时添加或撤销密钥。



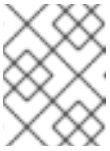
注意

只有 Red Hat Enterprise Linux (RHEL) 9 支持动态密钥注入。

密钥由 QEMU 客户机代理添加到虚拟机，该代理使用 RHEL 9 安装。

流程

1. 在 Web 控制台中，进入到 **Virtualization → Catalog**。
   InstanceTypes 选项卡默认为打开。
2. 选择以下选项之一：
   - 从列表中选择合适的可引导卷。如果列表已被截断，请点 **Show all** 按钮来显示整个列表。



注意

可引导的卷表仅列出 **openshift-virtualization-os-images** 命名空间中具有 **instancetype.kubevirt.io/default-preference** 标签的卷。

- 可选：点星号图标将可引导卷指定为热门卷。不足的可引导卷首先出现在卷列表中。

- 点 **Add volume** 上传新卷，或使用现有的持久性卷声明(PVC)、卷快照或 **containerDisk** 卷。点击 **Save**。
  集群中不可用的操作系统的徽标显示在列表的底部。您可以点 **Add volume** 链接为所需的操作系统添加卷。

  另外，还有 **创建 Windows 引导源**快速启动的链接。如果您将鼠标悬停在 *Select volume to boot from* 行旁边的问号图标上，则同一链接会出现在弹出窗口中。

  安装环境或环境断开连接后，从中引导的卷列表为空。在这种情况下，会显示三个操作系统徽标：Windows、RHEL 和 Linux。您可以点 **Add volume** 按钮添加新卷来满足您的要求。

3. 点实例类型标题，然后选择适合您的工作负载的资源大小。

4. 点 **Red Hat Enterprise Linux 9 虚拟机**标题。

5. 可选：选择虚拟机详情，包括虚拟机的名称，适用于您要从其引导的卷：

   - 对于基于 Linux 的卷，请按照以下步骤配置 SSH：

     a. 如果您还没有在项目中添加公共 SSH 密钥，点 **VirtualMachine details** 部分中的 **Authorized SSH key** 旁边的编辑图标。

     b. 选择以下选项之一：

        - **使用现有** ：从 secrets 列表中选择一个 secret。
        - **Add new**: 遵循以下步骤：
          i. 浏览到公共 SSH 密钥文件，或在 key 字段中粘贴文件。
          ii. 输入 secret 名称。
          iii. 可选： 选择 **Automatically apply this key to any new VirtualMachine you create in this project**。

     c. 点击 **Save**。

   - 对于 Windows 卷，请按照以下步骤配置 sysprep 选项：
     - 如果您还没有为 Windows 卷添加 sysprep 选项，请按照以下步骤执行：
       i. 点 **VirtualMachine 详情** 部分中的 **Sysprep** 的编辑图标。
       ii. 添加 **Autoattend.xml** 回答文件。
       iii. 添加 **Unattend.xml** 回答文件。
       iv. 点击 **Save**。
     - 如果要将现有的 sysprep 选项用于 Windows 卷，请按照以下步骤执行：

i. 点 **Attach existing sysprep**。

ii. 输入现有 sysprep **Unattend.xml** 回答文件的名称。

iii. 点击 **Save**。

6. 在 **VirtualMachine 详情** 部分中将 **Dynamic SSH 密钥注入** 设置为 on。
7. 可选： 如果要创建 Windows 虚拟机，您可以挂载 Windows 驱动程序磁盘：
   a. 点 **Customize VirtualMachine** 按钮。
   b. 在 **VirtualMachine 详情**页中，点 **Storage**。
   c. 选择 **Mount Windows 驱动程序磁盘**复选框。
8. 可选：点 **View YAML & CLI** 查看 YAML 文件。点 **CLI** 查看 CLI 命令。您还可以下载或复制 YAML 文件内容或 CLI 命令。
9. 点 **Create VirtualMachine**。

创建虚拟机后，您可以在 **VirtualMachine 详情**页中监控状态。

7.5.2.3.3. 使用 Web 控制台启用动态 SSH 密钥注入

您可以使用 OpenShift Container Platform Web 控制台为虚拟机 (VM) 启用动态密钥注入。然后，您可以在运行时更新公共 SSH 密钥。

该密钥由 QEMU 客户机代理添加到虚拟机，该代理与 Red Hat Enterprise Linux (RHEL) 9 一起安装。

先决条件

- 客户机操作系统是 RHEL 9。

流程

1. 在 web 控制台中进入到 **Virtualization → VirtualMachines**。
2. 选择一个虚拟机以打开 **VirtualMachine 详情**页。
3. 在 **Configuration** 选项卡上，点 **Scripts**。
4. 如果您还没有在项目中添加公共 SSH 密钥，点 **Authorized SSH key** 旁边的编辑图标，然后选择以下选项之一：
   - **使用现有** ：从 secrets 列表中选择一个 secret。
   - **添加新** ：
     a. 浏览到 SSH 密钥文件或在 key 字段中粘贴文件。
     b. 输入 secret 名称。
     c. 可选： 选择 **Automatically apply this key to any new VirtualMachine you create in this project**。
5. 将 **Dynamic SSH 密钥注入** 设置为 on。

6. 点击 **Save**。

7.5.2.3.4. 使用命令行启用动态密钥注入

您可以使用命令行为虚拟机启用动态密钥注入。然后，您可以在运行时更新公共 SSH 密钥。

注意

只有 Red Hat Enterprise Linux (RHEL) 9 支持动态密钥注入。

密钥由 QEMU 客户机代理添加到虚拟机，该代理使用 RHEL 9 自动安装安装。

先决条件

- 您可以通过运行 **ssh-keygen** 命令生成 SSH 密钥对。

流程

1. 为 **VirtualMachine** 对象和 **Secret** 对象创建清单文件：

清单示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  dataVolumeTemplates:
    - metadata:
        name: example-vm-volume
      spec:
        sourceRef:
          kind: DataSource
          name: rhel9
          namespace: openshift-virtualization-os-images
        storage:
          resources: {}
  instancetype:
    name: u1.medium
  preference:
    name: rhel.9
  running: true
  template:
    spec:
      domain:
        devices: {}
      volumes:
        - dataVolume:
            name: example-vm-volume
          name: rootdisk
        - cloudInitNoCloud: (1)
            userData: |-
              #cloud-config
```

```
            runcmd:
            - [ setsebool, -P, virt_qemu_ga_manage_ssh, on ]
          name: cloudinitdisk
      accessCredentials:
        - sshPublicKey:
            propagationMethod:
              qemuGuestAgent:
                users: ["cloud-user"]
            source:
              secret:
                secretName: authorized-keys (2)
---
apiVersion: v1
kind: Secret
metadata:
  name: authorized-keys
data:
  key: c3NoLXJzYSB... (3)
```

(1) 指定 **cloudInitNoCloud** 数据源。

(2) 指定 **Secret** 对象名称。

(3) 粘贴公共 SSH 密钥。

2. 运行以下命令来创建 **VirtualMachine** 和 **Secret** 对象：

```
$ oc create -f <manifest_file>.yaml
```

3. 运行以下命令来启动虚拟机：

```
$ virtctl start vm example-vm -n example-namespace
```

验证

- 获取虚拟机配置：

```
$ oc describe vm example-vm -n example-namespace
```

输出示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  template:
    spec:
      accessCredentials:
        - sshPublicKey:
            propagationMethod:
              qemuGuestAgent:
```

```
            users: ["cloud-user"]
        source:
          secret:
            secretName: authorized-keys
# ...
```

7.5.2.4. 使用 virtctl ssh 命令

您可以使用 **virtcl ssh** 命令访问正在运行的虚拟机(VM)。

先决条件

- 已安装 **virtctl** 命令行工具。
- 您已向虚拟机添加了一个公共 SSH 密钥。
- 已安装 SSH 客户端。
- 安装 **virtctl** 工具的环境具有访问虚拟机所需的集群权限。例如，运行 **oc login** 或设置了 **KUBECONFIG** 环境变量。

流程

- 运行 **virtctl ssh** 命令：

```
$ virtctl -n <namespace> ssh <username>@example-vm -i <ssh_key> 1
```

1 指定命名空间、用户名和 SSH 私钥。默认的 SSH 密钥位置为 **/home/user/.ssh**。如果密钥保存在不同的位置，您必须指定路径。

Example

```
$ virtctl -n my-namespace ssh cloud-user@example-vm -i my-key
```

提示

您可以在 web 控制台中复制 **virtctl ssh** 命令，方法是从 **VirtualMachines** 页中的虚拟机旁的选项菜单中选择 **Copy SSH 命令**。

7.5.3. 使用 virtctl port-forward 命令

您可以使用本地 OpenSSH 客户端和 **virtctl port-forward** 命令连接到正在运行的虚拟机 (VM)。您可以将此方法与 Ansible 配合使用，以自动配置虚拟机。

对于低流量应用程序，建议使用这个方法，因为端口转发流量通过 control plane 发送。对于 Rsync 或 Remote Desktop 协议等高流量应用程序（如 Rsync 或 Remote Desktop 协议）使用这个方法，因为它对 API 服务器造成大量负担。

先决条件

- 已安装 **virtctl** 客户端。

- 您要访问的虚拟机正在运行。
- 安装 **virtctl** 工具的环境具有访问虚拟机所需的集群权限。例如，运行 **oc login** 或设置了 **KUBECONFIG** 环境变量。

流程

1. 在客户端机器上的 **~/.ssh/config** 文件中添加以下文本：

```
Host vm/*
  ProxyCommand virtctl port-forward --stdio=true %h %p
```

2. 运行以下命令来连接到虚拟机：

```
$ ssh <user>@vm/<vm_name>.<namespace>
```

7.5.4. 使用服务进行 SSH 访问

您可以为虚拟机(VM)创建服务，并连接到该服务公开的 IP 地址和端口。

服务为从集群外部或集群外部访问的应用程序提供出色的性能，并推荐使用。入口流量受防火墙保护。

如果集群网络无法处理流量负载，请考虑使用二级网络进行虚拟机访问。

7.5.4.1. 关于服务

Kubernetes 服务将客户端的网络访问权限公开给一组容器集上运行的应用。服务在 **NodePort** 和 **LoadBalancer** 类型方面提供抽象、负载均衡以及暴露于外部世界。

ClusterIP

在内部 IP 地址上公开服务，并将 DNS 名称公开给集群中的其他应用程序。单个服务可映射到多个虚拟机。当客户端尝试连接到服务时，客户端请求会在可用后端之间平衡负载。**ClusterIP** 是默认的服务类型。

NodePort

在集群中每个所选节点的同一端口上公开该服务。**NodePort** 使端口可从集群外部访问，只要节点本身可以被客户端外部访问。

LoadBalancer

在当前云中创建外部负载均衡器（如果支持），并为该服务分配固定的外部 IP 地址。

注意

对于内部集群，您可以通过部署 MetalLB Operator 来配置负载均衡服务。

7.5.4.2. 创建服务

您可以使用 OpenShift Container Platform web 控制台、**virtctl** 命令行工具或 YAML 文件创建服务来公开虚拟机(VM)。

7.5.4.2.1. 使用 Web 控制台启用负载均衡器服务创建

您可以使用 OpenShift Container Platform web 控制台为虚拟机(VM)创建负载均衡器服务。

先决条件

- 已为集群配置负载均衡器。
- 以具有 **cluster-admin** 角色的用户身份登录。

流程

1. 进入到 **Virtualization → Overview**。
2. 在 **Settings** 选项卡中，点 **Cluster**。
3. 展开 **General settings** 和 **SSH 配置**。
4. 将 **SSH over LoadBalancer 服务**设置为 on。

7.5.4.2.2. 使用 Web 控制台创建服务

您可以使用 OpenShift Container Platform web 控制台为虚拟机(VM)创建节点端口或负载均衡器服务。

先决条件

- 已将集群网络配置为支持负载均衡器或节点端口。
- 要创建负载均衡器服务，您需要已启用了创建负载均衡器服务。

流程

1. 进入 **VirtualMachines** 并选择虚拟机来查看 **VirtualMachine 详情**页。
2. 在 **Details** 选项卡中，从 **SSH service type** 列表中选择 **SSH over LoadBalancer**。
3. 可选：点复制图标将 **SSH** 命令复制到您的剪贴板。

验证

- 检查 **Details** 标签页中的 **Services** 窗格，以查看新服务。

7.5.4.2.3. 使用 virtctl 创建服务

您可以使用 **virtctl** 命令行工具为虚拟机 (VM) 创建服务。

先决条件

- 已安装 **virtctl** 命令行工具。
- 您已将集群网络配置为支持该服务。
- 安装 **virtctl** 的环境具有访问虚拟机所需的集群权限。例如，运行 **oc login** 或设置了 **KUBECONFIG** 环境变量。

流程

- 运行以下命令来创建服务：

```
$ virtctl expose vm <vm_name> --name <service_name> --type <service_type> --port <port>
```
1

1 指定 **ClusterIP**、**NodePort** 或 **LoadBalancer** 服务类型。

Example

```
$ virtctl expose vm example-vm --name example-service --type NodePort --port 22
```

验证

- 运行以下命令验证服务：

```
$ oc get service
```

后续步骤

使用 **virtctl** 创建服务后，您必须将 **special: key** 添加到 **VirtualMachine** 清单的 **spec.template.metadata.labels** 小节中。请参阅使用命令行创建服务。

7.5.4.2.4. 使用命令行创建服务

您可以使用命令行创建服务并将其与虚拟机 (VM) 关联。

先决条件

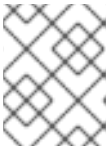
- 您已将集群网络配置为支持该服务。

流程

1. 编辑 **VirtualMachine** 清单，为创建服务添加标签：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  running: false
  template:
    metadata:
      labels:
        special: key
# ...
```
1

1 在 **spec.template.metadata.labels** 小节中添加 **special: key**。



注意

虚拟机上的标签会传递到 pod。**special: key** 标签必须与 **Service** 清单的 **spec.selector** 属性中的标签匹配。

2. 保存 **VirtualMachine** 清单文件以应用更改。

3. 创建 **Service** 清单以公开虚拟机：

```
apiVersion: v1
kind: Service
metadata:
  name: example-service
  namespace: example-namespace
spec:
# ...
  selector:
    special: key (1)
  type: NodePort (2)
  ports: (3)
    protocol: TCP
    port: 80
    targetPort: 9376
    nodePort: 30000
```

(1) 指定添加到 **VirtualMachine** 清单的 **spec.template.metadata.labels** 小节中的标签。

(2) 指定 **ClusterIP**、**NodePort** 或 **LoadBalancer**。

(3) 指定您要从虚拟机公开的网络端口和协议集合。

4. 保存 **Service** 清单文件。

5. 运行以下命令来创建服务：

```
$ oc create -f example-service.yaml
```

6. 重启虚拟机以应用更改。

验证

- 查询 **Service** 对象以验证它是否可用：

```
$ oc get service -n example-namespace
```

7.5.4.3. 使用 SSH 连接到服务公开的虚拟机

您可以使用 SSH 连接到服务公开的虚拟机 (VM)。

先决条件

- 您创建了服务来公开虚拟机。
- 已安装 SSH 客户端。
- 已登陆到集群。

流程

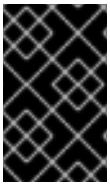
- 运行以下命令来访问虚拟机：

```
$ ssh <user_name>@<ip_address> -p <port>
```
(1)

(1) 指定集群 IP 服务的集群 IP、节点端口服务的节点 IP 或负载均衡器服务的外部 IP 地址。

7.5.5. 使用二级网络进行 SSH 访问

您可以配置二级网络，将虚拟机 (VM) 附加到二级网络接口，并使用 SSH 连接到 DHCP 分配的 IP 地址。



重要

辅助网络提供卓越的性能，因为流量不是由集群网络堆栈处理。但是，虚拟机直接公开给二级网络，不受防火墙保护。如果虚拟机被破坏，入侵者可能会获得对二级网络的访问权限。如果使用此方法，您必须在虚拟机操作系统中配置适当的安全性。

有关网络选项的更多信息，请参阅 OpenShift Virtualization 调优和扩展指南 中的 Multus 和 SR-IOV 文档。

先决条件

- 已配置了一个二级网络，如 Linux 网桥 或 SR-IOV。
- 为 Linux 网桥网络 或 SR-IOV Network Operator 创建网络附加定义，在创建 **SriovNetwork** 对象时创建了网络附加定义。

7.5.5.1. 使用 Web 控制台配置虚拟机网络接口

您可以使用 OpenShift Container Platform Web 控制台为虚拟机配置网络接口。

先决条件

- 为网络创建了网络附加定义。

流程

1. 进入到 **Virtualization → VirtualMachines**。
2. 点虚拟机查看 **VirtualMachine 详情**页。
3. 在 **Configuration** 选项卡上，点 **Network interfaces** 选项卡。
4. 点 **Add network interface**。
5. 输入接口名称，然后从 **Network** 列表中选择网络附加定义。
6. 点击 **Save**。
7. 重启虚拟机以应用更改。

7.5.5.2. 使用 SSH 连接到附加到二级网络的虚拟机

您可以使用 SSH 连接到二级网络的虚拟机 (VM)。

先决条件

- 将虚拟机附加到使用 DHCP 服务器的二级网络。
- 已安装 SSH 客户端。

流程

1. 运行以下命令来获取虚拟机的 IP 地址：

```
$ oc describe vm <vm_name> -n <namespace>
```

输出示例

```
# ...
Interfaces:
  Interface Name:  eth0
  Ip Address:      10.244.0.37/24
  Ip Addresses:
    10.244.0.37/24
    fe80::858:aff:fef4:25/64
  Mac:             0a:58:0a:f4:00:25
  Name:            default
# ...
```

2. 运行以下命令来连接到虚拟机：

```
$ ssh <user_name>@<ip_address> -i <ssh_key>
```

Example

```
$ ssh cloud-user@10.244.0.37 -i ~/.ssh/id_rsa_cloud-user
```

注意

您还可以使用 集群 FQDN 访问附加到二级网络接口的虚拟机。

7.6. 编辑虚拟机

您可以使用 OpenShift Container Platform web 控制台更新虚拟机(VM)配置。您可以更新 YAML 文件或 **VirtualMachine 详情**页。

您还可以使用命令行编辑虚拟机。

要编辑虚拟机以使用虚拟磁盘或 LUN 配置磁盘共享，请参阅为虚拟机配置共享卷。

7.6.1. 使用命令行编辑虚拟机

您可以使用命令行编辑虚拟机 (VM)。

先决条件

- 已安装 **oc** CLI。

流程

1. 运行以下命令来获取虚拟机配置：

```
$ oc edit vm <vm_name>
```

2. 编辑 YAML 配置。
3. 如果要编辑正在运行的虚拟机，您需要执行以下任一操作：
   - 重启虚拟机。
   - 运行以下命令使新配置生效：

```
$ oc apply vm <vm_name> -n <namespace>
```

7.6.2. 将磁盘添加到虚拟机

您可以使用 OpenShift Container Platform web 控制台将虚拟磁盘添加到虚拟机 (VM)。

流程

1. 在 web 控制台中进入到 **Virtualization → VirtualMachines**。
2. 选择一个虚拟机以打开 **VirtualMachine 详情**页。
3. 在 **Disks** 选项卡上，点 **Add disk**。
4. 指定 **Source**、**Name**、**Size**、**Type**、**Interface** 和 **Storage Class**。
   a. 可选：如果您使用空磁盘源并在创建数据卷时要求最大写入性能，则可以启用预分配。如果要这样做，可选中**启用预分配**复选框。
   b. 可选：您可以清除 **Apply optimized StorageProfile 设置**，以更改虚拟磁盘的**卷模式**和访问**模式**。如果没有指定这些参数，系统将使用 **kubevirt-storage-class-defaults** 配置映射中的默认值。
5. 点击 **Add**。

注意

如果虚拟机正在运行，您必须重启虚拟机以应用更改。

7.6.2.1. 存储字段

字段	描述
空白（创建 PVC）	创建一个空磁盘。
通过 URL 导入（创建 PVC）	通过 URL（HTTP 或 HTTPS 端点）导入内容。
使用现有的 PVC	使用集群中已可用的 PVC。

字段	描述
克隆现有的 PVC（创建 PVC）	选择集群中可用的现有 PVC 并克隆它。
通过 Registry 导入（创建 PVC）	通过容器 registry 导入内容。
容器（临时）	从集群可以访问的 registry 中的容器上传内容。容器磁盘应只用于只读文件系统，如 CD-ROM 或临时虚拟机。
名称	磁盘的名称。名称可包含小写字母 (**a-z**)、数字 (**0-9**)、连字符 (**-**) 和句点 (**.**)，最多 253 个字符。第一个和最后一个字符必须为字母数字。名称不得包含大写字母、空格或特殊字符。
Size	GiB 中磁盘的大小。
类型	磁盘类型。示例：磁盘或光盘
Interface	磁盘设备的类型。支持的接口包括 **virtIO**、**SATA** 和 **SCSI**。
Storage class	用于创建磁盘的存储类。

高级存储设置
以下高级存储设置是可选的，对 **Blank**, **Import via URL**, and **Clone existing PVC** 磁盘可用。

如果没有指定这些参数，系统将使用默认存储配置集值。

参数	选项	参数描述
卷模式	Filesystem	在基于文件系统的卷中保存虚拟磁盘。
	Block	直接将虚拟磁盘存储在块卷中。只有底层存储支持时才使用 **Block**。
访问模式	ReadWriteOnce (RWO)	卷可以被一个节点以读写模式挂载。
	ReadWriteMany (RWX)	卷可以被多个节点以读写模式挂载。 **注意** 实时迁移需要此模式。

7.6.3. 在虚拟机上挂载 Windows 驱动程序磁盘

您可以使用 OpenShift Container Platform Web 控制台将 Windows 驱动程序磁盘挂载到虚拟机 (VM) 上。

流程

1. 进入到 **Virtualization → VirtualMachines**。
2. 选择所需的虚拟机以打开 **VirtualMachine 详情**页。
3. 在 **Configuration** 选项卡中，点 **Storage**。
4. 选择 **Mount Windows 驱动程序磁盘**复选框。
   Windows 驱动程序磁盘显示在挂载的磁盘列表中。

7.6.4. 将 secret、配置映射或服务帐户添加到虚拟机

使用 OpenShift Container Platform Web 控制台向虚拟机添加 secret、配置映射或服务帐户。

这些资源作为磁盘添加到虚拟机中。您可在挂载任何其他磁盘时挂载 secret、配置映射或服务帐户。

如果虚拟机正在运行，则更改在重启虚拟机之后才会生效。新添加的资源在页面的顶部被标记为待处理更改。

先决条件

- 要添加的 secret、配置映射或服务帐户必须与目标虚拟机位于同一命名空间中。

流程

1. 在侧边菜单中点 **Virtualization → VirtualMachines**。
2. 选择虚拟机以打开 **VirtualMachine 详情**页面。
3. 点 **Configuration → Environment**。
4. 点 **Add Config Map、Secret 或 Service Account**。
5. 点 **Select a resource**，从列表中选择一个资源。为所选资源自动生成带有六个字符的序列号。
6. 可选：点 **Reload** 将环境恢复到其上次保存的状态。
7. 点击 **Save**。

验证

1. 在 **VirtualMachine 详情**页面中，点 **Configuration → Disks** 并验证资源是否在磁盘列表中显示。
2. 点 **Actions → Restart** 重启虚拟机。

现在，您可以在挂载任何其他磁盘时挂载 secret、配置映射或服务帐户。

配置映射、secret 和服务帐户的其他资源

- 了解配置映射
- 为 pod 提供敏感数据
- 了解并创建服务帐户

7.7. 编辑引导顺序

您可以使用 Web 控制台或 CLI 更新引导顺序列表的值。

通过 **Virtual Machine Overview** 页面中的 **Boot Order**，您可以：

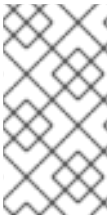
- 选择磁盘或网络接口控制器 (NIC) 并将其添加到引导顺序列表中。
- 编辑引导顺序列表中磁盘或 NIC 的顺序。
- 从引导顺序列表中移除磁盘或者 NIC，然后将其返回到可引导源清单。

7.7.1. 向 web 控制台的引导顺序列表中添加项目

使用 web 控制台将项目添加到引导顺序列表中。

流程

1. 在侧边菜单中点 **Virtualization → VirtualMachines**。
2. 选择虚拟机以打开 **VirtualMachine 详情**页面。
3. 点 **Details** 标签页。
4. 点击位于 **Boot Order** 右侧的铅笔图标。如果 YAML 配置不存在，或者是首次创建引导顺序列表时，会显示以下消息: **No resource selected.虚拟机会根据在 YAML 文件中的顺序从磁盘引导。**
5. 点 **Add Source**，为虚拟机选择一个可引导磁盘或网络接口控制器 (NIC)。
6. 在引导顺序列表中添加附加磁盘或者 NIC。
7. 点 **Save**。



注意

如果虚拟机正在运行，在重启虚拟机后对 **Boot Order** 的更改不会生效。

您可以点 **Boot Order** 字段右侧的 **View Pending Changes** 查看待处理的修改。页面顶部的 **Pending Changes** 标题显示虚拟机重启时将应用的所有更改列表。

7.7.2. 在 web 控制台中编辑引导顺序列表

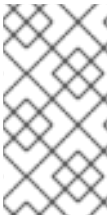
在 web 控制台中编辑引导顺序列表。

流程

1. 在侧边菜单中点 **Virtualization → VirtualMachines**。
2. 选择虚拟机以打开 **VirtualMachine 详情**页面。
3. 点 **Details** 标签页。
4. 点击位于 **Boot Order** 右侧的铅笔图标。
5. 选择适当的方法来移动引导顺序列表中的项目：

- 如果您没有使用屏幕阅读器，请在您想要移动的项目旁的箭头图标上切换，拖动或下移项目，然后将其放到您选择的位置。
- 如果您使用屏幕阅读器，请按上箭头或者下箭头键移动引导顺序列表中的项目。然后，按 Tab 键将项目放到您选择的位置。

6. 点 **Save**。



注意

如果虚拟机正在运行，对引导顺序列表的更改将在重启虚拟机后才会生效。

您可以点 **Boot Order** 字段右侧的 **View Pending Changes** 查看待处理的修改。页面顶部的 **Pending Changes** 标题显示虚拟机重启时将应用的所有更改列表。

7.7.3. 在 YAML 配置文件中编辑引导顺序列表

使用 CLI 编辑 YAML 配置文件中的引导顺序列表。

流程

1. 运行以下命令为虚拟机打开 YAML 配置文件：

```
$ oc edit vm <vm_name> -n <namespace>
```

2. 编辑 YAML 文件并修改与磁盘或网络接口控制器 (NIC) 关联的引导顺序值。例如：

```
disks:
  - bootOrder: 1 (1)
    disk:
      bus: virtio
    name: containerdisk
  - disk:
      bus: virtio
    name: cloudinitdisk
  - cdrom:
      bus: virtio
    name: cd-drive-1
interfaces:
  - boot Order: 2 (2)
    macAddress: '02:96:c4:00:00'
    masquerade: {}
    name: default
```

(1) 为磁盘指定的引导顺序值。

(2) 为网络接口控制器指定的引导顺序值。

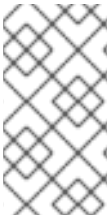
3. 保存 YAML 文件。

7.7.4. 从 web 控制台中的引导顺序列表中删除项目

使用 Web 控制台从引导顺序列表中移除项目。

流程

1. 在侧边菜单中点 **Virtualization → VirtualMachines**。
2. 选择虚拟机以打开 **VirtualMachine 详情**页面。
3. 点 **Details** 标签页。
4. 点击位于 **Boot Order** 右侧的铅笔图标。
5. 点击项 旁边的 **Remove** 图标。该项目从引导顺序列表中删除，可用引导源列表的内容被保存。如果您从引导顺序列表中删除所有项目，则会显示以下消息: **No resource selected.虚拟机会根据在 YAML 文件中的顺序从磁盘引导。**



注意

如果虚拟机正在运行，在重启虚拟机后对 **Boot Order** 的更改不会生效。

您可以点 **Boot Order** 字段右侧的 **View Pending Changes** 查看待处理的修改。页面顶部的 **Pending Changes** 标题显示虚拟机重启时将应用的所有更改列表。

7.8. 删除虚拟机

您可从 web 控制台或使用 **oc** 命令行删除虚拟机。

7.8.1. 使用 web 控制台删除虚拟机

删除虚拟机会将其从集群中永久移除。

流程

1. 在 OpenShift Container Platform 控制台中，从侧边菜单中点 **Virtualization → VirtualMachines**。
2. 点虚拟机旁边的 Options 菜单 并选择 **Delete**。
   或者，击虚拟机名称，打开 **VirtualMachine 详情**页面并点击 **Actions → Delete**。
3. 可选： 选择 **With grace period** 或清除 **Delete disks**。
4. 点 **Delete** 以永久删除虚拟机。

7.8.2. 使用 CLI 删除虚拟机

您可以使用 **oc** 命令行界面（CLI）删除虚拟机。**oc** 客户端允许您在多个虚拟机上执行操作。

先决条件

- 找到要删除的虚拟机名称。

流程

- 运行以下命令以删除虚拟机：

```
$ oc delete vm <vm_name>
```

注意

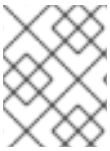
此命令只删除当前项目中的虚拟机。如果您要删除其他项目或命名空间中的虚拟机，请使用 **-n <project_name>** 选项。

7.9. 导出虚拟机

您可以导出虚拟机 (VM) 及其关联的磁盘，以将虚拟机导入到另一个集群或分析卷以备备目的。

您可以使用命令行界面创建一个 **VirtualMachineExport** 自定义资源 (CR)。

另外，您可以使用 **virtctl vmexport** 命令创建一个 **VirtualMachineExport** CR 并下载导出的卷。



注意

您可以使用 Migration Toolkit for Virtualization 在 OpenShift Virtualization 集群间迁移虚拟机。

7.9.1. 创建 VirtualMachineExport 自定义资源

您可以创建一个 **VirtualMachineExport** 自定义资源 (CR) 来导出以下对象：

- 虚拟机 (VM)：导出指定虚拟机的持久性卷声明 (PVC)。
- VM 快照：导出 **VirtualMachineSnapshot** CR 中包含的 PVC。
- PVC ：导出 PVC。如果 PVC 被另一个 pod （如 **virt-launcher** pod）使用，则导出会一直处于 **Pending** 状态，直到 PVC 不再使用为止。

VirtualMachineExport CR 为导出的卷创建内部和外部链接。内部链接在集群中有效。可以使用 **Ingress** 或 **Route** 访问外部链接。

导出服务器支持以下文件格式：

- **raw**: 原始磁盘镜像文件。
- **gzip** ：压缩的磁盘镜像文件.
- **dir** ：PVC 目录和文件。
- **tar.gz** ：压缩的 PVC 文件。

先决条件

- 必须为虚拟机导出关闭虚拟机。

流程

1. 创建一个 **VirtualMachineExport** 清单，根据以下示例从 **VirtualMachine**、**VirtualMachineSnapshot** 或 **PersistentVolumeClaim** CR 导出卷，并将其保存为 **example-export.yaml** ：

VirtualMachineExport 示例

```
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
metadata:
  name: example-export
spec:
  source:
    apiGroup: "kubevirt.io" (1)
    kind: VirtualMachine (2)
    name: example-vm
  ttlDuration: 1h (3)
```

(1) 指定适当的 API 组：

- **"kubevirt.io"** 用于 **VirtualMachine**。
- **"snapshot.kubevirt.io"** 用于 **VirtualMachineSnapshot**。
- **""** 用于 **PersistentVolumeClaim**。

(2) 指定 **VirtualMachine**, **VirtualMachineSnapshot**, 或 **PersistentVolumeClaim**。

(3) 可选。默认持续时间为 2 小时。

2. 创建 **VirtualMachineExport** CR：

```
$ oc create -f example-export.yaml
```

3. 获取 **VirtualMachineExport** CR：

```
$ oc get vmexport example-export -o yaml
```

导出的卷的内部和外部链接显示在 **status** 小节中：

输出示例

```
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
metadata:
  name: example-export
  namespace: example
spec:
  source:
    apiGroup: ""
    kind: PersistentVolumeClaim
    name: example-pvc
  tokenSecretRef: example-token
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2022-06-21T14:10:09Z"
    reason: podReady
```

```
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: "2022-06-21T14:09:02Z"
    reason: pvcBound
    status: "True"
    type: PVCReady
  links:
    external: 1
      cert: |-
        -----BEGIN CERTIFICATE-----
        ...
        -----END CERTIFICATE-----
      volumes:
      - formats:
        - format: raw
          url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/exam
ple-export/volumes/example-disk/disk.img
        - format: gzip
          url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/exam
ple-export/volumes/example-disk/disk.img.gz
        name: example-disk
    internal: 2
      cert: |-
        -----BEGIN CERTIFICATE-----
        ...
        -----END CERTIFICATE-----
      volumes:
      - formats:
        - format: raw
          url: https://virt-export-example-export.example.svc/volumes/example-disk/disk.img
        - format: gzip
          url: https://virt-export-example-export.example.svc/volumes/example-disk/disk.img.gz
        name: example-disk
  phase: Ready
  serviceName: virt-export-example-export
```

1 可以使用 **Ingress** 或 **Route** 从集群外部访问外部链接。

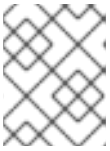
2 内部链接只在集群内有效。

7.9.2. 访问导出的虚拟机清单

导出虚拟机 (VM) 或快照后，您可以从导出服务器获取 **VirtualMachine** 清单和相关信息。

先决条件

- 您可以通过创建一个 **VirtualMachineExport** 自定义资源 (CR) 来导出虚拟机或虚拟机快照。



注意

具有 **spec.source.kind: PersistentVolumeClaim** 参数的 **VirtualMachineExport** 对象不会生成虚拟机清单。

流程

1. 要访问清单，您必须首先将证书从源集群复制到目标集群。
   a. 登录到源集群。
   b. 运行以下命令，将证书保存到 **cacert.crt** 文件中：

```
$ oc get vmexport <export_name> -o jsonpath={.status.links.external.cert} > cacert.crt
1
```

   1 使用 **VirtualMachineExport** 对象中的 **metadata.name** 值替换 **<export_name>**。

   c. 将 **cacert.crt** 文件复制到目标集群。
2. 运行以下命令，解码源集群中的令牌并将其保存到 **token_decode** 文件中：

```
$ oc get secret export-token-<export_name> -o jsonpath={.data.token} | base64 --decode >
token_decode 1
```

   1 使用 **VirtualMachineExport** 对象中的 **metadata.name** 值替换 **<export_name>**。

3. 将 **token_decode** 文件复制到目标集群。
4. 运行以下命令来获取 **VirtualMachineExport** 自定义资源：

```
$ oc get vmexport <export_name> -o yaml
```

5. 查看 **status.links** 小节，该小节被分为 **external** 和 **internal** 部分。请注意每个部分中的 **manifests.url** 字段：

输出示例

```
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
metadata:
  name: example-export
spec:
  source:
    apiGroup: "kubevirt.io"
    kind: VirtualMachine
    name: example-vm
  tokenSecretRef: example-token
status:
#...
  links:
    external:
#...
```

```
      manifests:
      - type: all
        url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/exam
ple-export/external/manifests/all (1)
      - type: auth-header-secret
        url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/exam
ple-export/external/manifests/secret (2)
    internal:
#...
      manifests:
      - type: all
        url: https://virt-export-export-pvc.default.svc/internal/manifests/all (3)
      - type: auth-header-secret
        url: https://virt-export-export-pvc.default.svc/internal/manifests/secret
  phase: Ready
  serviceName: virt-export-example-export
```

(1) 包含 **VirtualMachine** 清单、**DataVolume** 清单（如果存在），以及包含外部 URL ingress 或路由的公共证书的 **ConfigMap** 清单。

(2) 包含与 Containerized Data Importer (CDI) 兼容的标头的 secret。标头包含导出令牌的文本版本。

(3) 包含 **VirtualMachine** 清单、**DataVolume** 清单（如果存在），以及包含内部 URL 导出服务器证书的 **ConfigMap** 清单。

6. 登录到目标集群。

7. 运行以下命令来获取 **Secret** 清单：

```
$ curl --cacert cacert.crt <secret_manifest_url> -H \ (1)
"x-kubevirt-export-token:token_decode" -H \ (2)
"Accept:application/yaml"
```

(1) 将 **<secret_manifest_url>** 替换为 **VirtualMachineExport** YAML 输出中的 **auth-header-secret** URL。

(2) 引用之前创建的 **token_decode** 文件。

例如：

```
$ curl --cacert cacert.crt https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/exam
ple-export/external/manifests/secret -H "x-kubevirt-export-token:token_decode" -H
"Accept:application/yaml"
```

8. 运行以下命令，获取 **type: all** 清单，如 **ConfigMap** 和 **VirtualMachine** 清单：

```
$ curl --cacert cacert.crt <all_manifest_url> -H \ (1)
"x-kubevirt-export-token:token_decode" -H \ (2)
"Accept:application/yaml"
```

1. 将 **<all_manifest_url>** 替换为 **VirtualMachineExport** YAML 输出中的 URL。
2. 引用之前创建的 **token_decode** 文件。

例如：

```
$ curl --cacert cacert.crt https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/exam
ple-export/external/manifests/all -H "x-kubevirt-export-token:token_decode" -H
"Accept:application/yaml"
```

后续步骤

- 现在，您可以使用导出的清单在目标集群中创建 **ConfigMap** 和 **VirtualMachine** 对象。

7.10. 管理虚拟机实例

如果您在 OpenShift Virtualization 环境之外创建独立虚拟机实例(VMI)，您可以使用 web 控制台或使用命令行界面(CLI)使用 **oc** 或 **virtctl** 命令管理它们。

virtctl 命令提供比 **oc** 命令更多的虚拟化选项。例如，您可以使用 **virtctl** 暂停虚拟机或公开端口。

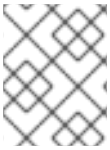
7.10.1. 关于虚拟机实例

虚拟机实例（VMI）代表正在运行的虚拟机(VM)。当某个 VMI 属于某个虚拟机或者其他对象，您可通过 web 控制台中的拥有者或使用 **oc** 命令行界面（CLI）来管理它。

通过自动化或其他 CLI 的方法使用脚本创建并启动独立 VMI。在您的环境中，您可能会在 OpenShift Virtualization 环境之外开发并启动的独立 VMI。您可以使用 CLI 继续管理这些独立的 VMI。您还可以将 Web 控制台用于与独立 VMI 关联的特定任务：

- 列出独立 VMI 及其详情。
- 编辑独立 VMI 的标签和注解。
- 删除独立 VMI。

当删除虚拟机时，相关的 VMI 会被自动删除。您直接删除一个独立的 VMI，因为它不归 VM 或其他对象所有。



注意

在卸载 OpenShift Virtualization 前，使用 CLI 或 Web 控制台列出并查看独立 VMI。然后，删除所有未完成的 VMI。

当您编辑虚拟机时，一些设置可能会动态地应用到 VMI 中，而无需重启。对无法动态应用到 VMI 的虚拟机对象所做的任何更改都会触发 **RestartRequired** VM 条件。更改在下次重启时有效，并删除了条件。

7.10.2. 使用 CLI 列出所有虚拟机实例

您可以使用 **oc** 命令行界面（CLI）列出集群中的所有虚拟机实例（VMI），包括独立 VMI 和虚拟机拥有的实例。

流程

- 运行以下命令列出所有 VMI：

```
$ oc get vmis -A
```

7.10.3. 使用 web 控制台列出独立虚拟机实例

使用 web 控制台，您可以列出并查看集群中不属于虚拟机（VM）的独立虚拟机实例（VMI）。

注意

受 VM 或其他对象拥有的 VMI 不会被显示在 web 控制台中。web 控制台仅显示独立 VMI。如果要列出集群中的所有 VMI，则必须使用 CLI。

流程

- 在侧边菜单中点 **Virtualization → VirtualMachines**。
  您可以在名称旁使用黑色徽标识别独立 VMI。

7.10.4. 使用 web 控制台编辑独立虚拟机实例

您可以使用 web 控制台编辑独立虚拟机实例（VMI）的注解和标签。其他字段不可编辑。

流程

1. 在 OpenShift Container Platform 控制台中，从侧边菜单中点 **Virtualization → VirtualMachines**。
2. 选择独立 VMI 以打开 **VirtualMachineInstance 详情**页面。
3. 在 **Details** 标签页中，点 **Annotations** 或 **Labels** 旁边的铅笔图标。
4. 进行相关的更改并点击 **Save**。

7.10.5. 使用 CLI 删除独立虚拟机实例

您可以使用 **oc** CLI 删除独立虚拟机实例。

先决条件

- 找出要删除的 VMI 的名称。

流程

- 运行以下命令来创建 VMI：

```
$ oc delete vmi <vmi_name>
```

7.10.6. 使用 web 控制台删除独立虚拟机实例

从 web 控制台删除独立虚拟机实例（VMI）。

流程

1. 在 OpenShift Container Platform web 控制台中，从侧边菜单中点 **Virtualization → VirtualMachines**。
2. 点 **Actions → Delete VirtualMachineInstance**。
3. 在弹出的确认窗口中点击 **Delete** 永久删除独立的 VMI。

7.11. 控制虚拟机状态

您可从 web 控制台来停止、启动和重启虚拟机。

您可使用 **virtctl** 管理虚拟机状态并从 CLI 执行其他操作。例如，您可以使用 **virtctl** 来强制停止虚拟机或公开端口。

7.11.1. 启动虚拟机

您可从 web 控制台启动虚拟机。

流程

1. 在侧边菜单中点 **Virtualization → VirtualMachines**。
2. 找到包含要启动的虚拟机的行。
3. 导航到适合您的用例的菜单：
   - 要保留此页面（您可以在其中对多个虚拟机执行操作）：
     a. 点击行右末尾的 Options 菜单  并点 **Start VirtualMachine**。
   - 在启动虚拟机前，要查看有关所选虚拟机的综合信息：
     a. 点虚拟机名称访问 **VirtualMachine 详情**页面。
     b. 点 **Actions → Start**。

注意

首次启动从 **URL** 源置备的虚拟机时，当 OpenShift Virtualization 从 URL 端点导入容器时，虚拟机将处于 **Importing** 状态。根据镜像大小，该过程可能需要几分钟时间。

7.11.2. 停止虚拟机

您可从 web 控制台停止虚拟机。

流程

1. 在侧边菜单中点 **Virtualization → VirtualMachines**。
2. 找到包含您要停止的虚拟机的行。
3. 导航到适合您的用例的菜单：

- 要保留此页面（您可以在其中对多个虚拟机执行操作）：
  a. 点击位于行右边的 Options 菜单  ，然后点 **Stop VirtualMachine**。
- 在停止之前，查看所选虚拟机的综合信息：
  a. 点虚拟机名称访问 **VirtualMachine 详情**页面。
  b. 点 **Actions → Stop**。

7.11.3. 重启虚拟机

您可从 web 控制台重启正在运行的虚拟机。

重要

为了避免错误，不要重启状态为 **Importing** 的虚拟机。

流程

1. 在侧边菜单中点 **Virtualization → VirtualMachines**。
2. 找到包含要启动的虚拟机的行。
3. 导航到适合您的用例的菜单：
   - 要保留此页面（您可以在其中对多个虚拟机执行操作）：
     a. 点击位于行右边的 Options 菜单  并点 **重启**。
   - 要在重启前查看有关所选虚拟机的综合信息：
     a. 点虚拟机名称访问 **VirtualMachine 详情**页面。
     b. 点 **Actions → Restart**。

7.11.4. 暂停虚拟机

您可从 web 控制台暂停虚拟机。

流程

1. 在侧边菜单中点 **Virtualization → VirtualMachines**。
2. 找到包含您要暂停的虚拟机的行。
3. 导航到适合您的用例的菜单：
   - 要保留此页面（您可以在其中对多个虚拟机执行操作）：
     a. 点击位于行右边的 Options 菜单  ，然后点 **暂停 VirtualMachine**。

- 在暂停前，要查看有关所选虚拟机的综合信息：
  a. 点虚拟机名称访问 **VirtualMachine 详情**页面。
  b. 点 **Actions → Pause**。

7.11.5. 取消暂停虚拟机

您可从 web 控制台取消暂停一个正暂停的虚拟机。

先决条件

- 至少一个虚拟机的状态是 **Paused**。

流程

1. 在侧边菜单中点 **Virtualization → VirtualMachines**。
2. 找到包含您要取消暂停的虚拟机的行。
3. 导航到适合您的用例的菜单：
   - 要保留此页面（您可以在其中对多个虚拟机执行操作）：
     a. 点击行右末尾的 Options 菜单  ，然后点 **Unpause VirtualMachine**。
   - 要在取消暂停之前查看所选虚拟机的综合信息：
     a. 点虚拟机名称访问 **VirtualMachine 详情**页面。
     b. 点 **Actions → Unpause**。

7.12. 使用虚拟可信平台模块设备

通过编辑 **VirtualMachine** (VM)或 **VirtualMachineInstance** (VMI)清单，将虚拟 Trusted Platform 模块(vTPM)设备添加到新的或现有虚拟机中。

7.12.1. 关于 vTPM 设备

虚拟可信平台模块(vTPM)设备功能，如物理信任平台模块(TPM)硬件芯片。

您可以将 vTPM 设备与任何操作系统一起使用，但 Windows 11 需要存在 TPM 芯片用来安装或引导的 TPM 芯片。vTPM 设备允许从 Windows 11 镜像创建的虚拟机在没有物理 TPM 芯片的情况下正常工作。

如果没有启用 vTPM，则虚拟机无法识别 TPM 设备，即使节点有一个。

vTPM 设备还通过在没有物理硬件的情况下存储 secret 来保护虚拟机。OpenShift Virtualization 支持为虚拟机使用持久性卷声明 (PVC) 来持久保留 vTPM 设备状态。您必须通过在 **HyperConverged** 自定义资源 (CR) 中设置 **vmStateStorageClass** 属性来指定 PVC 使用的存储类：

```
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
```

```
  vmStateStorageClass: <storage_class_name>

# ...
```

注意

存储类必须是 **Filesystem** 类型，并支持 **ReadWriteMany** (RWX) 访问模式。

7.12.2. 将 vTPM 设备添加到虚拟机

将虚拟 Trusted Platform 模块(vTPM)设备添加到虚拟机(VM)可让您从 Windows 11 镜像创建的虚拟机，而无需物理 TPM 设备。vTPM 设备还存储该虚拟机的 secret。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 您已将持久性卷声明 (PVC) 配置为使用支持 **ReadWriteMany** (RWX) 访问模式的 **Filesystem** 类型的存储类。这是 vTPM 设备数据在虚拟机重启后保留所必需的。

流程

1. 运行以下命令以更新虚拟机配置：

```
$ oc edit vm <vm_name> -n <namespace>
```

2. 编辑虚拟机规格以添加 vTPM 设备。例如：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          tpm: 1
            persistent: true 2
# ...
```

1. 将 vTPM 设备添加到虚拟机。
2. 指定 vTPM 设备状态在虚拟机关闭后保留。默认值为 **false**。

3. 若要应用您的更改，请保存并退出编辑器。
4. 可选：如果编辑了正在运行的虚拟机，您必须重启它才能使更改生效。

7.13. 使用 OPENSHIFT PIPELINES 管理虚拟机

Red Hat OpenShift Pipelines 是一个 Kubernetes 原生 CI/CD 框架，允许开发人员在其自己的容器中设计和运行 CI/CD 管道的每个步骤。

Scheduling、Scale 和 Performance (SSP) Operator 将 OpenShift Virtualization 与 OpenShift Pipelines 集成。SSP Operator 包括允许您的任务和示例管道：

- 创建和管理虚拟机 (VM)、持久性卷声明 (PVC) 和数据卷
- 在虚拟机中运行命令
- 使用 **libguestfs** 工具操作磁盘镜像

7.13.1. 先决条件

- 您可以使用 **cluster-admin** 权限访问 OpenShift Container Platform 集群。
- 已安装 OpenShift CLI(**oc**)。
- 已安装 OpenShift Pipelines。

7.13.2. SSP Operator 支持的虚拟机任务

下表显示了 SSP Operator 中包含的任务。

表 7.4. SSP Operator 支持的虚拟机任务

任务	描述
create-vm-from-manifest	从提供的清单或使用 **virtctl** 创建虚拟机。
create-vm-from-template	从模板创建虚拟机。
copy-template	复制虚拟机模板。
modify-vm-template	修改虚拟机模板。
modify-data-object	创建和删除数据卷或数据源。
cleanup-vm	在虚拟机上运行脚本或命令，并在之后停止或删除虚拟机。
disk-virt-customize	使用 **virt-customize** 工具在目标 PVC 上运行自定义脚本。
disk-virt-sysprep	使用 **virt-sysprep** 工具在目标 PVC 上运行 sysprep 脚本。
wait-for-vmi-status	等待虚拟机实例的特定状态，并根据状态失败或成功。

注意

在管道中创建虚拟机现在使用 **ClusterInstanceType** 和 **ClusterPreference** 而不是基于模板的任务，这些任务已弃用。**create-vm-from-template**、**copy-template** 和 **modify-vm-template** 命令仍然可用，但不用于默认管道任务。

7.13.3. Windows EFI 安装程序管道

您可以使用 Web 控制台或 CLI 运行 Windows EFI 安装程序管道。

Windows EFI 安装程序管道将 Windows 10、Windows 11 或 Windows Server 2022 安装到来自 Windows 安装镜像 (ISO 文件) 的新数据卷中。自定义应答文件用于运行安装过程。

注意

Windows EFI 安装程序管道使用带有 OpenShift Container Platform 预定义的 **sysprep** 的配置映射文件，并适合 Microsoft ISO 文件。对于与不同 Windows 版本相关的 ISO 文件，可能需要创建一个新的配置映射文件，并带有特定于系统的 **sysprep** 定义。

7.13.3.1. 使用 Web 控制台运行示例管道

您可以从 web 控制台中的 **Pipelines** 菜单运行示例管道。

流程

1. 在侧边菜单中点 **Pipelines** → **Pipelines**。
2. 选择一个管道以打开 **Pipeline 详情**页面。
3. 从 **Actions** 列表中，选择 **Start**。此时会显示 **Start Pipeline** 对话框。
4. 保留参数的默认值，然后点 **Start** 运行管道。**Details** 选项卡跟踪每个任务的进度，并显示管道状态。

7.13.3.2. 使用 CLI 运行示例管道

使用 **PipelineRun** 资源来运行示例管道。**PipelineRun** 对象是管道的运行实例。它使用集群上的特定输入、输出和执行参数来实例化 Pipeline 执行。它还为管道中的每个任务创建一个 **TaskRun** 对象。

流程

1. 要运行 Windows 10 安装程序管道，请创建以下 **PipelineRun** 清单：

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  generateName: windows10-installer-run-
  labels:
    pipelinerun: windows10-installer-run
spec:
  params:
  - name: winImageDownloadURL
    value: <link_to_windows_10_iso> (1)
  pipelineRef:
    name: windows10-installer
```

```
  taskRunSpecs:
    - pipelineTaskName: copy-template
      serviceAccountName: copy-template-task
    - pipelineTaskName: modify-vm-template
      serviceAccountName: modify-vm-template-task
    - pipelineTaskName: create-vm-from-template
      serviceAccountName: create-vm-from-template-task
    - pipelineTaskName: wait-for-vmi-status
      serviceAccountName: wait-for-vmi-status-task
    - pipelineTaskName: create-base-dv
      serviceAccountName: modify-data-object-task
    - pipelineTaskName: cleanup-vm
      serviceAccountName: cleanup-vm-task
  status: {}
```

1 指定 Windows 10 64 位 ISO 文件的 URL。产品语言必须是 English (United States)。

2. 应用 **PipelineRun** 清单：

```
$ oc apply -f windows10-installer-run.yaml
```

3. 要运行 Windows 10 自定义管道，请创建以下 **PipelineRun** 清单：

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  generateName: windows10-customize-run-
  labels:
    pipelinerun: windows10-customize-run
spec:
  params:
    - name: allowReplaceGoldenTemplate
      value: true
    - name: allowReplaceCustomizationTemplate
      value: true
  pipelineRef:
    name: windows10-customize
  taskRunSpecs:
    - pipelineTaskName: copy-template-customize
      serviceAccountName: copy-template-task
    - pipelineTaskName: modify-vm-template-customize
      serviceAccountName: modify-vm-template-task
    - pipelineTaskName: create-vm-from-template
      serviceAccountName: create-vm-from-template-task
    - pipelineTaskName: wait-for-vmi-status
      serviceAccountName: wait-for-vmi-status-task
    - pipelineTaskName: create-base-dv
      serviceAccountName: modify-data-object-task
    - pipelineTaskName: cleanup-vm
      serviceAccountName: cleanup-vm-task
    - pipelineTaskName: copy-template-golden
      serviceAccountName: copy-template-task
    - pipelineTaskName: modify-vm-template-golden
      serviceAccountName: modify-vm-template-task
status: {}
```

4. 应用 **PipelineRun** 清单：

```
$ oc apply -f windows10-customize-run.yaml
```

7.13.4. 其他资源

- 为使用 Red Hat OpenShift Pipelines 的应用程序创建 CI/CD 解决方案
- 创建 Windows 虚拟机

7.14. 高级虚拟机管理

7.14.1. 为虚拟机使用资源配额

为虚拟机创建和管理资源配额。

7.14.1.1. 为虚拟机设置资源配额限制

只有使用请求自动用于虚拟机 (VM) 的资源配额。如果您的资源配额使用限制，则必须为虚拟机手动设置资源限值。资源限值必须至少大于资源请求的 100 MiB。

流程

1. 通过编辑 **VirtualMachine** 清单来为虚拟机设置限值。例如：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: with-limits
spec:
  running: false
  template:
    spec:
      domain:
# ...
        resources:
          requests:
            memory: 128Mi
          limits:
            memory: 256Mi 1
```

1 这个配置被支持，因为 **limits.memory** 值至少比 **requests.memory** 的值大 **100Mi**。

2. 保存 **VirtualMachine** 清单。

7.14.1.2. 其他资源

- 项目的资源配额
- 跨越多个项目的资源配额

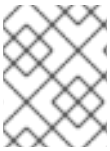
7.14.2. 为虚拟机指定节点

您可以使用节点放置规则将虚拟机放置到特定的节点上。

7.14.2.1. 关于虚拟机的节点放置

要确保虚拟机在适当的节点上运行，您可以配置节点放置规则。如果出现以下情况，您可能需要进行此操作：

- 您有多台虚拟机。为确保容错，您希望它们在不同节点上运行。
- 您有两个 chatty 虚拟机。为了避免冗余节点间路由，您希望虚拟机在同一节点上运行。
- 您的虚拟机需要所有可用节点上不存在的特定硬件功能。
- 您有一个 pod 可以向节点添加功能，并想将虚拟机放置到该节点上，以便它可以使用这些功能。



注意

虚拟机放置依赖于工作负载的现有节点放置规则。如果组件级别上的特定节点排除工作负载，则虚拟机无法放置在这些节点上。

您可以在 **VirtualMachine** 清单的 **spec** 字段中使用以下规则类型：

nodeSelector

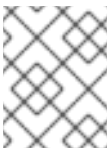
允许将虚拟机调度到使用此字段中指定的键值对标记的节点上。节点必须具有与所有列出的对完全匹配的标签。

关联性

这可让您使用更具表达力的语法来设置与虚拟机匹配的规则。例如，您可以指定规则是首选项，而非硬要求，因此在规则不满足时仍然可以调度虚拟机。虚拟机放置支持 Pod 关联性、pod 反关联性和节点关联性。Pod 关联性适用于虚拟机，因为 **VirtualMachine** 工作负载类型基于 **Pod** 对象。

容限（tolerations）

允许将虚拟机调度到具有匹配污点的节点。如果污点应用到某个节点，则该节点只接受容许该污点的虚拟机。



注意

关联性规则仅在调度期间应用。如果不再满足限制，OpenShift Container Platform 不会重新调度正在运行的工作负载。

7.14.2.2. 节点放置示例

以下示例 YAML 文件片断使用 **nodePlacement**、**affinity** 和 **tolerations** 字段为虚拟机自定义节点放置。

7.14.2.2.1. 示例：使用 nodeSelector 放置虚拟机节点

在本例中，虚拟机需要一个包含 **example-key-1 = example-value-1** 和 **example-key-2 = example-value-2** 标签的元数据的节点。

警告

如果没有节点适合此描述，则不会调度虚拟机。

VM 清单示例

```
metadata:
  name: example-vm-node-selector
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      nodeSelector:
        example-key-1: example-value-1
        example-key-2: example-value-2
# ...
```

7.14.2.2.2. 示例：使用 pod 关联性和 pod 反关联性的虚拟机节点放置

在本例中，虚拟机必须调度到具有标签 **example-key-1 = example-value-1** 的正在运行的 pod 的节点上。如果没有在任何节点上运行这样的 pod，则不会调度虚拟机。

如果可能，虚拟机不会调度到具有标签 **example-key-2 = example-value-2** 的 pod 的节点上。但是，如果所有候选节点都有具有此标签的 pod，调度程序会忽略此约束。

VM 清单示例

```
metadata:
  name: example-vm-pod-affinity
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution: (1)
          - labelSelector:
              matchExpressions:
              - key: example-key-1
                operator: In
                values:
                - example-value-1
            topologyKey: kubernetes.io/hostname
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution: (2)
          - weight: 100
            podAffinityTerm:
              labelSelector:
```

```
            matchExpressions:
            - key: example-key-2
              operator: In
              values:
              - example-value-2
          topologyKey: kubernetes.io/hostname
# ...
```

1 如果您使用 **requiredDuringSchedulingIgnoredDuringExecution** 规则类型，如果没有满足约束，则不会调度虚拟机。

2 如果您使用 **preferredDuringSchedulingIgnoredDuringExecution** 规则类型，只要满足所有必要的限制，仍会调度虚拟机（如果未满足约束）。

7.14.2.2.3. 示例：使用节点关联性进行虚拟机节点放置

在本例中，虚拟机必须调度到具有标签 **example.io/example-key = example-value-1** 或标签 **example.io/example-key = example-value-2** 的节点上。如果节点上只有一个标签，则会满足约束。如果没有标签，则不会调度虚拟机。

若有可能，调度程序会避免具有标签 **example-node-label-key = example-node-label-value** 的节点。但是，如果所有候选节点都具有此标签，调度程序会忽略此限制。

VM 清单示例

```
metadata:
  name: example-vm-node-affinity
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution: 1
            nodeSelectorTerms:
            - matchExpressions:
              - key: example.io/example-key
                operator: In
                values:
                - example-value-1
                - example-value-2
          preferredDuringSchedulingIgnoredDuringExecution: 2
          - weight: 1
            preference:
              matchExpressions:
              - key: example-node-label-key
                operator: In
                values:
                - example-node-label-value
# ...
```

1 如果您使用 **requiredDuringSchedulingIgnoredDuringExecution** 规则类型，如果没有满足约束，则不会调度虚拟机。

2 如果您使用 **preferredDuringSchedulingIgnoredDuringExecution** 规则类型，只要满足所有必要的限制，仍会调度虚拟机（如果未满足约束）。

7.14.2.2.4. 示例：带有容限的虚拟机节点放置

在本例中，为虚拟机保留的节点已使用 **key=virtualization:NoSchedule** 污点标记。由于此虚拟机具有匹配的**容限**，它可以调度到污点节点上。

注意

容许污点的虚拟机不需要调度到具有该污点的节点。

VM 清单示例

```
metadata:
  name: example-vm-tolerations
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  tolerations:
  - key: "key"
    operator: "Equal"
    value: "virtualization"
    effect: "NoSchedule"
# ...
```

7.14.2.3. 其他资源

- 为虚拟化组件指定节点
- 使用节点选择器将 pod 放置到特定节点
- 使用节点关联性规则控制节点上的 pod 放置
- 使用节点污点控制 pod 放置

7.14.3. 激活内核相同的页面合并 (KSM)

当节点过载时，OpenShift Virtualization 可以激活内核相同的页面合并 (KSM)。KSM 去除在虚拟机 (VM) 的内存页面中找到的相同数据。如果您有类似的虚拟机，则 KSM 可以在单个节点上调度更多虚拟机。

重要

您必须只使用带有可信工作负载的 KSM。

7.14.3.1. 先决条件

- 确保管理员已在您希望 OpenShift Virtualization 激活 KSM 的任何节点上配置了 KSM 支持。

7.14.3.2. 关于使用 OpenShift Virtualization 激活 KSM

当节点遇到内存过载时，您可以将 OpenShift Virtualization 配置为激活内核相同的页面合并(KSM)。

7.14.3.2.1. 配置方法

您可以使用 OpenShift Container Platform Web 控制台或编辑 **HyperConverged** 自定义资源(CR)来启用或禁用所有节点的 KSM 激活功能。**HyperConverged** CR 支持更精细的配置。

CR 配置
您可以通过编辑 **HyperConverged** CR 的 **spec.configuration.ksmConfiguration** 小节来配置 KSM 激活功能。

- 您可以通过编辑 **ksmConfiguration** 小节来启用功能和配置设置。
- 您可以通过删除 **ksmConfiguration** 小节来禁用该功能。
- 您可以通过在 **ksmConfiguration.nodeLabelSelector** 字段中添加节点选择语法来允许 OpenShift Virtualization 只在节点子集上启用 KSM。

注意

即使 OpenShift Virtualization 中禁用了 KSM 激活功能，管理员仍然可以在支持它的节点上启用 KSM。

7.14.3.2.2. KSM 节点标签

OpenShift Virtualization 识别配置为支持 KSM 并应用以下节点标签的节点：

kubevirt.io/ksm-handler-managed: "false"
当 OpenShift Virtualization 在遇到内存过载的节点上激活 KSM 时，该标签被设置为 **"true"**。如果管理员激活 KSM，则该标签没有设置为 **"true"**。

kubevirt.io/ksm-enabled: "false"
当节点上激活 KSM 时，此标签被设置为 **"true"**，即使 OpenShift Virtualization 没有激活 KSM。

这些标签不适用于不支持 KSM 的节点。

7.14.3.3. 使用 Web 控制台配置 KSM 激活

您可以使用 OpenShift Container Platform Web 控制台允许 OpenShift Virtualization 在集群中的所有节点上激活内核相同的页面合并 (KSM)。

流程

1. 在侧边菜单中点 **Virtualization → Overview**。
2. 选择 **Settings** 选项卡。
3. 选择 **Cluster** 选项卡。
4. 扩展 **资源管理**。
5. 为所有节点启用或禁用功能：
   - 将 **内核同页合并(KSM)** 设置为 on。
   - 将 **内核同页合并(KSM)** 设置为 off。

7.14.3.4. 使用 CLI 配置 KSM 激活

您可以通过编辑 **HyperConverged** 自定义资源(CR)来启用或禁用 OpenShift Virtualization 内核相同的页面合并(KSM)激活功能。如果您希望 OpenShift Virtualization 只在某个节点子集上激活 KSM，则使用此方法。

流程

1. 运行以下命令，在默认编辑器中打开 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 编辑 **ksmConfiguration** 小节：

- 要为所有节点启用 KSM 激活功能，请将 **nodeLabelSelector** 值设置为 **{}**。例如：

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  configuration:
    ksmConfiguration:
      nodeLabelSelector: {}
# ...
```

- 要在节点的子集上启用 KSM 激活功能，请编辑 **nodeLabelSelector** 字段。添加与 OpenShift Virtualization 启用 KSM 的节点匹配的语法。例如，以下配置允许 OpenShift Virtualization 在 **<first_example_key>** 和 **<second_example_key>** 被设置为 **"true"** 的节点上启用 KSM。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  configuration:
    ksmConfiguration:
      nodeLabelSelector:
        matchLabels:
          <first_example_key>: "true"
          <second_example_key>: "true"
# ...
```

- 要禁用 KSM 激活功能，请删除 **ksmConfiguration** 小节。例如：

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  configuration:
# ...
```

3. 保存该文件。

7.14.3.5. 其他资源

- 为虚拟机指定节点
- 使用节点选择器将 pod 放置到特定节点
- 在 Red Hat Enterprise Linux (RHEL) 文档中 管理内核相同的页面合并

7.14.4. 配置证书轮转

配置证书轮转参数以替换现有证书。

7.14.4.1. 配置证书轮转

您可以在 web 控制台中的 OpenShift Virtualization 安装过程中，或者在安装 **HyperConverged** 自定义资源（CR）后完成此操作。

流程

1. 运行以下命令打开 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 按照以下示例所示，编辑 **spec.certConfig** 字段。要避免系统过载，请确保所有值都大于或等于 10 分钟。将所有值显示为符合 golang **ParseDuration** 格式的字符串。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  certConfig:
    ca:
      duration: 48h0m0s
      renewBefore: 24h0m0s (1)
    server:
      duration: 24h0m0s (2)
      renewBefore: 12h0m0s (3)
```

(1) **ca.renewBefore** 的值必须小于或等于 **ca.duration** 的值。

(2) **server.duration** 的值必须小于或等于 **ca.duration** 的值。

(3) **server.renewBefore** 的值必须小于或等于 **server.duration** 的值。

3. 将 YAML 文件应用到集群。

7.14.4.2. 证书轮转参数故障排除

删除一个或多个 **certConfig** 值会导致它们恢复到默认值，除非默认值与以下条件之一冲突：

- **ca.renewBefore** 的值必须小于或等于 **ca.duration** 的值。
- **server.duration** 的值必须小于或等于 **ca.duration** 的值。
- **server.renewBefore** 的值必须小于或等于 **server.duration** 的值。

如果默认值与这些条件冲突，您将收到错误。

如果您删除了以下示例中的 **server.duration** 值，则默认值 **24h0m0s** 大于 **ca.duration** 的值，并与指定条件冲突。

Example

```
certConfig:
  ca:
    duration: 4h0m0s
    renewBefore: 1h0m0s
  server:
    duration: 4h0m0s
    renewBefore: 4h0m0s
```

这会生成以下出错信息：

```
error: hyperconvergeds.hco.kubevirt.io "kubevirt-hyperconverged" could not be patched: admission webhook "validate-hco.kubevirt.io" denied the request: spec.certConfig: ca.duration is smaller than server.duration
```

错误消息仅提及第一个冲突。在继续操作前，查看所有 certConfig 值。

7.14.5. 配置默认 CPU 型号

使用 **HyperConverged** 自定义资源 (CR) 中的 **defaultCPUModel** 设置来定义集群范围的默认 CPU 模型。

虚拟机 (VM) CPU 模型取决于虚拟机和集群中的 CPU 模型的可用性。

- 如果虚拟机没有定义的 CPU 模型：
  - **defaultCPUModel** 使用在集群范围级别上定义的 CPU 模型自动设置。
- 如果虚拟机和集群都有定义的 CPU 模型：
  - 虚拟机的 CPU 模型具有优先权。
- 如果虚拟机或集群都没有定义的 CPU 模型：
  - host-model 使用主机级别上定义的 CPU 模型自动设置。

7.14.5.1. 配置默认 CPU 型号

通过更新 **HyperConverged** 自定义资源(CR) 来配置 **defaultCPUModel**。您可以在 OpenShift Virtualization 运行时更改 **defaultCPUModel**。

注意

defaultCPUModel 是区分大小写的。

先决条件

- 安装 OpenShift CLI（oc）。

流程

1. 运行以下命令打开 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 将 **defaultCPUModel** 字段添加到 CR，并将值设置为集群中存在的 CPU 模型的名称：

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
 name: kubevirt-hyperconverged
 namespace: openshift-cnv
spec:
  defaultCPUModel: "EPYC"
```

3. 将 YAML 文件应用到集群。

7.14.6. 为虚拟机使用 UEFI 模式

您可以使用统一可扩展固件接口(UEFI)模式引导虚拟机(VM)。

7.14.6.1. 关于虚拟机的 UEFI 模式

像旧的 BIOS 一样，统一可扩展固件接口(UEFI)在计算机启动时初始化硬件组件和操作系统镜像文件。与 BIOS 相比，UEFI 支持更现代的功能和自定义选项，从而加快启动速度。

它将初始化和启动的所有信息保存在带有 **.efi** 扩展的文件中，该扩展被保存在名为 EFI 系统分区（ESP）的特殊分区中。ESP 还包含安装在计算机上的操作系统的引导装载程序程序。

7.14.6.2. 在 UEFI 模式中引导虚拟机

您可以通过编辑 **VirtualMachine** 清单，将虚拟机配置为在 UEFI 模式中引导。

先决条件

- 安装 OpenShift CLI (**oc**) 。

流程

1. 编辑或创建 **VirtualMachine** 清单文件。使用 **spec.firmware.bootloader** 小节来配置 UEFI 模式：

使用安全引导活跃在 UEFI 模式中引导

```
apiversion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    special: vm-secureboot
  name: vm-secureboot
spec:
  template:
    metadata:
      labels:
        special: vm-secureboot
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: containerdisk
        features:
          acpi: {}
          smm:
            enabled: true (1)
        firmware:
          bootloader:
            efi:
              secureBoot: true (2)
# ...
```

1. OpenShift Virtualization 需要为 UEFI 模式的安全引导启用系统管理模式(**SMM**)。
2. 使用 UEFI 模式时，OpenShift Virtualization 支持带有或不进行安全引导的虚拟机。如果启用了安全引导，则需要 UEFI 模式。但是，可以在不使用安全引导的情况下启用 UEFI 模式。

2. 运行以下命令，将清单应用到集群：

```
$ oc create -f <file_name>.yaml
```

7.14.6.3. 启用持久性 EFI

您可以通过在集群级别配置 RWX 存储类并调整虚拟机 EFI 部分中的设置来启用 EFI 持久性。

先决条件

- 您必须具有集群管理员特权。
- 您必须有一个支持 RWX 访问模式和 FS 卷模式的存储类。

流程

- 运行以下命令启用 **VMPersistentState** 功能门：

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op":"replace","path":"/spec/featureGates/VMPersistentState", "value":
true}]'
```

7.14.6.4. 使用持久性 EFI 配置虚拟机

您可以通过编辑清单文件，将虚拟机配置为启用 EFI 持久性。

先决条件

- **VMPersistentState** 功能门启用。

流程

- 编辑虚拟机清单文件并保存以应用设置。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm
spec:
  template:
    spec:
      domain:
        firmware:
          bootloader:
            efi:
              persistent: true
# ...
```

7.14.7. 为虚拟机配置 PXE 启动

OpenShift Virtualization 中提供 PXE 启动或网络启动。网络启动支持计算机启动和加载操作系统或其他程序，无需本地连接的存储设备。例如，在部署新主机时，您可使用 PXE 启动从 PXE 服务器中选择所需操作系统镜像。

7.14.7.1. 先决条件

- Linux 网桥必须已连接。
- PXE 服务器必须作为网桥连接至相同 VLAN。

7.14.7.2. 使用指定的 MAC 地址的 PXE 引导

作为管理员，您可首先为您的 PXE 网络创建 **NetworkAttachmentDefinition** 对象，以此通过网络引导客户端。然后在启动虚拟机实例前，在您的虚拟机实例配置文件中引用网络附加定义。如果 PXE 服务器需要，您还可在虚拟机实例配置文件中指定 MAC 地址。

先决条件

- 必须已连接 Linux 网桥。
- PXE 服务器必须作为网桥连接至相同 VLAN。

流程

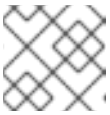
1. 在集群上配置 PXE 网络：

   a. 为 PXE 网络 **pxe-net-conf** 创建网络附加定义文件：

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: pxe-net-conf
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "pxe-net-conf",
    "plugins": [
      {
        "type": "cnv-bridge",
        "bridge": "br1",
        "vlan": 1 (1)
      },
      {
        "type": "cnv-tuning" (2)
      }
    ]
  }'
```

   (1) 可选： VLAN 标签。

   (2) **cnv-tuning** 插件为自定义 MAC 地址提供支持。



注意

虚拟机实例将通过所请求的 VLAN 的访问端口附加到网桥 **br1**。

2. 使用您在上一步中创建的文件创建网络附加定义：

```
$ oc create -f pxe-net-conf.yaml
```

3. 编辑虚拟机实例配置文件以包括接口和网络的详情。

   a. 如果 PXE 服务器需要，请指定网络和 MAC 地址。如果未指定 MAC 地址，则会自动分配一个值。
   请确保 **bootOrder** 设置为 **1**，以便该接口先启动。在本例中，该接口连接到了名为 **<pxe-net>** 的网络中：

```
interfaces:
- masquerade: {}
  name: default
- bridge: {}
  name: pxe-net
  macAddress: de:00:00:00:00:de
  bootOrder: 1
```

注意

启动顺序对于接口和磁盘全局通用。

b. 为磁盘分配一个启动设备号，以确保置备操作系统后能够正确启动。
将磁盘 **bootOrder** 值设置为 **2**：

```
devices:
  disks:
  - disk:
      bus: virtio
    name: containerdisk
    bootOrder: 2
```

c. 指定网络连接到之前创建的网络附加定义。在这种情况下，**<pxe-net>** 连接到名为 **<pxe-net-conf>** 的网络附加定义：

```
networks:
- name: default
  pod: {}
- name: pxe-net
  multus:
    networkName: pxe-net-conf
```

4. 创建虚拟机实例：

```
$ oc create -f vmi-pxe-boot.yaml
```

输出示例

```
 virtualmachineinstance.kubevirt.io "vmi-pxe-boot" created
```

5. 等待虚拟机实例运行：

```
$ oc get vmi vmi-pxe-boot -o yaml | grep -i phase
  phase: Running
```

6. 使用 VNC 查看虚拟机实例：

```
$ virtctl vnc vmi-pxe-boot
```

7. 查看启动屏幕，验证 PXE 启动是否成功。

8. 登录虚拟机实例：

```
$ virtctl console vmi-pxe-boot
```

验证

1. 验证虚拟机上的接口和 MAC 地址，并验证连接到网桥的接口是否具有指定的 MAC 地址。在本例中，我们使用了 **eth1** 进行 PXE 启动，无需 IP 地址。另一接口 **eth0** 从 OpenShift Container Platform 获取 IP 地址。

```
$ ip addr
```

输出示例

```
...
3. eth1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen
1000
  link/ether de:00:00:00:00:de brd ff:ff:ff:ff:ff:ff
```

7.14.7.3. OpenShift Virtualization 术语表

以下是整个 OpenShift Virtualization 文档中使用的术语：

Container Network Interface (CNI)

一个 Cloud Native Computing Foundation 项目，侧重容器网络连接。OpenShift Virtualization 使用 CNI 插件基于基本 Kubernetes 网络功能进行构建。

Multus

一个"meta"CNI 插件，支持多个 CNI 共存，以便 pod 或虚拟机可使用其所需的接口。

自定义资源定义(CRD)

一个 Kubernetes API 资源，用于定义自定义资源，或使用 CRD API 资源定义的对象。

网络附加定义(NAD)

由 Multus 项目引入的 CRD，允许您将 Pod、虚拟机和虚拟机实例附加到一个或多个网络。

节点网络配置策略(NNCP)

nmstate 项目引入的 CRD，描述节点上请求的网络配置。您可以通过将 **NodeNetworkConfigurationPolicy**清单应用到集群来更新节点网络配置，包括添加和删除网络接口。

7.14.8. 在虚拟机中使用巨页

您可以使用巨页作为集群中虚拟机的后备内存。

7.14.8.1. 先决条件

- 节点必须配置预分配的巨页。

7.14.8.2. 巨页的作用

内存在块（称为页）中进行管理。在大多数系统中，页的大小为 4Ki。1Mi 内存相当于 256 个页，1Gi 内存相当于 256,000 个页。CPU 有内置的内存管理单元，可在硬件中管理这些页的列表。Translation Lookaside Buffer (TLB) 是虚拟页到物理页映射的小型硬件缓存。如果在硬件指令中包括的虚拟地址可以在 TLB 中找到，则其映射信息可以被快速获得。如果没有包括在 TLN 中，则称为 TLB miss。系统将会使用基于软件的，速度较慢的地址转换机制，从而出现性能降低的问题。因为 TLB 的大小是固定的，因此降低 TLB miss 的唯一方法是增加页的大小。

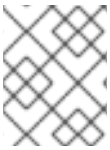
巨页指一个大于 4Ki 的内存页。在 x86_64 构架中，有两个常见的巨页大小: 2Mi 和 1Gi。在其它构架上的大小会有所不同。要使用巨页，必须写相应的代码以便应用程序了解它们。Transparent Huge Pages（THP）试图在应用程序不需要了解的情况下自动管理巨页，但这个技术有一定的限制。特别是，它的页大小会被限为 2Mi。当有较高的内存使用率时，THP 可能会导致节点性能下降，或出现大量内存碎片（因为 THP 的碎片处理）导致内存页被锁定。因此，有些应用程序可能更适用于（或推荐）使用预先分配的巨页，而不是 THP。

在 OpenShift Virtualization 中，可将虚拟机配置为消耗预先分配的巨页。

7.14.8.3. 为虚拟机配置巨页

您可以在虚拟机配置中包括 **memory.hugepages.pageSize** 和 **resources.requests.memory** 参数来配置虚拟机来使用预分配的巨页。

内存请求必须按页大小分离。例如，您不能对大小为 **1Gi** 的页请求 **500Mi** 内存。



注意

主机的内存布局和客户端操作系统不相关。虚拟机清单中请求的巨页适用于 QEMU。客户端中的巨页只能根据虚拟机实例的可用内存量来配置。

如果您编辑了正在运行的虚拟机，则必须重启虚拟机才能使更改生效。

先决条件

- 节点必须配置预先分配的巨页。

流程

1. 在虚拟机配置中，把 **resources.requests.memory** 和 **memory.hugepages.pageSize** 参数添加到 **spec.domain**。以下配置片段适用于请求总计 **4Gi** 内存的虚拟机，页面大小为 **1Gi**:

```
kind: VirtualMachine
# ...
spec:
  domain:
    resources:
      requests:
        memory: "4Gi" (1)
    memory:
      hugepages:
        pageSize: "1Gi" (2)
# ...
```

(1) 为虚拟机请求的总内存量。这个值必须可以被按页大小整除。

(2) 每个巨页的大小。x86_64 架构的有效值为 **1Gi** 和 **2Mi**。页面大小必须小于请求的内存。

2. 应用虚拟机配置：

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

7.14.9. 为虚拟机启用专用资源

要提高性能，您可以将节点的资源（如 CPU）专用于特定的一个虚拟机。

7.14.9.1. 关于专用资源

当为您的虚拟机启用专用资源时，您的工作负载将会在不会被其他进程使用的 CPU 上调度。通过使用专用资源，您可以提高虚拟机性能以及延迟预测的准确性。

7.14.9.2. 先决条件

- 节点上必须配置 CPU Manager。在调度虚拟机工作负载前，请确认节点具有 **cpumanager = true** 标签。
- 虚拟机必须关机。

7.14.9.3. 为虚拟机启用专用资源

您可以在 **Details** 选项卡中为虚拟机启用专用资源。从红帽模板创建的虚拟机可以使用专用资源进行配置。

流程

1. 在 OpenShift Container Platform 控制台中，从侧边菜单中点 **Virtualization → VirtualMachines**。
2. 选择虚拟机以打开 **VirtualMachine 详情**页面。
3. 在 **Configuration → Scheduling** 选项卡中，点 **Dedicated Resources** 旁边的编辑图标。
4. 选择 **Schedule this workload with dedicated resources (guaranteed policy)**。
5. 点 **Save**。

7.14.10. 调度虚拟机

在确保虚拟机的 CPU 模型和策略属性与节点支持的 CPU 模型和策略属性兼容的情况下，可在节点上调度虚拟机（VM）。

7.14.10.1. 策略属性

您可以指定策略属性和在虚拟机调度到节点上时匹配的 CPU 功能来调度虚拟机（VM）。为虚拟机指定的策略属性决定了如何在节点上调度该虚拟机。

策略属性	描述
force	VM 被强制调度到某个节点上。即使主机 CPU 不支持虚拟机的 CPU，也是如此。
require	在虚拟机没有使用特定 CPU 模型和功能规格配置时，应用于虚拟机的默认策略。如果节点没有配置为支持使用此默认策略属性或其他策略属性的 CPU 节点发现，则虚拟机不会调度到该节点上。主机 CPU 必须支持虚拟机的 CPU，或者虚拟机监控程序必须可以模拟支持的 CPU 模型。
optional	如果主机物理机器 CPU 支持该虚拟机，则虚拟机会被添加到节点。
disable	无法通过 CPU 节点发现调度虚拟机。

策略属性	描述
forbid	即使主机 CPU 支持该功能，且启用了 CPU 节点发现，也不会调度虚拟机。

7.14.10.2. 设置策略属性和 CPU 功能

您可以为每个虚拟机（VM）设置策略属性和 CPU 功能，以确保根据策略和功能在节点上调度该功能。验证您设置的 CPU 功能以确保主机 CPU 支持或者虚拟机监控程序模拟该功能。

流程

- 编辑虚拟机配置文件的 **domain** spec。以下示例设置虚拟机 (VM) 的 CPU 功能和 **require** 策略：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          features:
            - name: apic (1)
              policy: require (2)
```

(1) 虚拟机的 CPU 功能名称。

(2) 虚拟机的策略属性.

7.14.10.3. 使用支持的 CPU 型号调度虚拟机

您可以为虚拟机 (VM) 配置 CPU 模型，将其调度到支持其 CPU 模型的节点。

流程

- 编辑虚拟机配置文件的 **domain** spec。以下示例显示了为虚拟机定义的特定 CPU 模型：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          model: Conroe (1)
```

(1) 虚拟机的 CPU 模型.

7.14.10.4. 使用主机模型调度虚拟机

当将虚拟机（VM）的 CPU 模型设置为 **host-model** 时，虚拟机会继承调度节点的 CPU 模型。

流程

- 编辑虚拟机配置文件的 **domain** spec。以下示例演示了为虚拟机指定 **host-model** ：

```
apiVersion: kubevirt/v1alpha3
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          model: host-model 1
```

1 继承调度节点的 CPU 模型的虚拟机。

7.14.10.5. 使用自定义调度程序调度虚拟机

您可以使用自定义调度程序在节点上调度虚拟机 (VM)。

先决条件

- 为集群配置二级调度程序。

流程

- 通过编辑 **VirtualMachine** 清单，将自定义调度程序添加到虚拟机配置中。例如：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-fedora
spec:
  running: true
  template:
    spec:
      schedulerName: my-scheduler 1
      domain:
        devices:
          disks:
            - name: containerdisk
              disk:
                bus: virtio
# ...
```

1 自定义调度程序的名称。如果 **schedulerName** 值与现有调度程序不匹配，**virt-launcher** pod 会一直处于 **Pending** 状态，直到找到指定的调度程序为止。

验证

- 通过检查 **virt-launcher** pod 事件来验证虚拟机是否使用 **VirtualMachine** 清单中指定的自定义调度程序：

  a. 输入以下命令来查看集群中的 pod 列表：

  ```
  $ oc get pods
  ```

  输出示例

  ```
  NAME                            READY   STATUS    RESTARTS   AGE
  virt-launcher-vm-fedora-dpc87   2/2     Running   0          24m
  ```

  b. 运行以下命令以显示 pod 事件：

  ```
  $ oc describe pod virt-launcher-vm-fedora-dpc87
  ```

  输出中的 **From** 字段的值验证调度程序名称与 **VirtualMachine** 清单中指定的自定义调度程序匹配：

  输出示例

  ```
  [...]
  Events:
    Type    Reason     Age   From            Message
    ----    ------     ----  ----            -------
    Normal  Scheduled  21m   my-scheduler  Successfully assigned default/virt-launcher-vm-fedora-dpc87 to node01
  [...]
  ```

其他资源

- 部署二级调度程序

7.14.11. 配置 PCI 透传

通过 Peripheral Component Interconnect (PCI) 透传功能，您可以从虚拟机 (VM) 访问和管理硬件设备。配置 PCI 透传后，PCI 设备的功能就如同它们实际上附加到客户机操作系统上一样。

集群管理员可以使用 **oc** CLI 来公开和管理集群中允许在集群中使用的主机设备。

7.14.11.1. 为 GPU 透传准备节点

您可以防止 GPU 操作对象部署到您为 GPU 透传指定的 worker 节点上。

7.14.11.1.1. 防止在节点上部署 NVIDIA GPU 操作对象

如果在集群中使用 NVIDIA GPU Operator，您可以将 **nvidia.com/gpu.deploy.operands=false** 标签应用到您不想为 GPU 或 vGPU 操作对象配置的节点。该标签可防止创建配置 GPU 或 vGPU 操作对象的 pod，并在 pod 已存在时终止 pod。

先决条件

- 已安装 OpenShift CLI (**oc**)。

流程

- 运行以下命令标记节点：

```
$ oc label node <node_name> nvidia.com/gpu.deploy.operands=false 1
```

1 将 **<node_name>** 替换为您要安装 NVIDIA GPU 操作对象的节点名称。

验证

1. 运行以下命令，验证标签是否已添加到节点：

```
$ oc describe node <node_name>
```

2. 可选： 如果之前在节点上部署了 GPU 操作对象，请验证其删除。

   a. 运行以下命令，检查 **nvidia-gpu-operator** 命名空间中的 pod 状态：

```
$ oc get pods -n nvidia-gpu-operator
```

输出示例

```
NAME                           READY   STATUS       RESTARTS   AGE
gpu-operator-59469b8c5c-hw9wj    1/1     Running      0          8d
nvidia-sandbox-validator-7hx98   1/1     Running      0          8d
nvidia-sandbox-validator-hdb7p   1/1     Running      0          8d
nvidia-sandbox-validator-kxwj7   1/1     Terminating  0          9d
nvidia-vfio-manager-7w9fs        1/1     Running      0          8d
nvidia-vfio-manager-866pz        1/1     Running      0          8d
nvidia-vfio-manager-zqtck        1/1     Terminating  0          9d
```

   b. 监控 pod 状态，直到具有 **Terminating** 状态的 pod 被删除：

```
$ oc get pods -n nvidia-gpu-operator
```

输出示例

```
NAME                           READY   STATUS    RESTARTS   AGE
gpu-operator-59469b8c5c-hw9wj    1/1     Running   0          8d
nvidia-sandbox-validator-7hx98   1/1     Running   0          8d
nvidia-sandbox-validator-hdb7p   1/1     Running   0          8d
nvidia-vfio-manager-7w9fs        1/1     Running   0          8d
nvidia-vfio-manager-866pz        1/1     Running   0          8d
```

7.14.11.2. 为 PCI 透传准备主机设备

7.14.11.2.1. 关于为 PCI 透传准备主机设备

要使用 CLI 为 PCI 透传准备主机设备，请创建一个 **MachineConfig** 对象并添加内核参数，以启用输入输

出内存管理单元（IOMMU）。将 PCI 设备绑定到虚拟功能 I/O（VFIO）驱动程序，然后通过编辑 **HyperConverged** 自定义资源（CR）的 **allowedHostDevices** 字段在集群中公开它。首次安装 OpenShift Virtualization Operator 时，**allowedHostDevices** 列表为空。

要使用 CLI 从集群中删除 PCI 主机设备，可从 **HyperConverged** CR 中删除 PCI 设备信息。

7.14.11.2.2. 添加内核参数以启用 IOMMU 驱动程序

要在内核中启用 IOMMU 驱动程序，请创建 **MachineConfig** 对象并添加内核参数。

先决条件

- 有集群管理员权限。
- 您的 CPU 硬件是 Intel 或 AMD。
- 您在 BIOS 中为直接 I/O 扩展或 AMD IOMMU 启用 Intel 虚拟化技术。

流程

1. 创建用于标识内核参数的 **MachineConfig** 对象。以下示例显示了 Intel CPU 的内核参数。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker 1
  name: 100-worker-iommu 2
spec:
  config:
    ignition:
      version: 3.2.0
  kernelArguments:
      - intel_iommu=on 3
# ...
```

1. 仅将新内核参数应用到 worker 节点。
2. **name** 表示此内核参数（100）在机器配置及其目的中的排名。如果您有 AMD CPU，请将内核参数指定为 **amd_iommu=on**。
3. 将内核参数标识为 Intel CPU 的 **intel_iommu**。

2. 创建新的 **MachineConfig** 对象：

```
$ oc create -f 100-worker-kernel-arg-iommu.yaml
```

验证

- 验证是否添加了新的 **MachineConfig** 对象。

```
$ oc get MachineConfig
```

7.14.11.2.3. 将 PCI 设备绑定到 VFIO 驱动程序

要将 PCI 设备绑定到 VFIO（虚拟功能 I/O）驱动程序，请从每个设备获取 **vendor-ID** 和 **device-ID** 的值，并创建值的列表。将这个列表添加到 **MachineConfig** 对象。**MachineConfig** Operator 在带有 PCI 设备的节点上生成 **/etc/modprobe.d/vfio.conf**，并将 PCI 设备绑定到 VFIO 驱动程序。

先决条件

- 您添加了内核参数来为 CPU 启用 IOMMU。

流程

1. 运行 **lspci** 命令，以获取 PCI 设备的 **vendor-ID** 和 **device-ID**。

```
$ lspci -nnv | grep -i nvidia
```

输出示例

```
02:01.0 3D controller [0302]: NVIDIA Corporation GV100GL [Tesla V100 PCIe 32GB] [10de:1eb8] (rev a1)
```

2. 创建 Butane 配置文件 **100-worker-vfiopci.bu**，将 PCI 设备绑定到 VFIO 驱动程序。

注意

有关 Butane 的信息，请参阅"使用 Butane 创建机器配置"。

Example

```
variant: openshift
version: 4.16.0
metadata:
  name: 100-worker-vfiopci
  labels:
    machineconfiguration.openshift.io/role: worker 1
storage:
  files:
  - path: /etc/modprobe.d/vfio.conf
    mode: 0644
    overwrite: true
    contents:
      inline: |
        options vfio-pci ids=10de:1eb8 2
  - path: /etc/modules-load.d/vfio-pci.conf 3
    mode: 0644
    overwrite: true
    contents:
      inline: vfio-pci
```

1 仅将新内核参数应用到 worker 节点。

2 指定之前确定的 **vendor-ID** 值（**10de**）和 **device-ID** 值（**1eb8**）来将单个设备绑定到 VFIO 驱动程序。您可以使用其供应商和设备信息添加多个设备列表。

3 在 worker 节点上载入 vfio-pci 内核模块的文件。

3. 使用 Butane 生成 **MachineConfig** 对象文件 **100-worker-vfiopci.yaml**，包含要发送到 worker 节点的配置：

```
$ butane 100-worker-vfiopci.bu -o 100-worker-vfiopci.yaml
```

4. 将 **MachineConfig** 对象应用到 worker 节点：

```
$ oc apply -f 100-worker-vfiopci.yaml
```

5. 验证 **MachineConfig** 对象是否已添加。

```
$ oc get MachineConfig
```

输出示例

```
NAME                             GENERATEDBYCONTROLLER                      IGNITIONVERSION
AGE
00-master                        d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0             25h
00-worker                        d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0             25h
01-master-container-runtime      d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
01-master-kubelet                d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
01-worker-container-runtime      d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
01-worker-kubelet                d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
100-worker-iommu                                                            3.2.0             30s
100-worker-vfiopci-configuration                                            3.2.0             30s
```

验证

- 验证是否已加载 VFIO 驱动程序。

```
$ lspci -nnk -d 10de:
```

输出确认使用了 VFIO 驱动程序。

输出示例

```
04:00.0 3D controller [0302]: NVIDIA Corporation GP102GL [Tesla P40] [10de:1eb8] (rev a1)
        Subsystem: NVIDIA Corporation Device [10de:1eb8]
        Kernel driver in use: vfio-pci
        Kernel modules: nouveau
```

7.14.11.2.4. 使用 CLI 在集群中公开 PCI 主机设备

要在集群中公开 PCI 主机设备，将 PCI 设备的详细信息添加到 **HyperConverged** 自定义资源（CR）的 **spec.permittedHostDevices.pciHostDevices** 数组中。

流程

1. 运行以下命令，在默认编辑器中编辑 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 将 PCI 设备信息添加到 **spec.percommitHostDevices.pciHostDevices** 数组。例如：

配置文件示例

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  permittedHostDevices: 1
    pciHostDevices: 2
    - pciDeviceSelector: "10DE:1DB6" 3
      resourceName: "nvidia.com/GV100GL_Tesla_V100" 4
    - pciDeviceSelector: "10DE:1EB8"
      resourceName: "nvidia.com/TU104GL_Tesla_T4"
    - pciDeviceSelector: "8086:6F54"
      resourceName: "intel.com/qat"
      externalResourceProvider: true 5
# ...
```

1. 允许在集群中使用的主机设备。
2. 节点上可用的 PCI 设备列表。
3. 标识 PCI 设备所需的 **vendor-ID** 和 **device-ID**。
4. PCI 主机设备的名称。
5. 可选：将此字段设置为 **true** 表示资源由外部设备插件提供。OpenShift Virtualization 允许在集群中使用这个设备，但会把分配和监控留给外部设备插件。

注意

上例代码片段显示有两个 PCI 主机设备，名为 **nvidia.com/GV100GL_Tesla_V100** 和 **nvidia.com/TU104GL_Tesla_Tesla_T4**。它们被添加到 **HyperConverged** CR 中的允许主机设备列表中。这些设备已经过测试和验证以用于 OpenShift Virtualization。

3. 保存更改并退出编辑器。

验证

- 运行以下命令，验证 PCI 主机设备是否已添加到节点。示例输出显示，每个设备都与 **nvidia.com/GV100GL_Tesla_V100**、**nvidia.com/TU104GL_Tesla_T4** 和 **intel.com/qat** 资源名称关联。

```
$ oc describe node <node_name>
```

输出示例

```
Capacity:
  cpu:                          64
  devices.kubevirt.io/kvm:      110
  devices.kubevirt.io/tun:      110
  devices.kubevirt.io/vhost-net:  110
  ephemeral-storage:            915128Mi
  hugepages-1Gi:                0
  hugepages-2Mi:                0
  memory:                       131395264Ki
  nvidia.com/GV100GL_Tesla_V100   1
  nvidia.com/TU104GL_Tesla_T4     1
  intel.com/qat:                1
  pods:                         250
Allocatable:
  cpu:                          63500m
  devices.kubevirt.io/kvm:      110
  devices.kubevirt.io/tun:      110
  devices.kubevirt.io/vhost-net:  110
  ephemeral-storage:            863623130526
  hugepages-1Gi:                0
  hugepages-2Mi:                0
  memory:                       130244288Ki
  nvidia.com/GV100GL_Tesla_V100   1
  nvidia.com/TU104GL_Tesla_T4     1
  intel.com/qat:                1
  pods:                         250
```

7.14.11.2.5. 使用 CLI 从集群中删除 PCI 主机设备

要从集群中删除 PCI 主机设备，请从 **HyperConverged** 自定义资源（CR）中删除该设备的信息。

流程

1. 运行以下命令，在默认编辑器中编辑 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 通过删除相应设备的 **pciDeviceSelector**、**resourceName** 和 **externalResourceProvider**（如果适用）字段来从 **spec.permittedHostDevices.pciHostDevices** 阵列中删除 PCI 设备信息。在本例中，**intel.com/qat** 资源已被删除。

配置文件示例

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  permittedHostDevices:
```

```
    pciHostDevices:
    - pciDeviceSelector: "10DE:1DB6"
      resourceName: "nvidia.com/GV100GL_Tesla_V100"
    - pciDeviceSelector: "10DE:1EB8"
      resourceName: "nvidia.com/TU104GL_Tesla_T4"
# ...
```

3. 保存更改并退出编辑器。

验证

- 运行以下命令，验证 PCI 主机设备已从节点移除。示例输出显示，与 **intel.com/qat** 资源名称关联的设备为零。

```
$ oc describe node <node_name>
```

输出示例

```
Capacity:
  cpu:                          64
  devices.kubevirt.io/kvm:      110
  devices.kubevirt.io/tun:      110
  devices.kubevirt.io/vhost-net:  110
  ephemeral-storage:            915128Mi
  hugepages-1Gi:                0
  hugepages-2Mi:                0
  memory:                       131395264Ki
  nvidia.com/GV100GL_Tesla_V100   1
  nvidia.com/TU104GL_Tesla_T4     1
  intel.com/qat:                0
  pods:                         250
Allocatable:
  cpu:                          63500m
  devices.kubevirt.io/kvm:      110
  devices.kubevirt.io/tun:      110
  devices.kubevirt.io/vhost-net:  110
  ephemeral-storage:            863623130526
  hugepages-1Gi:                0
  hugepages-2Mi:                0
  memory:                       130244288Ki
  nvidia.com/GV100GL_Tesla_V100   1
  nvidia.com/TU104GL_Tesla_T4     1
  intel.com/qat:                0
  pods:                         250
```

7.14.11.3. 为 PCI 透传配置虚拟机

将 PCI 设备添加到集群中后，您可以将它们分配到虚拟机。PCI 设备现在可用。就像它们被物理地连接到虚拟机一样。

7.14.11.3.1. 为虚拟机分配 PCI 设备

当集群中有 PCI 设备时，您可以将其分配到虚拟机并启用 PCI 透传。

流程

- 将 PCI 设备分配到虚拟机作为主机设备。

  Example

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  domain:
    devices:
      hostDevices:
      - deviceName: nvidia.com/TU104GL_Tesla_T4 (1)
        name: hostdevices1
```

(1) 集群中作为主机设备允许的 PCI 设备的名称。虚拟机可以访问此主机设备。

验证

- 使用以下命令，验证主机设备可从虚拟机使用。

```
$ lspci -nnk | grep NVIDIA
```

输出示例

```
$ 02:01.0 3D controller [0302]: NVIDIA Corporation GV100GL [Tesla V100 PCIe 32GB]
[10de:1eb8] (rev a1)
```

7.14.11.4. 其他资源

- 在 BIOS 中启用 Intel VT-X 和 AMD-V 虚拟化硬件扩展
- 管理文件权限
- 机器配置概述

7.14.12. 配置虚拟 GPU

如果您有图形处理单元(GPU)卡，OpenShift Virtualization 可以自动创建您可以分配给虚拟机(VM)的虚拟 GPU (vGPU)。

7.14.12.1. 关于在 OpenShift Virtualization 中使用虚拟 GPU

有些图形处理单元(GPU)卡支持创建虚拟 GPU(vGPU)。如果管理员在 **HyperConverged** 自定义资源(CR)中提供配置详情，则 OpenShift Virtualization 可以自动创建 vGPU 和其他介质设备。这个自动化对大型集群特别有用。

注意

有关功能和支持详情，请参考您的硬件供应商文档。

介质设备

划分为一个或多个虚拟设备的物理设备。vGPU 是一个介质设备(mdev)类型，物理 GPU 的性能会被划分到各个虚拟设备中。您可以将介质设备分配给一个或多个虚拟机(VM)，但客户机数量必须与您的 GPU 兼容。有些 GPU 不支持多个虚拟机。

7.14.12.2. 为介质设备准备主机

在配置介质设备前，您必须启用输入输出内存管理单元 (IOMMU) 驱动程序。

7.14.12.2.1. 添加内核参数以启用 IOMMU 驱动程序

要在内核中启用 IOMMU 驱动程序，请创建 **MachineConfig** 对象并添加内核参数。

先决条件

- 有集群管理员权限。
- 您的 CPU 硬件是 Intel 或 AMD。
- 您在 BIOS 中为直接 I/O 扩展或 AMD IOMMU 启用 Intel 虚拟化技术。

流程

1. 创建用于标识内核参数的 **MachineConfig** 对象。以下示例显示了 Intel CPU 的内核参数。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker 1
  name: 100-worker-iommu 2
spec:
  config:
    ignition:
      version: 3.2.0
  kernelArguments:
      - intel_iommu=on 3
# ...
```

1. 仅将新内核参数应用到 worker 节点。
2. **name** 表示此内核参数（100）在机器配置及其目的中的排名。如果您有 AMD CPU，请将内核参数指定为 **amd_iommu=on**。
3. 将内核参数标识为 Intel CPU 的 **intel_iommu**。

2. 创建新的 **MachineConfig** 对象：

```
$ oc create -f 100-worker-kernel-arg-iommu.yaml
```

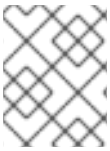
验证

- 验证是否添加了新的 **MachineConfig** 对象。

```
$ oc get MachineConfig
```

7.14.12.3. 配置 NVIDIA GPU Operator

您可以使用 NVIDIA GPU Operator 置备 worker 节点，以便在 OpenShift Virtualization 中运行 GPU 加速虚拟机(VM)。



注意

NVIDIA GPU Operator 仅支持 NVIDIA。如需更多信息，请参阅红帽知识库中的从 NVIDIA 获得支持。

7.14.12.3.1. 关于使用 NVIDIA GPU Operator

您可以将 NVIDIA GPU Operator 与 OpenShift Virtualization 搭配使用来快速置备 worker 节点来运行启用了 GPU 的虚拟机 (VM)。NVIDIA GPU Operator 在 OpenShift Container Platform 集群中管理 NVIDIA GPU 资源，并自动执行为 GPU 工作负载准备节点时所需的任务。

在将应用程序工作负载部署到 GPU 资源前，您必须安装如 NVIDIA 驱动程序，如启用计算统一设备架构 (CUDA)、Kubernetes 设备插件、容器运行时和其他功能，如自动节点标签和监控。通过自动化这些任务，您可以快速扩展基础架构的 GPU 容量。NVIDIA GPU Operator 有助于置备复杂智能和机器学习 (AI/ML) 工作负载。

7.14.12.3.2. 配置介质设备的选项

使用 NVIDIA GPU Operator 时有两种可用的配置介质设备的方法。红帽测试的方法使用 OpenShift Virtualization 功能来调度介质设备，而 NVIDIA 方法只使用 GPU Operator。

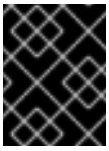
使用 NVIDIA GPU Operator 配置介质设备

这个方法只使用 NVIDIA GPU Operator 来配置介质设备。要使用这个方法，请参阅 NVIDIA 文档中的带有 OpenShift Virtualization 的 NVIDIA GPU Operator。

使用 OpenShift Virtualization 配置介质设备

这个方法由红帽测试，使用 OpenShift Virtualization 的功能来配置介质设备。在这种情况下，NVIDIA GPU Operator 仅用于使用 NVIDIA vGPU Manager 安装驱动程序。GPU Operator 不配置介质设备。使用 OpenShift Virtualization 方法时，您仍遵循 NVIDIA 文档 配置 GPU Operator。但是，此方法与 NVIDIA 文档的以下方法不同：

- 您不能覆盖 **HyperConverged** 自定义资源(CR) 中的默认 **disableMDEVConfiguration: false** 设置。

  重要

  按照 NVIDIA 文档所述 设置此功能门可防止 OpenShift Virtualization 配置介质设备。

- 您必须配置 **ClusterPolicy** 清单，使其与以下示例匹配：

  清单示例

  ```
  kind: ClusterPolicy
  apiVersion: nvidia.com/v1
  metadata:
  ```

```
  name: gpu-cluster-policy
spec:
  operator:
    defaultRuntime: crio
    use_ocp_driver_toolkit: true
    initContainer: {}
  sandboxWorkloads:
    enabled: true
    defaultWorkload: vm-vgpu
  driver:
    enabled: false 1
  dcgmExporter: {}
  dcgm:
    enabled: true
  daemonsets: {}
  devicePlugin: {}
  gfd: {}
  migManager:
    enabled: true
  nodeStatusExporter:
    enabled: true
  mig:
    strategy: single
  toolkit:
    enabled: true
  validator:
    plugin:
      env:
        - name: WITH_WORKLOAD
          value: "true"
  vgpuManager:
    enabled: true 2
    repository: <vgpu_container_registry> 3
    image: <vgpu_image_name>
    version: nvidia-vgpu-manager
  vgpuDeviceManager:
    enabled: false 4
    config:
      name: vgpu-devices-config
      default: default
  sandboxDevicePlugin:
    enabled: false 5
  vfioManager:
    enabled: false 6
```

1. 将此值设置为 **false**。虚拟机不需要。
2. 将此值设置为 **true**。将 vGPU 与虚拟机搭配使用是必需的。
3. 将 **<vgpu_container_registry>** 替换为您的 registry 值。
4. 将此值设置为 **false**，以允许 OpenShift Virtualization 配置介质设备而不是 NVIDIA GPU Operator。
5. 将此值设置为 **false** 以防止发现 vGPU 设备并将 vGPU 设备公告到 kubelet。

6 将此值设置为 **false** 以防止加载 **vfio-pci** 驱动程序。相反，请按照 OpenShift Virtualization 文档来配置 PCI 透传。

其他资源

- 配置 PCI 透传

7.14.12.4. vGPU 如何分配给节点

对于每个物理设备，OpenShift Virtualization 配置以下值：

- 单个 mdev 类型。
- 所选 **mdev** 类型的最大实例数量。

集群架构会影响创建设备并分配到节点的方式。

每个节点具有多个卡的大型集群

在支持多个 vGPU 类型的节点上，以轮循方式创建相关设备类型。例如：

```
# ...
mediatedDevicesConfiguration:
  mediatedDeviceTypes:
  - nvidia-222
  - nvidia-228
  - nvidia-105
  - nvidia-108
# ...
```

在这种情况下，每个节点有两个卡，它们支持以下 vGPU 类型：

```
nvidia-105
# ...
nvidia-108
nvidia-217
nvidia-299
# ...
```

在每个节点上，OpenShift Virtualization 会创建以下 vGPU：

- 在第一个卡上，16 个类型为 nvidia-105 的 vGPU。
- 第二卡上的 2 个类型为 nvidia-108 的 vGPU。

一个节点有一个卡，它支持多个请求的 vGPU 类型

OpenShift Virtualization 使用最先在 **mediatedDeviceTypes** 列表中提供的支持的类型。
例如，节点卡中的卡支持 **nvidia-223** 和 **nvidia-224**。配置了以下 **mediatedDeviceTypes** 列表：

```
# ...
mediatedDevicesConfiguration:
  mediatedDeviceTypes:
  - nvidia-22
```

```
  - nvidia-223
  - nvidia-224
# ...
```

在本例中，OpenShift Virtualization 使用 **nvidia-223** 类型。

7.14.12.5. 管理介质设备

在向虚拟机分配介质设备前，您必须创建设备并将其公开给集群。您还可以重新配置和删除介质设备。

7.14.12.5.1. 创建并公开介质设备

作为管理员，您可以通过编辑 **HyperConverged** 自定义资源(CR)来创建介质设备并将其公开给集群。

先决条件

- 启用了输入输出内存管理单元(IOMMU)驱动程序。
- 如果您的硬件厂商提供驱动程序，您可以在要创建介质设备的节点上安装它们。
  - 如果您使用 NVIDIA 卡，则 安装了 NVIDIA GRID 驱动程序。

流程

1. 运行以下命令，在默认编辑器中打开 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

例 7.1. 配置了介质设备的配置文件示例

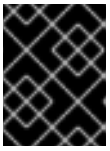
```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  mediatedDevicesConfiguration:
    mediatedDeviceTypes:
    - nvidia-231
    nodeMediatedDeviceTypes:
    - mediatedDeviceTypes:
      - nvidia-233
      nodeSelector:
        kubernetes.io/hostname: node-11.redhat.com
  permittedHostDevices:
    mediatedDevices:
    - mdevNameSelector: GRID T4-2Q
      resourceName: nvidia.com/GRID_T4-2Q
    - mdevNameSelector: GRID T4-8Q
      resourceName: nvidia.com/GRID_T4-8Q
# ...
```

2. 通过在 **spec.mediatedDevicesConfiguration** 小节中添加来创建介质设备：

YAML 片断示例

```
# ...
spec:
  mediatedDevicesConfiguration:
    mediatedDeviceTypes: 1
    - <device_type>
    nodeMediatedDeviceTypes: 2
    - mediatedDeviceTypes: 3
      - <device_type>
      nodeSelector: 4
        <node_selector_key>: <node_selector_value>
# ...
```

1. 必需：为集群配置全局设置。
2. 可选：覆盖特定节点或一组节点的全局配置。必须与全局 **mediatedDeviceTypes** 配置一起使用。
3. 使用 **nodeMediatedDeviceTypes** 时需要此项。覆盖指定节点的全局 **mediatedDeviceTypes** 配置。
4. 使用 **nodeMediatedDeviceTypes** 时需要此项。必须包含一个 **key:value** 对。



重要

在 OpenShift Virtualization 4.14 之前，**mediatedDeviceTypes** 字段被命名为 **mediatedDevicesTypes**。确定在配置介质设备时使用正确的字段名称。

3. 识别您要公开给集群的设备的 name selector 和 resource name 值。您将在下一步中将这些值添加到 **HyperConverged** CR 中。

   a. 运行以下命令来查找 **resourceName** 值：

```
$ oc get $NODE -o json \
  | jq '.status.allocatable \
    | with_entries(select(.key | startswith("nvidia.com/"))) \
    | with_entries(select(.value != "0"))'
```

   b. 通过查看 **/sys/bus/pci/devices/<slot>:<bus>:<domain>.<function>/mdev_supported_types/<type>/name** 的内容来查找 **mdevNameSelector** 值，替换您的系统的正确值。
   例如，**nvidia-231** 类型的名称文件包含选择器字符串 **GRID T4-2Q**。使用 **GRID T4-2Q** 作为 **mdevNameSelector** 值，允许节点使用 **nvidia-231** 类型。

4. 通过将 **mdevNameSelector** 和 **resourceName** 值添加到 **HyperConverged** CR 的 **spec.permittedHostDevices.mediatedDevices** 小节来向集群公开介质设备：

YAML 片断示例

```
# ...
```

```
  permittedHostDevices:
    mediatedDevices:
    - mdevNameSelector: GRID T4-2Q (1)
      resourceName: nvidia.com/GRID_T4-2Q (2)
# ...
```

(1) 公开映射到主机上这个值的介质设备。

(2) 匹配节点上分配的资源名称。

5. 保存更改并退出编辑器。

验证

- 可选：通过运行以下命令确认将设备添加到特定节点：

```
$ oc describe node <node_name>
```

7.14.12.5.2. 关于更改和删除介质设备

您可以通过几种方式重新配置或删除介质设备：

- 编辑 **HyperConverged** CR 并更改 **mediatedDeviceTypes** 小节的内容。
- 更改与 **nodeMediatedDeviceTypes** 节点选择器匹配的节点标签。
- 从 **HyperConverged** CR 的 **spec.mediatedDevicesConfiguration** 和 **spec.permittedHostDevices** 小节中删除设备信息。

注意

如果您在 **spec.permittedHostDevices** 小节中删除设备信息，且没有将其从 **spec.mediatedDevicesConfiguration** 小节中移除，则无法在同一节点上创建新的介质设备类型。要正确删除介质设备，请从两个段中删除设备信息。

7.14.12.5.3. 从集群中删除介质设备

要从集群中删除介质设备，请从 **HyperConverged** 自定义资源(CR)中删除该设备的信息。

流程

1. 运行以下命令，在默认编辑器中编辑 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 从 **HyperConverged** CR 的 **spec.mediatedDevicesConfiguration** 和 **spec.permittedHostDevices** 小节中删除设备信息。删除这两个条目可确保您稍后在同一节点上创建新的介质设备类型。例如：

配置文件示例

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
```

```
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  mediatedDevicesConfiguration:
    mediatedDeviceTypes: 1
      - nvidia-231
  permittedHostDevices:
    mediatedDevices: 2
    - mdevNameSelector: GRID T4-2Q
      resourceName: nvidia.com/GRID_T4-2Q
```

1. 要删除 **nvidia-231** 设备类型，请从 **mediatedDeviceTypes** 阵列中删除它。
2. 要删除 **GRID T4-2Q** 设备，请删除 **mdevNameSelector** 字段及其对应的 **resourceName** 字段。

3. 保存更改并退出编辑器。

7.14.12.6. 使用介质设备

您可以将介质设备分配给一个或多个虚拟机。

7.14.12.6.1. 使用 CLI 将 vGPU 分配给虚拟机

为虚拟机(VM)分配介质设备，如虚拟 GPU (vGPU)。

先决条件

- 介质设备在 **HyperConverged** 自定义资源中配置。
- 虚拟机已停止。

流程

- 通过编辑 **VirtualMachine** 清单的 **spec.domain.devices.gpus** 小节，将介质设备分配给虚拟机(VM)：

  虚拟机清单示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  domain:
    devices:
      gpus:
      - deviceName: nvidia.com/TU104GL_Tesla_T4 1
        name: gpu1 2
      - deviceName: nvidia.com/GRID_T4-2Q
        name: gpu2
```

1. 与介质设备关联的资源名称。

2 用于标识虚拟机上设备的名称。

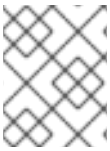
验证

- 要验证该设备在虚拟机中可用，运行以下命令，将 **<device_name>** 替换为 **VirtualMachine** 清单中的 **deviceName** 值：

```
$ lspci -nnk | grep <device_name>
```

7.14.12.6.2. 使用 Web 控制台为虚拟机分配 vGPU

您可以使用 OpenShift Container Platform web 控制台为虚拟机分配虚拟 GPU。



注意

您可以将硬件设备添加到从自定义模板或 YAML 文件创建的虚拟机中。您不能将设备添加到特定操作系统的预先提供的引导源模板中。

先决条件

- vGPU 配置为集群中的介质设备。
  - 要查看连接到集群的设备，请从侧边菜单中点 **Compute → Hardware Devices**。
- 虚拟机已停止。

流程

1. 在 OpenShift Container Platform web 控制台中，从侧边菜单中点 **Virtualization → VirtualMachines**。
2. 选择您要为其分配该设备的虚拟机。
3. 在 **Details** 标签页中，点 **GPU 设备**。
4. 点 **Add GPU 设备**。
5. 在 **Name** 字段中输入识别值。
6. 在 **Device name** 列表中选择您要添加到虚拟机的设备。
7. 点击 **Save**。

验证

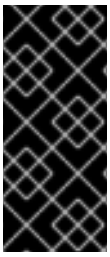
- 要确认设备已添加到虚拟机，请点 **YAML** 选项卡并查看 **VirtualMachine** 配置。介质设备添加到 **spec.domain.devices** 小节中。

7.14.12.7. 其他资源

- 在 BIOS 中启用 Intel VT-X 和 AMD-V 虚拟化硬件扩展

7.14.13. 在虚拟机上启用 descheduler 驱除

您可以使用 descheduler 来驱除 pod，以便可将 pod 重新调度到更合适的节点上。如果 pod 是虚拟机，pod 驱除会导致虚拟机实时迁移到另一节点。



重要

虚拟机的 descheduler 驱除功能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。

7.14.13.1. Descheduler 配置集

使用技术预览 **DevPreviewLifecycle** 配置集为虚拟机启用 descheduler。这是当前可用于 OpenShift Virtualization 的 descheduler 配置集。为确保正确调度，请创建带有 CPU 和内存请求的虚拟机用于预期的负载。

DevPreviewLongLifecycle

此配置集在节点间平衡资源使用量并启用以下策略：

- **RemovePodsHavingTooManyRestarts** ：删除其容器重启次数太多的 pod，以及其中所有容器（包括 Init 容器）重启的总数超过 100 的 pod。重启虚拟机客户端操作系统不会增加这个计数。
- **LowNodeUtilization** ：在存在没有被充分利用的节点时，将 pod 从过度使用的节点上驱除。被驱除的 pod 的目标节点将由调度程序决定。
  - 如果节点的用量低于 20%（CPU、内存和 pod 的数量）,则该节点将被视为使用率不足。
  - 如果节点的用量超过 50%（CPU、内存和 pod 的数量）,则该节点将被视为过量使用。

7.14.13.2. 安装 descheduler

在默认情况下，不提供 descheduler。要启用 descheduler，您必须从 OperatorHub 安装 Kube Descheduler Operator，并启用一个或多个 descheduler 配置集。

默认情况下，descheduler 以预测模式运行，这意味着它只模拟 pod 驱除。您必须将 descheduler 的模式更改为自动进行 pod 驱除。

重要

如果您在集群中启用了托管的 control plane，设置自定义优先级阈值，以降低托管 control plane 命名空间中的 pod 被驱除。将优先级阈值类名称设置为 **hypershift-control-plane**，因为它有托管的 control plane 优先级类的最低优先级值（**100000000**）。

先决条件

- 以具有 **cluster-admin** 角色的用户身份登录到 OpenShift Container Platform。
- 访问 OpenShift Container Platform Web 控制台。

流程

1. 登陆到 OpenShift Container Platform Web 控制台。

2. 为 Kube Descheduler Operator 创建所需的命名空间。

   a. 进行 **Administration → Namespaces**，点 **Create Namespace**。

   b. 在 **Name** 字段中输入 **openshift-kube-descheduler-operator**，在 **Labels** 字段中输入 **openshift.io/cluster-monitoring=true** 来启用 descheduler 指标，然后点击 **Create**。

3. 安装 Kube Descheduler Operator。

   a. 进入 **Operators → OperatorHub**。

   b. 在过滤框中输入 **Kube Descheduler Operator**。

   c. 选择 **Kube Descheduler Operator** 并点 **Install**。

   d. 在 **Install Operator** 页面中，选择 **A specific namespace on the cluster**。从下拉菜单中选择 **openshift-kube-descheduler-operator**。

   e. 将 **Update Channel** 和 **Approval Strategy** 的值调整为所需的值。

   f. 点击 **Install**。

4. 创建 descheduler 实例。

   a. 在 **Operators → Installed Operators** 页面中，点 **Kube Descheduler Operator**。

   b. 选择 **Kube Descheduler** 标签页并点 **Create KubeDescheduler**。

   c. 根据需要编辑设置。

      i. 要驱除 pod 而不是模拟驱除，请将 **Mode** 字段更改为 **Automatic**。

      ii. 展开 **Profiles** 部分，再选择 **DevPreviewLongLifecycle**。**AffinityAndTaints** 配置集默认为启用。

重要

当前仅适用于 OpenShift Virtualization 的配置集是 **DevPreviewLongLifecycle**。

您还可以稍后使用 OpenShift CLI(**oc**)为 descheduler 配置配置集和设置。

7.14.13.3. 在虚拟机(VM)上启用 descheduler 驱除

安装 descheduler 后，您可以通过在 **VirtualMachine** 自定义资源(CR)中添加注解来在虚拟机上启用 descheduler 驱除。

先决条件

- 在 OpenShift Container Platform Web 控制台或 OpenShift CLI(**oc**)中安装 descheduler。
- 确保虚拟机没有运行。

流程

1. 在启动虚拟机前，将 **descheduler.alpha.kubernetes.io/evict** 注解添加到 **VirtualMachine** CR:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    metadata:
      annotations:
        descheduler.alpha.kubernetes.io/evict: "true"
```

2. 如果您还没有在安装过程中在 web 控制台中设置 **DevPreviewLongLifecycle** 配置集，请在 **KubeDescheduler** 对象的 **spec.profile** 部分指定 **DevPreviewLongLifecycle** ：

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
  name: cluster
  namespace: openshift-kube-descheduler-operator
spec:
  deschedulingIntervalSeconds: 3600
  profiles:
  - DevPreviewLongLifecycle
  mode: Predictive 1
```

1 默认情况下，descheduler 不会驱除 pod。要驱除 pod，请将 **mode** 设置为 **Automatic**。

现在在虚拟机上启用了 descheduler。

7.14.13.4. 其他资源

- Descheduler 概述

7.14.14. 关于虚拟机的高可用性

您可以通过手动删除故障节点来触发虚拟机故障切换或配置补救节点，为虚拟机 (VM) 启用高可用性。

手动删除出现故障的节点

如果节点失败，且集群中没有部署机器健康检查，则带有 **runStrategy: Always** 配置的虚拟机不会自动重新定位到健康的节点。要触发虚拟机故障切换，您必须手动删除 **Node** 对象。

请参阅删除失败的节点来触发虚拟机故障切换。

配置补救节点

您可以通过从 OperatorHub 安装 Self Node Remediation Operator 或 Fence Agents Remediation Operator 来配置补救节点，并启用机器健康检查或节点补救检查。

有关补救、隔离和维护节点的更多信息，请参阅 Workload Availability for Red Hat OpenShift 文档。

7.14.15. 虚拟机 control plane 调整

OpenShift Virtualization 在 control-plane 级别提供以下调整选项：

- **highBurst** 配置集（使用固定 **QPS** 和 **burst** 率）在一个批处理中创建数百个虚拟机 (VM)

- 基于工作负载类型的迁移设置调整

7.14.15.1. 配置 highBurst 配置集

使用 **highBurst** 配置集在一个集群中创建和维护大量虚拟机(VM)。

流程

- 应用以下补丁以启用 **highBurst** 调优配置文件：

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  --type=json -p='[{"op": "add", "path": "/spec/tuningPolicy", \
  "value": "highBurst"}]'
```

验证

- 运行以下命令，以验证启用了 **highBurst** 调优配置文件：

```
$ oc get kubevirt.kubevirt.io/kubevirt-kubevirt-hyperconverged \
  -n openshift-cnv -o go-template --template='{{range $config, \
  $value := .spec.configuration}} {{if eq $config "apiConfiguration" \
  "webhookConfiguration" "controllerConfiguration" "handlerConfiguration"}} \
  {{"\n"}} {{$config}} = {{$value}} {{end}} {{end}} {{"\n"}}
```

7.14.16. 分配计算资源

在 OpenShift Virtualization 中，分配给虚拟机的计算资源由保证 CPU 或时间分片 CPU 共享提供支持。

保证的 CPU （也称为 CPU 保留）将 CPU 内核或线程专用于特定工作负载，从而使它们对任何其他工作负载都不可用。为虚拟机分配有保证的 CPU 可确保虚拟机只能访问保留的物理 CPU。为虚拟机启用专用资源以使用保证 CPU。

时间分片 CPU 将一个时间分片用于每个工作负载共享物理 CPU。您可以指定虚拟机创建过程中或虚拟机离线期间的分片大小。默认情况下，每个 vCPU 收到 100 毫秒，或一个物理 CPU 时间的 1/10 秒。

CPU 保留类型取决于实例类型或虚拟机配置。

7.14.16.1. 过度分配 CPU 资源

时间分片允许多个虚拟 CPU (vCPU) 共享单个物理 CPU。这称为 *CPU 过量分配（CPU overcommitment）*。保证虚拟机不会被过度分配。

配置 CPU 过量分配，以便在为虚拟机分配 CPU 时优先选择虚拟机密度。对于 vCPU 的 CPU 过量分配，更多虚拟机适合给定节点。

7.14.16.2. 设置 CPU 分配比率

CPU 分配率通过将 vCPU 映射到物理 CPU 时间分片来指定过量分配程度。

例如，一个映射或 10:1 的比例通过使用时间片段将 10 个虚拟 CPU 映射到 1 个物理 CPU。

要更改映射到每个物理 CPU 的默认 vCPU 数量，请在 **HyperConverged** CR 中设置 **vmiCPUAllocationRatio** 值。pod CPU 请求是通过 CPU 分配比率重新处理的 vCPU 数量来计算的。例如，如果 **vmiCPUAllocationRatio** 设置为 10，OpenShift Virtualization 会在该虚拟机的 pod 上请求 10

倍的 CPU。

流程

在 **HyperConverged** CR 中设置 **vmiCPUAllocationRatio** 值来定义节点 CPU 分配比率。

1. 运行以下命令，在默认编辑器中打开 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 设置 **vmiCPUAllocationRatio** ：

```
...
spec:
  resourceRequirements:
    vmiCPUAllocationRatio: 1 (1)
# ...
```

(1) 当 **vmiCPUAllocationRatio** 设置为 **1** 时，容器集请求的最大 vCPU 数量。

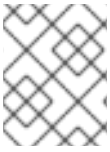
7.14.16.3. 其他资源

- Pod 服务质量类

7.14.17. 关于多队列功能

使用多队列功能在具有多个 vCPU 的虚拟机 (VM) 上扩展网络吞吐量和性能。

默认情况下，从域 XML 派生的 **queueCount** 值由分配给虚拟机的 vCPU 数量决定。随着 vCPU 数量增加，网络性能无法扩展。另外，因为 virtio-net 只有一个 Tx 和 Rx 队列，所以客户机无法并行传输或检索软件包。



注意

当客户机实例中的 vNIC 数量与 vCPU 数量成比例时，启用 virtio-net 多队列并不会显著改进。

7.14.17.1. 已知限制

- 如果在主机中启用了 virtio-net multiqueue，但管理员未在客户机操作系统中启用它，则 MSI vectors 仍然会被使用。
- 每个 virtio-net 队列为 vhost 驱动程序使用 64 KiB 内核内存。
- 如果 **networkInterfaceMultiqueue** 设置为 'true'，启动带有超过 16 个 CPU 的虚拟机会导致没有连接(CNV-16107)。

7.14.17.2. 启用多队列功能

为使用 VirtIO 模型配置的接口启用多队列功能。

流程

1. 在虚拟机的 **VirtualMachine** 清单文件中将 **networkInterfaceMultiqueue** 值设置为 **true** 来启用多队列功能：

```
apiVersion: kubevirt.io/v1
kind: VM
spec:
  domain:
    devices:
      networkInterfaceMultiqueue: true
```

2. 保存 **VirtualMachine** 清单文件以应用更改。

7.15. VM 磁盘

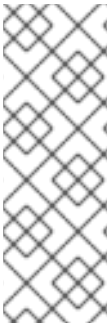
7.15.1. 热插虚拟机磁盘

您可以在不停止虚拟机(VM)或虚拟机实例(VMI)的情况下添加或删除虚拟磁盘。

只有数据卷和持久性卷声明 (PVC) 才能热插和热拔。您无法热插或热拔容器磁盘。

热插磁盘在重新引导后仍会附加到虚拟机。您必须分离磁盘才能从虚拟机中删除它。

您可以使热插磁盘持久保留，使其永久挂载在虚拟机上。



注意

每个虚拟机都有一个 **virtio-scsi** 控制器，以便热插磁盘可以使用 **scsi** 总线。**virtio-scsi** 控制器克服了 **virtio** 的限制，同时保持其性能优势。它高度可扩展，支持超过 400 万个磁盘的热插拔。

常规 **virtio** 不适用于热插磁盘，因为它不可扩展。每个 **virtio** 磁盘都使用虚拟机中的一个有限的 PCI Express (PCIe) 插槽。PCIe 插槽也被其他设备使用，必须提前保留。因此，插槽可能按需提供。

7.15.1.1. 使用 Web 控制台热插和热拔磁盘

您可以使用 OpenShift Container Platform web 控制台在虚拟机运行时将其附加到虚拟机 (VM) 来热插磁盘。

热插磁盘会附加到虚拟机，直到您拔出为止。

您可以使热插磁盘持久保留，使其永久挂载在虚拟机上。

先决条件

- 您必须至少有一个数据卷或持久性卷声明（PVC）可用于热插。

流程

1. 在 web 控制台中进入到 **Virtualization → VirtualMachines**。
2. 选择一个正在运行的虚拟机来查看其详情。
3. 在 **VirtualMachine 详情**页面中，点 **Configuration → Disks**。

4. 添加热插磁盘：

   a. 点 **Add disk**。

   b. 在 **Add disk (hot plugged)** 窗口中，从 **Source** 列表中选择磁盘，然后点 **Save**。

5. 可选： 热插磁盘：

   a. 点磁盘  旁边的选项菜单，然后选择 **Detach**。

   b. 单击 **Detach**。

6. 可选：使热插磁盘持久：

   a. 点磁盘旁的选项菜单  并选择 **Make persistent**。

   b. 重启虚拟机以应用更改。

7.15.1.2. 使用命令行热插和热拔磁盘

您可以使用命令行在虚拟机 (VM) 运行时热插和热拔磁盘。

您可以使热插磁盘持久保留，使其永久挂载在虚拟机上。

先决条件

- 您必须至少有一个数据卷或持久性卷声明（PVC）可用于热插。

流程

- 运行以下命令来热插磁盘：

```
$ virtctl addvolume <virtual-machine|virtual-machine-instance> \
  --volume-name=<datavolume|PVC> \
  [--persist] [--serial=<label-name>]
```

  - 使用可选 **--persist** 标志，将热插磁盘作为永久挂载的虚拟磁盘添加到虚拟机规格中。停止、重新启动或重新启动虚拟机以永久挂载虚拟磁盘。指定 **--persist** 标志后，您无法再热插或热拔虚拟磁盘。**Persist** 标志适用于虚拟机，不适用于虚拟机实例。

  - 可选 **--serial** 标志允许您添加您选择的字母数字字符串标签。这有助于您识别客户机虚拟机中的热插磁盘。如果没有指定这个选项，则标签默认为热插数据卷或 PVC 的名称。

- 运行以下命令来热拔磁盘：

```
$ virtctl removevolume <virtual-machine|virtual-machine-instance> \
  --volume-name=<datavolume|PVC>
```

7.15.2. 扩展虚拟机磁盘

您可以通过扩展磁盘的持久性卷声明(PVC)来增加虚拟机(VM)磁盘的大小。

如果您的存储供应商不支持卷扩展，您可以通过添加空白数据卷来扩展虚拟机的可用虚拟存储。

您不能缩小虚拟机磁盘的大小。

7.15.2.1. 扩展虚拟机磁盘 PVC

您可以通过扩展磁盘的持久性卷声明(PVC)来增加虚拟机(VM)磁盘的大小。

如果 PVC 使用文件系统卷模式，磁盘镜像文件会扩展到可用大小，同时为文件系统开销保留一些空间。

流程

1. 编辑您要扩展的虚拟机磁盘的 **PersistentVolumeClaim** 清单：

```
$ oc edit pvc <pvc_name>
```

2. 更新磁盘大小：

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: vm-disk-expand
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 3Gi (1)
# ...
```

(1) 指定新磁盘大小。

卷扩展的其他资源

- 在 Windows 中扩展基本卷
- 在 Red Hat Enterprise Linux 中扩展现有文件系统分区而不破坏数据 。
- 在 Red Hat Enterprise Linux 中在线扩展逻辑卷及其文件系统

7.15.2.2. 通过添加空白数据卷来扩展可用虚拟存储

您可以通过添加空白数据卷来扩展虚拟机的可用存储。

先决条件

- 您必须至少有一个持久性卷。

流程

1. 如以下示例所示创建 **DataVolume** 清单：

DataVolume 清单示例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: blank-image-datavolume
spec:
  source:
    blank: {}
  storage:
    resources:
      requests:
        storage: <2Gi> 1
  storageClassName: "<storage_class>" 2
```

1 指定为数据卷请求的可用空间量。

2 可选：如果您没有指定存储类，则会使用默认存储类。

2. 运行以下命令来创建数据卷：

```
$ oc create -f <blank-image-datavolume>.yaml
```

数据卷的其他资源

- 为数据卷配置预分配模式
- 管理数据卷注解

7.15.3. 为虚拟机配置共享卷

您可以配置共享磁盘，以允许多个虚拟机 (VM) 共享相同的底层存储。共享磁盘的卷必须是块模式。

您可以通过将存储公开为以下类型之一来配置磁盘共享：

- 普通虚拟机磁盘
- 具有 iSCSi 连接和原始设备映射的逻辑单元号 (LUN) 磁盘，作为共享卷的 Windows 故障切换集群需要

除了配置磁盘共享外，您还可以为每个普通虚拟机磁盘或 LUN 磁盘设置错误策略。错误策略用于控制，当在磁盘的读或写出现输入/输出错误时，hypervisor 的行为。

7.15.3.1. 使用虚拟机磁盘配置磁盘共享

您可以配置块卷，以便多个虚拟机 (VM) 可以共享存储。

在客户机操作系统中运行的应用程序决定了您必须为虚拟机配置的 storage 选项。类型磁盘的磁盘将卷作为普通 **磁盘** 公开给虚拟机。

您可以为每个磁盘设置错误策略。错误策略用于控制，当在磁盘的读或写出现输入/输出错误时，hypervisor 的行为。默认行为是，停止虚拟机并生成 Kubernetes 事件。

您可以接受默认行为，也可以将错误策略设置为以下选项之一：

- **report**，它将报告虚拟客户机错误。

- **ignore**，忽略错误。没有检测到 Read 或 Write。
- **enospace**，生成一个错误，表示没有足够的磁盘空间。

先决条件

- 如果共享磁盘的虚拟机在不同节点上运行，则卷访问模式必须是 **ReadWriteMany** (RWX)。
  如果共享磁盘的虚拟机在同一节点上运行，则 **ReadWriteOnce** (RWO) 卷访问模式就足够了。
- 存储供应商必须支持所需的 Container Storage Interface (CSI) 驱动程序。

流程

1. 为虚拟机创建 **VirtualMachine** 清单来设置所需的值，如下例所示：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: <vm_name>
spec:
  template:
# ...
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: rootdisk
            errorPolicy: report (1)
            disk1: disk_one (2)
          - disk:
              bus: virtio
            name: cloudinitdisk
            disk2: disk_two
            shareable: true (3)
          interfaces:
          - masquerade: {}
            name: default
```

(1) 标识错误策略。

(2) 将设备标识为磁盘。

(3) 标识共享磁盘。

2. 保存 **VirtualMachine** 清单文件以应用更改。

7.15.3.2. 使用 LUN 配置磁盘共享

要保护虚拟机上的数据不受外部访问，您可以启用 SCSI 持久保留并配置 LUN 支持的虚拟机磁盘在多个虚拟机间共享。启用共享选项允许您针对底层存储使用高级 SCSI 命令，如 Windows 故障转移集群实施所需的高级 SCSI 命令。

当存储卷配置为 **LUN** 磁盘类型时，虚拟机可以使用这个卷作为逻辑单元号 (LUN) 设备。因此，虚拟机可以使用 SCSI 命令部署和管理磁盘。

您可以通过 SCSI 持久保留选项保留 LUN。启用保留：

1. 配置功能门选项
2. 可以激活 LUN 磁盘上的选项，以发出虚拟机所需的 SCSI 设备特定输入和输出控制 (IOCTLs)。

您可以为每个 LUN 磁盘设置错误策略。错误策略用于控制，当在磁盘的读或写出现输入/输出错误时，hypervisor 的行为。默认行为是，停止客户机并生成 Kubernetes 事件。

对于具有 iSCSi 连接和持久保留的 LUN 磁盘，根据 Windows Failover 集群用于共享卷，您可以将错误策略设置为 **report**。

先决条件

- 您必须具有集群管理员特权才能配置功能门选项。
- 如果共享磁盘的虚拟机在不同节点上运行，则卷访问模式必须是 **ReadWriteMany** (RWX)。如果共享磁盘的虚拟机在同一节点上运行，则 **ReadWriteOnce** (RWO) 卷访问模式就足够了。
- 存储供应商必须支持使用 SCSI 协议的 Container Storage Interface (CSI) 驱动程序。
- 如果您是集群管理员，并打算使用 LUN 配置磁盘共享，您必须在 **HyperConverged** 自定义资源 (CR) 上启用集群的功能门。
- 要共享的磁盘必须处于块模式。

流程

1. 编辑或为虚拟机创建 **VirtualMachine** 清单来设置所需的值，如下例所示：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-0
spec:
  template:
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: sata
            name: rootdisk
          - errorPolicy: report (1)
            lun: (2)
              bus: scsi
              reservation: true (3)
            name: na-shared
            serial: shared1234
      volumes:
      - dataVolume:
          name: vm-0
        name: rootdisk
```

```
    - name: na-shared
      persistentVolumeClaim:
        claimName: pvc-na-share
```

1. 标识错误策略。
2. 标识 LUN 磁盘。
3. 标识启用了持久性保留。

2. 保存 **VirtualMachine** 清单文件以应用更改。

7.15.3.2.1. 使用 LUN 和 Web 控制台配置磁盘共享

您可以使用 OpenShift Container Platform Web 控制台使用 LUN 配置磁盘共享。

先决条件

- 集群管理员必须启用 **persistentreservation** 功能门设置。

流程

1. 在 web 控制台中点 **Virtualization → VirtualMachines**。
2. 选择一个虚拟机以打开 **VirtualMachine 详情**页。
3. 展开 **Storage**。
4. 在 **Disks** 选项卡上，点 **Add disk**。
5. 指定 **Name**, **Source**, **Size**, **Interface**, 和 **Storage Class**。
6. 选择 **LUN** 作为**类型**。
7. 选择 **Shared access (RWX)** 作为 **Access Mode**。
8. 选择 **Block** 作为 **卷模式**。
9. 展开 **Advanced Settings**，然后选中这两个复选框。
10. 点击 **Save**。

7.15.3.2.2. 使用 LUN 和命令行配置磁盘共享

您可以使用 LUN 配置磁盘共享。

流程

1. 编辑或为虚拟机创建 **VirtualMachine** 清单来设置所需的值，如下例所示：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-0
spec:
```

```
template:
  spec:
    domain:
      devices:
        disks:
        - disk:
            bus: sata
          name: rootdisk
        - errorPolicy: report
          lun: (1)
            bus: scsi
            reservation: true (2)
          name: na-shared
          serial: shared1234
    volumes:
    - dataVolume:
        name: vm-0
      name: rootdisk
    - name: na-shared
      persistentVolumeClaim:
        claimName: pvc-na-share
```

1. 标识 LUN 磁盘。
2. 标识启用了持久性保留。

2. 保存 **VirtualMachine** 清单文件以应用更改。

7.15.3.3. 启用 PersistentReservation 功能门

您可以启用 SCSI **persistentReservation** 功能门，并允许在多个虚拟机间共享 LUN 支持的块模式虚拟机 (VM) 磁盘。

默认禁用 **persistentReservation** 功能门。您可以使用 Web 控制台或命令行启用 **persistentReservation** 功能门。

先决条件

- 需要集群管理员权限。
- 如果共享磁盘的虚拟机在不同节点上运行，则需要卷访问模式 **ReadWriteMany** (RWX)。如果共享磁盘的虚拟机在同一节点上运行，则 **ReadWriteOnce** (RWO) 卷访问模式就足够了。
- 存储供应商必须支持使用 SCSI 协议的 Container Storage Interface (CSI) 驱动程序。

7.15.3.3.1. 使用 Web 控制台启用 PersistentReservation 功能门

您必须启用 PersistentReservation 功能门，以允许在多个虚拟机间共享 LUN 支持的块模式虚拟机(VM)磁盘。启用功能门需要集群管理员权限。

流程

1. 在 web 控制台中点 **Virtualization → Overview**。

2. 点 **Settings** 选项卡。
3. 选择 **Cluster**。
4. 扩展 **SCSI persistent reservation** 并将 **Enable persistent reservation** 设置为 on。

7.15.3.3.2. 使用命令行启用 PersistentReservation 功能门

您可以使用命令行启用 **persistentReservation** 功能门。启用功能门需要集群管理员权限。

流程

1. 运行以下命令启用 **persistentReservation** 功能门：

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv --type json -p \
'[{"op":"replace","path":"/spec/featureGates/persistentReservation", "value": true}]'
```

其他资源

- 持久性保留帮助程序协议
- Windows 服务器和 Azure Stack HCI 中的故障转移集群

第 8 章 网络

8.1. 网络概述

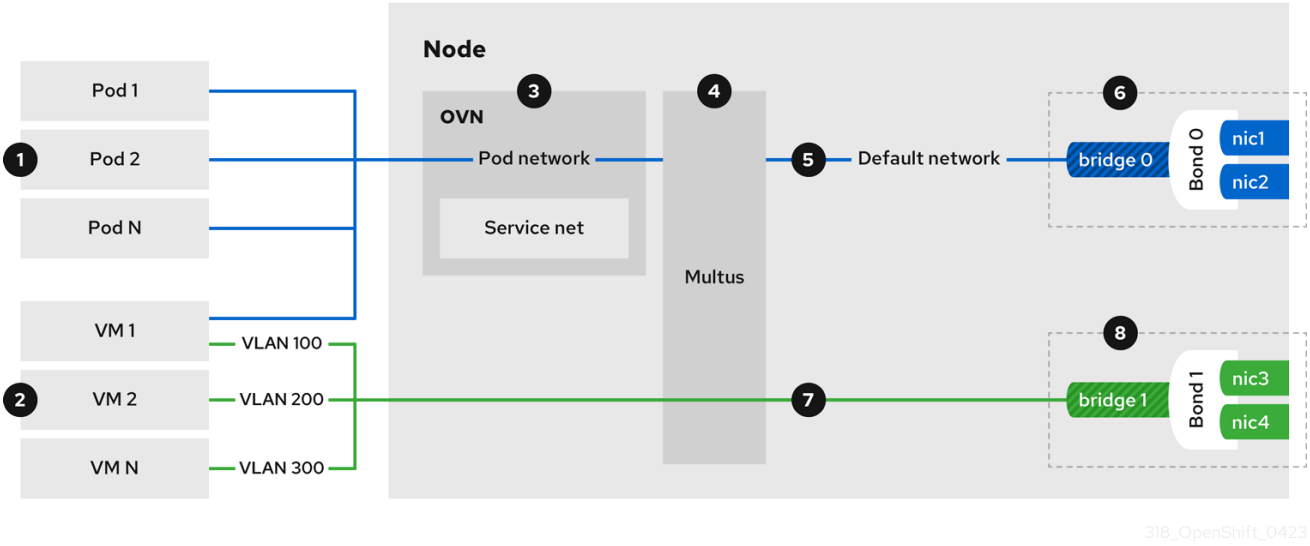
OpenShift Virtualization 使用自定义资源和插件提供高级联网功能。虚拟机(VM)与 OpenShift Container Platform 网络及其生态系统集成。

注意

您无法在单堆栈 IPv6 集群上运行 OpenShift Virtualization。

下图演示了 OpenShift Virtualization 的典型网络设置。也可以进行其他配置。

图 8.1. OpenShift Virtualization 网络概述



1. Pod 和虚拟机在同一网络基础架构上运行，允许您轻松连接容器化和虚拟化的工作负载。
2. 您可以将虚拟机连接到默认 pod 网络和任意数量的二级网络。
3. 默认 pod 网络提供其所有成员、服务抽象、IP 管理、微分段和其他功能之间的连接。
4. Multus 是一个 "meta" CNI 插件，它允许 pod 或虚拟机使用其他兼容 CNI 插件连接到其他网络接口。
5. 默认 pod 网络是基于覆盖的，通过底层机器网络进行隧道连接。
6. 机器网络可以在一组选定的网络接口控制器(NIC)上定义。
7. 辅助虚拟机网络通常直接桥接到物理网络，且没有 VLAN 封装。
8. 二级虚拟机网络可以在专用的 NIC 集合上定义，如图 1 所示，也可以使用机器网络。

8.1.1. OpenShift Virtualization 术语表

以下是整个 OpenShift Virtualization 文档中使用的术语：

Container Network Interface (CNI)

一个 Cloud Native Computing Foundation 项目，侧重容器网络连接。OpenShift Virtualization 使用 CNI 插件基于基本 Kubernetes 网络功能进行构建。

Multus

一个"meta"CNI 插件，支持多个 CNI 共存，以便 pod 或虚拟机可使用其所需的接口。

自定义资源定义(CRD)

一个 Kubernetes API 资源，用于定义自定义资源，或使用 CRD API 资源定义的对象。

网络附加定义(NAD)

由 Multus 项目引入的 CRD，允许您将 Pod、虚拟机和虚拟机实例附加到一个或多个网络。

节点网络配置策略(NNCP)

nmstate 项目引入的 CRD，描述节点上请求的网络配置。您可以通过将 **NodeNetworkConfigurationPolicy**清单应用到集群来更新节点网络配置，包括添加和删除网络接口。

8.1.2. 使用默认 pod 网络

将虚拟机连接到默认 pod 网络

每个虚拟机默认连接到默认的内部 pod 网络。您可以通过编辑虚拟机规格来添加或删除网络接口。

将虚拟机作为服务公开

您可以通过创建 **Service** 对象在集群内或集群外公开虚拟机。对于内部集群，您可以使用 MetalLB Operator 配置负载均衡服务。您可以使用 OpenShift Container Platform Web 控制台或 CLI 安装 MetalLB Operator。

8.1.3. 配置虚拟机二级网络接口

您可以使用 Linux 网桥、SR-IOV 和 OVN-Kubernetes CNI 插件将虚拟机连接到二级网络。您可以在虚拟机规格中列出多个二级网络和接口。连接到二级网络接口时，不需要在虚拟机规格中指定主 pod 网络。

将虚拟机连接到 Linux 网桥网络

安装 Kubernetes NMState Operator，为您的二级网络配置 Linux 网桥、VLAN 和绑定。
您可以通过执行以下步骤来创建 Linux 网桥网络并将虚拟机附加到网络：

1. 通过创建 **NodeNetworkConfigurationPolicy** 自定义资源定义(CRD) 来配置 Linux 网桥网络设备。
2. 通过创建 **NetworkAttachmentDefinition** CRD 来配置 Linux 网桥网络。
3. 通过在虚拟机配置中包含网络详情，将虚拟机连接到 Linux 网桥网络。

将虚拟机连接到 SR-IOV 网络

对于需要高带宽或低延迟的应用程序，您可以使用单根 I/O 虚拟化 (SR-IOV) 网络设备，并在裸机或 Red Hat OpenStack Platform (RHOSP) 基础架构上安装额外网络。
您必须在集群中安装 SR-IOV Network Operator，以管理 SR-IOV 网络设备和网络附加。

您可以通过执行以下步骤将虚拟机连接到 SR-IOV 网络：

1. 通过创建一个 **SriovNetworkNodePolicy** CRD 配置一个 SR-IOV 网络设备。

2. 通过创建一个 **SriovNetwork** 对象来配置 SR-IOV 网络。

3. 通过在虚拟机配置中包含网络详情，将虚拟机连接到 SR-IOV 网络。

将虚拟机连接到 OVN-Kubernetes 二级网络

您可以将虚拟机连接到 Open Virtual Network (OVN) -Kubernetes 二级网络。OpenShift Virtualization 支持 OVN-Kubernetes 的第 2 层和 localnet 拓扑。

- 第 2 层拓扑通过集群范围的逻辑交换机连接工作负载。OVN-Kubernetes Container Network Interface (CNI) 插件使用 Geneve (Generic Network Virtualization Encapsulation) 协议在节点间创建覆盖网络。您可以使用此覆盖网络在不同的节点上连接虚拟机，而无需配置任何其他物理网络基础架构。

- localnet 拓扑将二级网络连接到物理网络。这可让 east-west 集群流量并访问在集群外运行的服务，但它需要在集群节点上配置底层 Open vSwitch (OVS) 系统。

要配置 OVN-Kubernetes 二级网络并将虚拟机附加到该网络，请执行以下步骤：

1. 通过创建网络附加定义(NAD)来配置 OVN-Kubernetes 二级网络。

注意

对于 localnet 拓扑，您必须在创建 NAD 前创建一个 **NodeNetworkConfigurationPolicy** 对象来配置 OVS 网桥。

2. 通过在虚拟机规格中添加网络详情，将虚拟机连接到 OVN-Kubernetes 二级网络。

8.1.3.1. 比较 Linux 网桥 CNI 和 OVN-Kubernetes localnet 拓扑

下表提供了使用 Linux bridge CNI 和 OVN-Kubernetes 插件 localnet 拓扑时可用的功能比较：

表 8.1. Linux bridge CNI 与 OVN-Kubernetes localnet 拓扑的比较

功能	在 Linux 网桥 CNI 上提供	OVN-Kubernetes localnet 上提供
第 2 层访问 underlay 原生网络	仅在二级网络接口控制器(NIC)上。	是
Layer 2 访问底层 VLAN	是	是
网络策略	否	是
受管 IP 池	否	否
MAC spoof 过滤	是	是

热插二级网络接口

您可以在不停止虚拟机的情况下添加或删除二级网络接口。OpenShift Virtualization 支持热插和热拔使用 VirtIO 设备驱动程序的 Linux 网桥接口。

在 SR-IOV 中使用 DPDK

Data Plane Development Kit (DPDK) 提供了一组库和驱动程序，用于快速数据包处理。您可以配置集群和虚拟机，以通过 SR-IOV 网络运行 DPDK 工作负载。

为实时迁移配置专用网络

您可以为实时迁移配置专用 Multus 网络。专用的网络可最小化实时迁移期间对租户工作负载的网络饱和影响。

使用集群 FQDN 访问虚拟机

您可以使用其完全限定域名 (FQDN) 访问从集群外部附加到二级网络接口的虚拟机。

配置和查看 IP 地址

您可以在创建虚拟机时配置二级网络接口的 IP 地址。IP 地址使用 cloud-init 置备。您可以使用 OpenShift Container Platform Web 控制台或命令行查看虚拟机的 IP 地址。QEMU 客户机代理收集网络信息。

8.1.4. 与 OpenShift Service Mesh 集成

将虚拟机连接到服务网格

OpenShift Virtualization 与 OpenShift Service Mesh 集成。您可以监控、视觉化和控制 pod 和虚拟机之间的流量。

8.1.5. 管理 MAC 地址池

为网络接口管理 MAC 地址池

KubeMacPool 组件从共享 MAC 地址池为虚拟机网络接口分配 MAC 地址。这样可确保为每个网络接口分配唯一的 MAC 地址。从该虚拟机创建的虚拟机实例在重启后保留分配的 MAC 地址。

8.1.6. 配置 SSH 访问

配置对虚拟机的 SSH 访问

您可以使用以下方法配置到虚拟机的 SSH 访问：

- **virtctl ssh 命令**
  您可以创建一个 SSH 密钥对，将公钥添加到虚拟机，并使用私钥运行 **virtctl ssh** 命令连接到虚拟机。

  您可以在运行时将公共 SSH 密钥添加到 Red Hat Enterprise Linux (RHEL) 9 虚拟机，或第一次引导到使用 cloud-init 数据源配置的客户机操作系统的虚拟机。

- **virtctl port-forward 命令**
  您可以将 **virtctl port-foward** 命令添加到 **.ssh/config** 文件中，并使用 OpenSSH 连接到虚拟机。

- 服务
  您可以创建一个服务，将服务与虚拟机关联，并连接到该服务公开的 IP 地址和端口。

- 二级网络
  您可以配置二级网络，将虚拟机附加到二级网络接口，并连接到其分配的 IP 地址。

8.2. 将虚拟机连接到默认 POD 网络

您可以通过将其网络接口配置为使用 **masquerade** 绑定模式，将虚拟机连接到默认的内部 pod 网络。

注意

在实时迁移过程中，通过网络接口到默认 pod 网络的流量会中断。

8.2.1. 从命令行配置伪装模式

您可以使用伪装模式将虚拟机的外发流量隐藏在 pod IP 地址后。伪装模式使用网络地址转换 (NAT) 来通过 Linux 网桥将虚拟机连接至 pod 网络后端。

启用伪装模式，并通过编辑虚拟机配置文件让流量进入虚拟机。

先决条件

- 虚拟机必须配置为使用 DHCP 来获取 IPv4 地址。

流程

1. 编辑虚拟机配置文件的 **interfaces** 规格：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - name: default
              masquerade: {} 1
              ports: 2
                - port: 80
# ...
      networks:
      - name: default
        pod: {}
```

1 使用伪装模式进行连接。

2 可选：列出您要从虚拟机公开的端口，每个都由 **port** 字段指定。**port** 值必须是 0 到 65536 之间的数字。如果没有使用 **port** 数组，则有效范围内的所有端口都开放给传入流量。在本例中，端口 **80** 上允许传入的流量。

注意

端口 49152 和 49153 保留供 libvirt 平台使用，这些端口的所有其他传入流量将被丢弃。

2. 创建虚拟机：

```
$ oc create -f <vm-name>.yaml
```

8.2.2. 使用双栈（IPv4 和 IPv6）配置伪装模式

您可以使用 cloud-init 将新虚拟机配置为在默认 pod 网络上同时使用 IPv6 和 IPv4。

虚拟机实例配置中的 **Network.pod.vmIPv6NetworkCIDR** 字段决定虚拟机的静态 IPv6 地址和网关 IP 地址。virt-launcher Pod 使用它们将 IPv6 流量路由到虚拟机，而不在外部使用。**Network.pod.vmIPv6NetworkCIDR** 字段在无类别域间路由(CIDR)标记中指定一个 IPv6 地址块。默认值为 **fd10:0:2::2/120**。您可以根据网络要求编辑这个值。

当虚拟机运行时，虚拟机的传入和传出流量将路由到 IPv4 地址和 virt-launcher Pod 的唯一 IPv6 地址。virt-launcher pod 随后将 IPv4 流量路由到虚拟机的 DHCP 地址，并将 IPv6 流量路由到虚拟机的静态设置 IPv6 地址。

先决条件

- OpenShift Container Platform 集群必须使用为双栈配置的 OVN-Kubernetes Container Network Interface (CNI) 网络插件。

流程

1. 在新的虚拟机配置中，包含具有 **masquerade** 的接口，并使用 cloud-init 配置 IPv6 地址和默认网关。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm-ipv6
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - name: default
              masquerade: {} 1
              ports:
                - port: 80 2
# ...
      networks:
      - name: default
        pod: {}
      volumes:
      - cloudInitNoCloud:
          networkData: |
            version: 2
            ethernets:
              eth0:
                dhcp4: true
                addresses: [ fd10:0:2::2/120 ] 3
                gateway6: fd10:0:2::1 4
```

1. 使用伪装模式进行连接。
2. 允许虚拟机上端口 80 上的传入流量。

3 由虚拟机实例配置中的 **Network.pod.vmIPv6NetworkCIDR** 字段确定的静态 IPv6 地址。默认值为 **fd10:0:2::2/120**。

4 网关 IP 地址由虚拟机实例配置中的 **Network.pod.vmIPv6NetworkCIDR** 字段决定。默认值为 **fd10:0:2::1**。

2. 在命名空间中创建虚拟机：

```
$ oc create -f example-vm-ipv6.yaml
```

验证

- 要验证 IPv6 是否已配置，启动虚拟机并查看虚拟机实例的接口状态，以确保它具有 IPv6 地址：

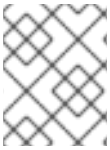
```
$ oc get vmi <vmi-name> -o jsonpath="{.status.interfaces[*].ipAddresses}"
```

8.2.3. 关于巨型帧支持

使用 OVN-Kubernetes CNI 插件时，您可以在默认 pod 网络上连接的两个虚拟机 (VM) 之间发送未分片的巨型帧数据包。巨型帧有一个大于 1500 字节的最大传输单元 (MTU) 值。

虚拟机自动获得集群网络的 MTU 值，由集群管理员设置，如下所示：

- **libvirt** ：如果客户机操作系统具有 VirtIO 驱动程序的最新版本，该驱动程序可通过模拟设备中的 Peripheral Component Interconnect (PCI) 配置寄存器来解释传入的数据。
- DHCP ：如果客户机 DHCP 客户端可以从 DHCP 服务器响应中读取 MTU 值。



注意

对于没有 VirtIO 驱动程序的 Windows 虚拟机，您必须使用 **netsh** 或类似的工具手动设置 MTU。这是因为 Windows DHCP 客户端没有读取 MTU 值。

8.2.4. 其他资源

- 更改集群网络的 MTU
- 为您的网络优化 MTU

8.3. 使用服务公开虚拟机

您可以通过创建 **Service** 对象在集群内或集群外公开虚拟机。

8.3.1. 关于服务

Kubernetes 服务将客户端的网络访问权限公开给一组容器集上运行的应用。服务在 **NodePort** 和 **LoadBalancer** 类型方面提供抽象、负载均衡以及暴露于外部世界。

ClusterIP

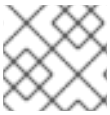
在内部 IP 地址上公开服务，并将 DNS 名称公开给集群中的其他应用程序。单个服务可映射到多个虚拟机。当客户端尝试连接到服务时，客户端请求会在可用后端之间平衡负载。**ClusterIP** 是默认的服务类型。

NodePort

在集群中每个所选节点的同一端口上公开该服务。**NodePort** 使端口可从集群外部访问，只要节点本身可以被客户端外部访问。

LoadBalancer

在当前云中创建外部负载均衡器（如果支持），并为该服务分配固定的外部 IP 地址。



注意

对于内部集群，您可以通过部署 MetalLB Operator 来配置负载均衡服务。

其他资源

- 安装 MetalLB Operator
- 将服务配置为使用 MetalLB

8.3.2. 双栈支持

如果为集群启用了 IPv4 和 IPv6 双栈网络，您可以通过定义 **Service** 对象中的 **spec.ipFamilyPolicy** 和 **spec.ipFamilies** 字段来创建使用 IPv4、IPv6 或两者的服务。

spec.ipFamilyPolicy 字段可以设置为以下值之一：

SingleStack

control plane 根据配置的第一个服务集群 IP 范围为该服务分配集群 IP 地址。

PreferDualStack

control plane 为配置了双栈的集群中的服务分配 IPv4 和 IPv6 集群 IP 地址。

RequireDualStack

对于没有启用双栈网络的集群，这个选项会失败。对于配置了双栈的集群，其行为与将值设置为 **PreferDualStack** 时相同。control plane 从 IPv4 和 IPv6 地址范围分配集群 IP 地址。

您可以通过将 **spec.ipFamilies** 字段设置为以下数组值之一来定义用于单堆栈的 IP 系列，或者定义双栈 IP 系列的顺序：

- **[IPv4]**
- **[IPv6]**
- **[IPv4, IPv6]**
- **[IPv6, IPv4]**

8.3.3. 使用命令行创建服务

您可以使用命令行创建服务并将其与虚拟机 (VM) 关联。

先决条件

- 您已将集群网络配置为支持该服务。

流程

1. 编辑 **VirtualMachine** 清单，为创建服务添加标签：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  running: false
  template:
    metadata:
      labels:
        special: key (1)
# ...
```

(1) 在 **spec.template.metadata.labels** 小节中添加 **special: key**。

注意

虚拟机上的标签会传递到 pod。**special: key** 标签必须与 **Service** 清单的 **spec.selector** 属性中的标签匹配。

2. 保存 **VirtualMachine** 清单文件以应用更改。

3. 创建 **Service** 清单以公开虚拟机：

```
apiVersion: v1
kind: Service
metadata:
  name: example-service
  namespace: example-namespace
spec:
# ...
  selector:
    special: key (1)
  type: NodePort (2)
  ports: (3)
    protocol: TCP
    port: 80
    targetPort: 9376
    nodePort: 30000
```

(1) 指定添加到 **VirtualMachine** 清单的 **spec.template.metadata.labels** 小节中的标签。

(2) 指定 **ClusterIP**、**NodePort** 或 **LoadBalancer**。

(3) 指定您要从虚拟机公开的网络端口和协议集合。

4. 保存 **Service** 清单文件。

5. 运行以下命令来创建服务：

```
$ oc create -f example-service.yaml
```

6. 重启虚拟机以应用更改。

验证

- 查询 **Service** 对象以验证它是否可用：

```
$ oc get service -n example-namespace
```

8.3.4. 其他资源

- 使用 NodePort 配置集群入口流量
- 使用负载均衡器配置集群入口流量

8.4. 使用其内部 FQDN 访问虚拟机

现在，您可以使用无头服务，访问连接到一个具有稳定 FQDN 的默认内部 pod 网络的虚拟机。

Kubernetes *无头服务* 是一种形式的服务，它不会分配集群 IP 地址来代表一组 pod。对于无头服务，它不是为服务提供一个虚拟 IP 地址，而是为与服务关联的每个 pod 创建一个 DNS 记录。您可以通过 FQDN 来公开虚拟机，而无需公开特定的 TCP 或 UDP 端口。

重要

如果使用 OpenShift Container Platform Web 控制台创建虚拟机，您可以在 **VirtualMachine 详情页的 Overview** 标签页中找到它在 **Network** 标题中列出的内部 FQDN。有关连接到虚拟机的更多信息，请参阅使用其内部 FQDN 连接到虚拟机。

8.4.1. 使用 CLI 在项目中创建无头服务

要在命名空间中创建无头服务，请将 **clusterIP: None** 参数添加到服务 YAML 定义中。

先决条件

- 已安装 OpenShift CLI（**oc**）。

流程

1. 创建 **Service** 清单以公开虚拟机，如下例所示：

```
apiVersion: v1
kind: Service
metadata:
  name: mysubdomain (1)
spec:
  selector:
    expose: me (2)
  clusterIP: None (3)
  ports: (4)
  - protocol: TCP
    port: 1234
    targetPort: 1234
```

-

1. 服务的名称。这必须与 **VirtualMachine** 清单文件中的 **spec.subdomain** 属性匹配。
2. 此服务选择器必须与 **VirtualMachine** 清单文件中的 **expose:me** 标签匹配。
3. 指定无头服务。
4. 服务公开的端口列表。您必须至少定义一个端口。这可以是任意值，因为它不会影响无头服务。

2. 保存 **Service** 清单文件。
3. 运行以下命令来创建服务：

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

8.4.2. 使用 CLI 将虚拟机映射到无头服务

要使用其内部完全限定域名 (FQDN) 从集群中连接到虚拟机 (VM)，您必须首先将虚拟机映射到无头服务。在虚拟机配置文件中设置 **spec.hostname** 和 **spec.subdomain** 参数。

如果一个无头服务的名称与子域匹配，则会为虚拟机创建一个唯一的 DNS 记录，格式为 **<vm.spec.hostname>.<vm.spec.subdomain>.<vm.metadata.namespace>.svc.cluster.local**。

流程

1. 运行以下命令，编辑 **VirtualMachine** 清单以添加服务选择器标签和子域：

```
$ oc edit vm <vm_name>
```

VirtualMachine 清单文件示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-fedora
spec:
  template:
    metadata:
      labels:
        expose: me 1
    spec:
      hostname: "myvm" 2
      subdomain: "mysubdomain" 3
# ...
```

1. **expose:me** 标签必须与之前创建的 **Service** 清单的 **spec.selector** 属性匹配。
2. 如果没有指定此属性，则生成的 DNS A 记录的格式为 **<vm.metadata.name>.<vm.spec.subdomain>.<vm.metadata.namespace>.svc.cluster.local**。
3. **spec.subdomain** 属性必须与 **Service** 对象的 **metadata.name** 值匹配。

2. 保存更改并退出编辑器。
3. 重启虚拟机以应用更改。

8.4.3. 使用其内部 FQDN 连接到虚拟机

您可以使用其内部完全限定域名 (FQDN) 连接到虚拟机 (VM)。

先决条件

- 已安装 **virtctl** 工具。
- 您已从 web 控制台或将虚拟机映射到无头服务来识别虚拟机的内部 FQDN。内部 FQDN 的格式是 **<vm.spec.hostname>.<vm.spec.subdomain>.<vm.metadata.namespace>.svc.cluster.local**。

流程

1. 输入以下命令连接到虚拟机控制台：

```
$ virtctl console vm-fedora
```

2. 要使用请求的 FQDN 连接到虚拟机，请运行以下命令：

```
$ ping myvm.mysubdomain.<namespace>.svc.cluster.local
```

输出示例

```
PING myvm.mysubdomain.default.svc.cluster.local (10.244.0.57) 56(84) bytes of data.
64 bytes from myvm.mysubdomain.default.svc.cluster.local (10.244.0.57): icmp_seq=1 ttl=64
time=0.029 ms
```

在前面的示例中，**myvm.mysubdomain.default.svc.cluster.local** 的 DNS 记录指向 **10.244.0.57**，这是目前分配给虚拟机的集群 IP 地址。

8.4.4. 其他资源

- 使用一个服务公开虚拟机

8.5. 将虚拟机连接到 LINUX 网桥网络

默认情况下，OpenShift Virtualization 安装了一个内部 pod 网络。

您可以通过执行以下步骤来创建 Linux 网桥网络，并将虚拟机 (VM) 附加到网络：

1. 创建 Linux 网桥节点网络配置策略 (NNCP)。
2. 使用 Web 控制台或命令行创建 Linux 网桥网络附加定义(NAD)。
3. 使用 Web 控制台或命令行配置虚拟机，以识别 NAD。

8.5.1. 创建 Linux 网桥 NNCP

您可以为 Linux 网桥网络创建一个 **NodeNetworkConfigurationPolicy** (NNCP) 清单。

先决条件

- 已安装 Kubernetes NMState Operator。

流程

- 创建 **NodeNetworkConfigurationPolicy** 清单。本例包含示例值，您必须替换为您自己的信息。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy (1)
spec:
  desiredState:
    interfaces:
      - name: br1 (2)
        description: Linux bridge with eth1 as a port (3)
        type: linux-bridge (4)
        state: up (5)
        ipv4:
          enabled: false (6)
        bridge:
          options:
            stp:
              enabled: false (7)
          port:
            - name: eth1 (8)
```

(1) 策略的名称。

(2) 接口的名称。

(3) 可选：接口人类可读的接口描述。

(4) 接口的类型。这个示例会创建一个桥接。

(5) 创建后接口的请求状态。

(6) 在这个示例中禁用 IPv4。

(7) 在这个示例中禁用 STP。

(8) 网桥附加到的节点 NIC。

8.5.2. 创建 Linux 网桥 NAD

您可以使用 OpenShift Container Platform Web 控制台或命令行创建 Linux 网桥网络附加定义 (NAD)。

8.5.2.1. 使用 Web 控制台创建 Linux 网桥 NAD

您可以创建一个网络附加定义(NAD)来使用 OpenShift Container Platform web 控制台为 Pod 和虚拟机提供第 2 层网络。

Linux 网桥网络附加定义是将虚拟机连接至 VLAN 的最有效方法。

警告

不支持在虚拟机的网络附加定义中配置 IP 地址管理(IPAM)。

流程

1. 在 Web 控制台中，点 **Networking → NetworkAttachmentDefinitions**。
2. 点 **Create Network Attachment Definition**。

注意

网络附加定义必须与 pod 或虚拟机位于同一个命名空间中。

3. 输入唯一 **Name** 和可选 **Description**。
4. 从 **Network Type** 列表中选择 **CNV Linux 网桥**。
5. 在 **Bridge Name** 字段输入网桥名称。
6. 可选：如果资源配置了 VLAN ID，请在 **VLAN Tag Number** 字段中输入 ID 号。
7. 可选： 选择 **MAC Spoof Check** 来启用 MAC spoof 过滤。此功能只允许单个 MAC 地址退出 pod，从而可以防止使用 MAC 欺骗进行的安全攻击。
8. 点 **Create**。

8.5.2.2. 使用命令行创建 Linux 网桥 NAD

您可以创建一个网络附加定义 (NAD) 来使用命令行为 pod 和虚拟机 (VM) 提供第 2 层网络。

NAD 和虚拟机必须位于同一命名空间中。

警告

不支持在虚拟机的网络附加定义中配置 IP 地址管理(IPAM)。

先决条件

- 节点必须支持 nftables，必须部署 **nft** 二进制文件才能启用 MAC 欺骗检查。

流程

1. 将虚拟机添加到 **NetworkAttachmentDefinition** 配置中，如下例所示：

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bridge-network (1)
  annotations:
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/bridge-interface (2)
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "bridge-network", (3)
    "type": "cnv-bridge", (4)
    "bridge": "bridge-interface", (5)
    "macspoofchk": false, (6)
    "vlan": 100, (7)
    "disableContainerInterface": "true",
    "preserveDefaultVlan": false (8)
  }'
```

(1) **NetworkAttachmentDefinition** 对象的名称。

(2) 可选：为节点选择注解键值对，其中 **bridge-interface** 必须与某些节点上配置的桥接名称匹配。如果在网络附加定义中添加此注解，您的虚拟机实例将仅在连接 **bridge-interface** 网桥的节点中运行。

(3) 配置的名称。建议您将配置名称与网络附加定义的 **name** 值匹配。

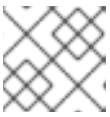
(4) 为这个网络附加定义的 Container Network Interface（CNI）插件的实际名称。不要更改此字段，除非要使用不同的 CNI。

(5) 节点上配置的 Linux 网桥名称。

(6) 可选：启用 MAC 欺骗检查的标记。当设置为 **true** 时，您无法更改 pod 或客户机接口的 MAC 地址。此属性只允许单个 MAC 地址退出 pod，从而可防止 MAC 欺骗攻击。

(7) 可选： VLAN 标签。节点网络配置策略不需要额外的 VLAN 配置。

(8) 可选：指示虚拟机是否通过默认 VLAN 连接到网桥。默认值为 **true**。



注意

Linux 网桥网络附加定义是将虚拟机连接至 VLAN 的最有效方法。

2. 创建网络附加定义：

```
$ oc create -f network-attachment-definition.yaml (1)
```

(1) 其中 **network-attachment-definition.yaml** 是网络附加定义清单的文件名。

验证

- 运行以下命令验证网络附加定义是否已创建：

```
$ oc get network-attachment-definition bridge-network
```

8.5.3. 配置虚拟机网络接口

您可以使用 OpenShift Container Platform web 控制台或命令行配置虚拟机 (VM) 网络接口。

8.5.3.1. 使用 Web 控制台配置虚拟机网络接口

您可以使用 OpenShift Container Platform Web 控制台为虚拟机配置网络接口。

先决条件

- 为网络创建了网络附加定义。

流程

1. 进入到 **Virtualization → VirtualMachines**。
2. 点虚拟机查看 **VirtualMachine 详情**页。
3. 在 **Configuration** 选项卡上，点 **Network interfaces** 选项卡。
4. 点 **Add network interface**。
5. 输入接口名称，然后从 **Network** 列表中选择网络附加定义。
6. 点击 **Save**。
7. 重启虚拟机以应用更改。

网络字段

名称	描述
Name	网络接口控制器的名称。
model	指明网络接口控制器的型号。支持的值有 **e1000e** 和 **virtio**。
网络	可用网络附加定义的列表。
类型	可用绑定方法列表。选择适合网络接口的绑定方法： • 默认 pod 网络：**masquerade** • Linux 网桥网络：**bridge** • SR-IOV 网络： **SR-IOV**

名称	描述
MAC 地址	网络接口控制器的 MAC 地址。如果没有指定 MAC 地址，则会自动分配一个。

8.5.3.2. 使用命令行配置虚拟机网络接口

您可以使用命令行为桥接网络配置虚拟机 (VM) 网络接口。

先决条件

- 在编辑配置前关闭虚拟机。如果编辑正在运行的虚拟机，您必须重启虚拟机才能使更改生效。

流程

1. 在虚拟机配置中添加网桥接口和网络附加定义，如下例所示：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - bridge: {}
              name: bridge-net 1
# ...
      networks:
        - name: bridge-net 2
          multus:
            networkName: a-bridge-network 3
```

1. 网桥接口的名称。
2. 网络的名称。这个值必须与对应的 **spec.template.spec.domain.devices.interfaces** 条目的 **name** 值匹配。
3. 网络附加定义的名称。

2. 应用配置：

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

3. 可选：如果编辑了正在运行的虚拟机，您必须重启它才能使更改生效。

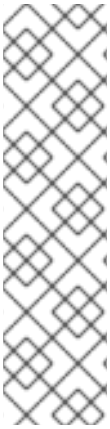
8.6. 将虚拟机连接到 SR-IOV 网络

您可以通过执行以下步骤将虚拟机 (VM) 连接到单根 I/O 虚拟化 (SR-IOV) 网络：

- 配置 SR-IOV 网络设备
- 配置 SR-IOV 网络
- 将虚拟机连接到 SR-IOV 网络

8.6.1. 配置 SR-IOV 网络设备

SR-IOV Network Operator 把 **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CRD 添加到 OpenShift Container Platform。您可以通过创建一个 SriovNetworkNodePolicy 自定义资源 (CR) 来配置 SR-IOV 网络设备。



注意

在应用由 **SriovNetworkNodePolicy** 对象中指定的配置时，SR-IOV Operator 可能会排空节点，并在某些情况下会重启节点。仅在以下情况下重启：

- 使用 Mellanox NIC (**mlx5** 驱动程序)时，当虚拟功能(VF)数量增加时，节点重启都会在物理功能(PF)上增加。
- 使用 Intel NIC 时，只有在内核参数不包含 **intel_iommu=on** 和 **iommu=pt** 时，才会重启。

它可能需要几分钟时间来应用配置更改。

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。
- 已安装 SR-IOV Network Operator。
- 集群中有足够的可用节点，用于处理从排空节点中驱除的工作负载。
- 您还没有为 SR-IOV 网络设备配置选择任何 control plane 节点。

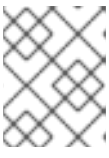
流程

1. 创建一个 **SriovNetworkNodePolicy** 对象，然后在 **<name>-sriov-node-network.yaml** 文件中保存 YAML。使用配置的实际名称替换 **<name>**。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" 4
  priority: <priority> 5
  mtu: <mtu> 6
  numVfs: <num> 7
```

```
  nicSelector: 8
    vendor: "<vendor_code>" 9
    deviceID: "<device_id>" 10
    pfNames: ["<pf_name>", ...] 11
    rootDevices: ["<pci_bus_id>", "..."] 12
  deviceType: vfio-pci 13
  isRdma: false 14
```

1. 为 CR 对象指定一个名称。
2. 指定 SR-IOV Operator 安装到的命名空间。
3. 指定 SR-IOV 设备插件的资源名称。您可以为一个资源名称创建多个 **SriovNetworkNodePolicy** 对象。
4. 指定节点选择器来选择要配置哪些节点。只有所选节点上的 SR-IOV 网络设备才会被配置。SR-IOV Container Network Interface（CNI）插件和设备插件仅在所选节点上部署。
5. 可选：指定一个 **0** 到 **99** 之间的整数。较小的数值具有较高的优先权，优先级 **10** 高于优先级 **99**。默认值为 **99**。
6. 可选：为虚拟功能（VF）的最大传输单位 (MTU) 指定一个值。最大 MTU 值可能因不同的 NIC 型号而有所不同。
7. 为 SR-IOV 物理网络设备指定要创建的虚拟功能 (VF) 的数量。对于 Intel 网络接口控制器（NIC），VF 的数量不能超过该设备支持的 VF 总数。对于 Mellanox NIC，VF 的数量不能超过 **127**。
8. **nicSelector** 映射为 Operator 选择要配置的以太网设备。您不需要为所有参数指定值。建议您以足够的准确度来识别以太网适配器，以便尽量减小意外选择其他以太网设备的可能性。如果指定了 **rootDevices**，则必须同时为 **vendor**、**deviceID** 或 **pfNames** 指定一个值。如果同时指定了 **pfNames** 和 **rootDevices**，请确保它们指向同一个设备。
9. 可选：指定 SR-IOV 网络设备的厂商十六进制代码。允许的值只能是 **8086** 或 **15b3**。
10. 可选：指定 SR-IOV 网络设备的设备十六进制代码。允许的值只能是 **158b**、**1015**、**1017**。
11. 可选：参数接受包括以太网设备的一个或多个物理功能 (PF) 的数组。
12. 参数接受一个包括一个或多个 PCI 总线地址，用于以太网设备的物理功能的数组。使用以下格式提供地址: **0000:02:00.1**。
13. OpenShift Virtualization 中的虚拟功能需要 **vfio-pci** 驱动程序类型。
14. 可选：指定是否启用远程直接访问（RDMA）模式。对于 Mellanox 卡，请将 **isRdma** 设置为 **false**。默认值为 **false**。



注意

如果将 **RDMA** 标记设定为 **true**，您可以继续使用启用了 RDMA 的 VF 作为普通网络设备。设备可在其中的一个模式中使用。

2. 可选：将 SR-IOV 功能的集群节点标记为 **SriovNetworkNodePolicy.Spec.NodeSelector**（如果它们还没有标记）。有关标记节点的更多信息，请参阅"了解如何更新节点上的标签"。

3. 创建 **SriovNetworkNodePolicy** 对象：

```
$ oc create -f <name>-sriov-node-network.yaml
```

其中 **<name>** 指定这个配置的名称。

在应用配置更新后，**sriov-network-operator** 命名空间中的所有 Pod 都会变为 **Running** 状态。

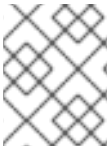
4. 要验证是否已配置了 SR-IOV 网络设备，请输入以下命令。将 **<node_name>** 替换为带有您刚才配置的 SR-IOV 网络设备的节点名称。

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o jsonpath='{.status.syncStatus}'
```

8.6.2. 配置 SR-IOV 额外网络

您可以通过创建一个 **SriovNetwork** 对象来配置使用 SR-IOV 硬件的额外网络。

创建 **SriovNetwork** 对象时，SR-IOV Network Operator 会自动创建一个 **NetworkAttachmentDefinition** 对象。



注意

如果一个 **SriovNetwork** 对象已被附加到状态为 **running** 的 Pod 或虚拟机上，则不能修改或删除它。

先决条件

- 安装 OpenShift CLI（**oc**）。
- 以具有 **cluster-admin** 特权的用户身份登录。

流程

1. 创建以下 **SriovNetwork** 对象，然后在 **<name>-sriov-network.yaml** 文件中保存 YAML。用这个额外网络的名称替换 **<name>** 。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  networkNamespace: <target_namespace> 4
  vlan: <vlan> 5
  spoofChk: "<spoof_check>" 6
  linkState: <link_state> 7
  maxTxRate: <max_tx_rate> 8
  minTxRate: <min_rx_rate> 9
  vlanQoS: <vlan_qos> 10
  trust: "<trust_vf>" 11
  capabilities: <capabilities> 12
```

-

1. 将 **<name>** 替换为对象的名称。SR-IOV Network Operator 创建一个名称相同的 **NetworkAttachmentDefinition** 对象。
2. 指定 SR-IOV Network Operator 安装到的命名空间。
3. 将 **<sriov_resource_name>** 替换为来自为这个额外网络定义 SR-IOV 硬件的 **SriovNetworkNodePolicy** 对象的 **.spec.resourceName** 参数的值。
4. 将 **<target_namespace>** 替换为 SriovNetwork 的目标命名空间。只有目标命名空间中的 pod 或虚拟机可以附加到 SriovNetwork。
5. 可选：使用额外网络的虚拟 LAN (VLAN) ID 替换 **<vlan>**。它需要是一个从 **0** 到 **4095** 范围内的一个整数值。默认值为 **0**。
6. 可选：将 **<spoof_check>** 替换为 VF 的 spoof 检查模式。允许的值是字符串 **"on"** 和 **"off"**。

重要

指定的值必须由引号包括，否则 SR-IOV Network Operator 将拒绝 CR。

7. 可选：将 **<link_state>** 替换为 Virtual Function (VF) 的链接状态。允许的值是 **enable**、**disable** 和 **auto**。
8. 可选：将 **<max_tx_rate>** 替换为 VF 的最大传输率（以 Mbps 为单位）。
9. 可选：将 **<min_tx_rate>** 替换为 VF 的最小传输率（以 Mbps 为单位）。这个值应该总是小于或等于最大传输率。

注意

Intel NIC 不支持 **minTxRate** 参数。如需更多信息，请参阅 BZ#1772847。

10. 可选：将 **<vlan_qos>** 替换为 VF 的 IEEE 802.1p 优先级级别。默认值为 **0**。
11. 可选：将 **<trust_vf>**替换为 VF 的信任模式。允许的值是字符串 **"on"** 和 **"off"**。

重要

指定的值必须由引号包括，否则 SR-IOV Network Operator 将拒绝 CR。

12. 可选：将 **<capabilities>** 替换为为这个网络配置的功能。

2. 运行以下命令来创建对象。用这个额外网络的名称替换 **<name>** 。

```
$ oc create -f <name>-sriov-network.yaml
```

3. 可选： 要确认与您在上一步中创建的 **SriovNetwork** 对象关联的 **NetworkAttachmentDefinition** 对象是否存在，请输入以下命令。将 **<namespace>** 替换为您在 **SriovNetwork** 对象中指定的命名空间。

```
$ oc get net-attach-def -n <namespace>
```

8.6.3. 使用命令行将虚拟机连接到 SR-IOV 网络

您可以通过在虚拟机配置中包含网络详情将虚拟机 (VM) 连接到 SR-IOV 网络。

流程

1. 将 SR-IOV 网络详情添加到虚拟机配置的 **spec.domain.devices.interfaces** 和 **spec.networks** 小节中，如下例所示：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  domain:
    devices:
      interfaces:
      - name: nic1 (1)
        sriov: {}
  networks:
  - name: nic1 (2)
    multus:
        networkName: sriov-network (3)
# ...
```

(1) 为 SR-IOV 接口指定唯一名称。

(2) 指定 SR-IOV 接口的名称。这必须与之前定义的 **interfaces.name** 相同。

(3) 指定 SR-IOV 网络附加定义的名称。

2. 应用虚拟机配置：

```
$ oc apply -f <vm_sriov>.yaml (1)
```

(1) 虚拟机 YAML 文件的名称。

8.6.4. 使用 Web 控制台将虚拟机连接到 SR-IOV 网络

您可以通过在虚拟机配置中包含网络详情将虚拟机连接到 SR-IOV 网络。

先决条件

- 您必须为网络创建网络附加定义。

流程

1. 进入到 **Virtualization → VirtualMachines**。
2. 点虚拟机查看 **VirtualMachine 详情**页。
3. 在 **Configuration** 选项卡上，点 **Network interfaces** 选项卡。

4. 点 **Add network interface**。
5. 输入接口名称。
6. 从 **Network** 列表中选择 SR-IOV 网络附加定义。
7. 从 **Type** 列表中选择 **SR-IOV**。
8. 可选：添加网络 **Model** 或 **Mac 地址**。
9. 点击 **Save**。
10. 重启或实时迁移虚拟机以应用更改。

8.6.5. 其他资源

- 配置 DPDK 工作负载以提高性能

8.7. 在 SR-IOV 中使用 DPDK

Data Plane Development Kit (DPDK) 提供了一组库和驱动程序，用于快速数据包处理。

您可以配置集群和虚拟机(VM)，以通过 SR-IOV 网络运行 DPDK 工作负载。

8.7.1. 为 DPDK 工作负载配置集群

您可以配置 OpenShift Container Platform 集群来运行 Data Plane Development Kit (DPDK) 工作负载，以提高网络性能。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问集群。
- 已安装 OpenShift CLI(**oc**)。
- 已安装 SR-IOV Network Operator。
- 已安装 Node Tuning Operator。

流程

1. 映射计算节点拓扑，以确定为 DPDK 应用程序隔离哪些非统一内存访问 (NUMA) CPU，以及为操作系统 (OS) 保留哪些非一致性内存访问 (NUMA) CPU。
2. 使用自定义角色标记计算节点的子集，例如 **worker-dpdk** ：

```
$ oc label node <node_name> node-role.kubernetes.io/worker-dpdk=""
```

3. 创建一个新的 **MachineConfigPool** 清单，其中包含 **spec.machineConfigSelector** 对象中的 **worker-dpdk** 标签：

MachineConfigPool 清单示例

```
apiVersion: machineconfiguration.openshift.io/v1
```

```
kind: MachineConfigPool
metadata:
  name: worker-dpdk
  labels:
    machineconfiguration.openshift.io/role: worker-dpdk
spec:
  machineConfigSelector:
    matchExpressions:
      - key: machineconfiguration.openshift.io/role
        operator: In
        values:
          - worker
          - worker-dpdk
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker-dpdk: ""
```

4. 创建一个 **PerformanceProfile** 清单，它应用到标记的节点以及您在上一步中创建的机器配置池。性能配置集指定为 DPDK 应用程序隔离的 CPU，以及用于保留保留而保留的 CPU。

   PerformanceProfile 清单示例

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: profile-1
spec:
  cpu:
    isolated: 4-39,44-79
    reserved: 0-3,40-43
  globallyDisableIrqLoadBalancing: true
  hugepages:
    defaultHugepagesSize: 1G
    pages:
    - count: 8
      node: 0
      size: 1G
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/worker-dpdk: ""
  numa:
    topologyPolicy: single-numa-node
```

注意

应用 **MachineConfigPool** 和 **PerformanceProfile** 清单后，计算节点会自动重启。

5. 从 **PerformanceProfile** 对象的 **status.runtimeClass** 字段检索生成的 **RuntimeClass** 资源的名称：

```
$ oc get performanceprofiles.performance.openshift.io profile-1 -
o=jsonpath='{.status.runtimeClass}{"\n"}'
```

6. 通过编辑 **HyperConverged** 自定义资源 (CR) 将之前获取的 **RuntimeClass** 名称设置为 **virt-launcher** pod 的默认容器运行时类：

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
    --type='json' -p='[{"op": "add", "path": "/spec/defaultRuntimeClass", "value":"<runtimeclass-name>"}]'
```

注意

编辑 **HyperConverged** CR 会更改影响更改后创建的所有虚拟机的全局设置。

7. 如果您的启用 DPDK 的计算节点使用 Simultaneous 多线程 (SMT)，请通过编辑 **HyperConverged** CR 来启用 **AlignCPUs** enabler：

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
    --type='json' -p='[{"op": "replace", "path": "/spec/featureGates/alignCPUs", "value": true}]'
```

注意

启用 **AlignCPU** 可让 OpenShift Virtualization 请求两个额外的专用 CPU，在使用仿真程序线程隔离时将 CPU 总数引入到甚至奇偶校验。

8. 创建一个 **SriovNetworkNodePolicy** 对象，并将 **spec.deviceType** 字段设置为 **vfio-pci** ：

SriovNetworkNodePolicy 清单示例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: intel_nics_dpdk
  deviceType: vfio-pci
  mtu: 9000
  numVfs: 4
  priority: 99
  nicSelector:
    vendor: "8086"
    deviceID: "1572"
    pfNames:
      - eno3
    rootDevices:
      - "0000:19:00.2"
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
```

其他资源

- 使用 CPU Manager 和拓扑管理器
- 配置巨页

- 创建自定义机器配置池

8.7.2. 为 DPDK 工作负载配置项目

您可以将项目配置为在 SR-IOV 硬件中运行 DPDK 工作负载。

先决条件

- 集群被配置为运行 DPDK 工作负载。

流程

1. 为您的 DPDK 应用程序创建一个命名空间：

```
$ oc create ns dpdk-checkup-ns
```

2. 创建一个 **SriovNetwork** 对象来引用 **SriovNetworkNodePolicy** 对象。创建 **SriovNetwork** 对象时，SR-IOV Network Operator 会自动创建一个 **NetworkAttachmentDefinition** 对象。

   SriovNetwork 清单示例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: dpdk-sriovnetwork
  namespace: openshift-sriov-network-operator
spec:
  ipam: |
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [{
        "dst": "0.0.0.0/0"
      }],
      "gateway": "10.56.217.1"
    }
  networkNamespace: dpdk-checkup-ns (1)
  resourceName: intel_nics_dpdk (2)
  spoofChk: "off"
  trust: "on"
  vlan: 1019
```

   (1) 部署 **NetworkAttachmentDefinition** 对象的命名空间。

   (2) 为 DPDK 工作负载配置集群时创建的 **SriovNetworkNodePolicy** 对象的 **spec.resourceName** 属性的值。

3. 可选：运行虚拟机延迟检查以验证网络是否已正确配置。

4. 可选：运行 DPDK 检查，以验证命名空间是否已准备好 DPDK 工作负载。

其他资源

其他资源

- 处理项目
- 虚拟机延迟检查
- DPDK 检查

8.7.3. 为 DPDK 工作负载配置虚拟机

您可以在虚拟机 (VM) 上运行 Data Packet Development Kit (DPDK) 工作负载，以实现较低延迟和更高的吞吐量，以便在用户空间中更快地处理数据包。DPDK 使用 SR-IOV 网络进行基于硬件的 I/O 共享。

先决条件

- 集群被配置为运行 DPDK 工作负载。
- 您已创建并配置了运行虚拟机的项目。

流程

1. 编辑 **VirtualMachine** 清单，使其包含 SR-IOV 网络接口、CPU 拓扑、CRI-O 注解和巨页的信息：

   VirtualMachine 清单示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: rhel-dpdk-vm
spec:
  running: true
  template:
    metadata:
      annotations:
        cpu-load-balancing.crio.io: disable (1)
        cpu-quota.crio.io: disable (2)
        irq-load-balancing.crio.io: disable (3)
    spec:
      domain:
        cpu:
          sockets: 1 (4)
          cores: 5 (5)
          threads: 2
          dedicatedCpuPlacement: true
          isolateEmulatorThread: true
        interfaces:
          - masquerade: {}
            name: default
          - model: virtio
            name: nic-east
            pciAddress: '0000:07:00.0'
            sriov: {}
        networkInterfaceMultiqueue: true
        rng: {}
```

```
      memory:
        hugepages:
          pageSize: 1Gi 6
          guest: 8Gi
      networks:
        - name: default
          pod: {}
        - multus:
            networkName: dpdk-net 7
          name: nic-east
# ...
```

1. 此注解指定为容器使用的 CPU 禁用负载均衡。
2. 此注解指定容器使用的 CPU 配额被禁用。
3. 此注解指定，容器使用的 CPU 禁用中断请求 (IRQ) 负载均衡。
4. 虚拟机中的插槽数。此字段必须设置为 **1**，才能从相同的 Non-Uniform Memory Access (NUMA) 节点调度 CPU。
5. 虚拟机中的内核数。这必须大于或等于 **1**。在本例中，虚拟机被调度有 5 个超线程或 10 个 CPU。
6. 巨页的大小。x86-64 构架的可能值为 1Gi 和 2Mi。在这个示例中，请求大小为 1Gi 的 8 个巨页。
7. SR-IOV **NetworkAttachmentDefinition** 对象的名称。

2. 保存并退出编辑器。
3. 应用 **VirtualMachine** 清单：

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

4. 配置客户机操作系统。以下示例显示了 RHEL 8 OS 的配置步骤：
   a. 使用 GRUB 引导装载程序命令行界面配置巨页。在以下示例中，指定了 8 个 1G 巨页。

```
$ grubby --update-kernel=ALL --args="default_hugepagesz=1GB hugepagesz=1G
hugepages=8"
```

   b. 要使用 TuneD 应用程序中的 **cpu-partitioning** 配置集来实现低延迟性能优化，请运行以下命令：

```
$ dnf install -y tuned-profiles-cpu-partitioning
```

```
$ echo isolated_cores=2-9 > /etc/tuned/cpu-partitioning-variables.conf
```

前两个 CPU (0 和 1) 被设置，用于保存任务，其余则隔离 DPDK 应用程序。

```
$ tuned-adm profile cpu-partitioning
```

c. 使用 **driverctl** 设备驱动程序控制工具覆盖 SR-IOV NIC 驱动程序：

```
$ dnf install -y driverctl
```

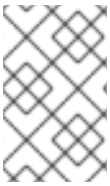
```
$ driverctl set-override 0000:07:00.0 vfio-pci
```

5. 重启虚拟机以应用更改。

8.8. 将虚拟机连接到 OVN-KUBERNETES 二级网络

您可以将虚拟机 (VM) 连接到 Open Virtual Network (OVN)-Kubernetes 二级网络。OpenShift Virtualization 支持 OVN-Kubernetes 的第 2 层和 localnet 拓扑。

- 第 2 层拓扑通过集群范围的逻辑交换机连接工作负载。OVN-Kubernetes Container Network Interface (CNI) 插件使用 Geneve (Generic Network Virtualization Encapsulation) 协议在节点间创建覆盖网络。您可以使用此覆盖网络在不同的节点上连接虚拟机，而无需配置任何其他物理网络基础架构。
- localnet 拓扑将二级网络连接到物理网络。这可让 east-west 集群流量并访问在集群外运行的服务，但它需要在集群节点上配置底层 Open vSwitch (OVS) 系统。

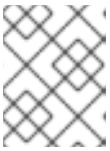


注意

OVN-Kubernetes 二级网络与 多网络策略 API 兼容，它提供 **MultiNetworkPolicy** 自定义资源定义(CRD)来控制进出虚拟机的流量流。您可以使用 **ipBlock** 属性来定义特定 CIDR 块的网络策略入口和出口规则。

要配置 OVN-Kubernetes 二级网络并将虚拟机附加到该网络，请执行以下步骤：

1. 通过创建网络附加定义(NAD)来配置 OVN-Kubernetes 二级网络。



注意

对于 localnet 拓扑，您必须在创建 NAD 前创建一个 **NodeNetworkConfigurationPolicy** 对象来配置 OVS 网桥。

2. 通过在虚拟机规格中添加网络详情，将虚拟机连接到 OVN-Kubernetes 二级网络。

8.8.1. 创建 OVN-Kubernetes NAD

您可以使用 OpenShift Container Platform Web 控制台或 CLI 创建 OVN-Kubernetes 层 2 或 localnet 网络附加定义(NAD)。

注意

不支持在虚拟机的网络附加定义中配置 IP 地址管理(IPAM)。

8.8.1.1. 使用 CLI 为第 2 层拓扑创建 NAD

您可以创建一个网络附加定义(NAD)，它描述了如何将 pod 附加到第 2 层覆盖网络。

先决条件

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

流程

1. 创建 **NetworkAttachmentDefinition** 对象：

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: l2-network
  namespace: my-namespace
spec:
  config: |2
    {
            "cniVersion": "0.3.1", (1)
            "name": "my-namespace-l2-network", (2)
            "type": "ovn-k8s-cni-overlay", (3)
            "topology":"layer2", (4)
            "mtu": 1300, (5)
            "netAttachDefName": "my-namespace/l2-network" (6)
    }
```

(1) CNI 规格版本。所需的值为 **0.3.1**。

(2) 网络的名称。此属性不是命名空间。例如，您可以有一个名为 **l2-network** 的网络，该网络从两个不同的命名空间中存在的两个不同的 **NetworkAttachmentDefinition** 对象引用。此功能可用于连接不同命名空间中的虚拟机。

(3) 要配置的 CNI 插件的名称。所需的值为 **ovn-k8s-cni-overlay**。

(4) 网络的拓扑配置。所需的值为 **layer2**。

(5) 可选：最大传输单元 (MTU) 值。默认值由内核自动设置。

(6) **NetworkAttachmentDefinition** 对象的 **metadata** 小节中的 **namespace** 和 **name** 字段的值。

注意

上例配置了一个集群范围的覆盖，没有定义子网。这意味着实现网络的逻辑交换机仅提供第 2 层通信。您必须在创建虚拟机时配置 IP 地址，方法是设置静态 IP 地址，或在网络上为动态 IP 地址部署 DHCP 服务器。

2. 应用清单：

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

8.8.1.2. 使用 CLI 为 localnet 拓扑创建 NAD

您可以创建一个网络附加定义 (NAD)，它描述了如何将 pod 附加到底层物理网络。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问集群。
- 已安装 OpenShift CLI(**oc**)。
- 已安装 Kubernetes NMState Operator。
- 您已创建了 **NodeNetworkConfigurationPolicy** 对象，将 OVN-Kubernetes 二级网络映射到 Open vSwitch (OVS) 网桥。

流程

1. 创建 **NetworkAttachmentDefinition** 对象：

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: localnet-network
  namespace: default
spec:
  config: |2
    {
            "cniVersion": "0.3.1", (1)
            "name": "localnet-network", (2)
            "type": "ovn-k8s-cni-overlay", (3)
            "topology": "localnet", (4)
            "netAttachDefName": "default/localnet-network" (5)
    }
```

(1) CNI 规格版本。所需的值为 **0.3.1**。

(2) 网络的名称。此属性必须与定义 OVS 网桥映射的 **NodeNetworkConfigurationPolicy** 对象的 **spec.desiredState.ovn.bridge-mappings.localnet** 字段的值匹配。

(3) 要配置的 CNI 插件的名称。所需的值为 **ovn-k8s-cni-overlay**。

(4) 网络的拓扑配置。所需的值为 **localnet**。

(5) **NetworkAttachmentDefinition** 对象的 **metadata** 小节中的 **namespace** 和 **name** 字段的值。

2. 应用清单：

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

8.8.1.3. 使用 Web 控制台为第 2 层拓扑创建 NAD

您可以创建一个网络附加定义 (NAD) 来描述如何将 pod 附加到第 2 层覆盖网络。

先决条件

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

流程

1. 在 web 控制台中进入 **Networking → NetworkAttachmentDefinition**。
2. 点 **Create Network Attachment Definition**。网络附加定义必须与 pod 或虚拟机位于同一个命名空间中。
3. 输入唯一 **Name** 和可选 **Description**。
4. 从 **Network Type** 列表中选择 **OVN Kubernetes L2 overlay 网络**。
5. 点 **Create**。

8.8.1.4. 使用 Web 控制台为 localnet 拓扑创建 NAD

您可以使用 OpenShift Container Platform Web 控制台创建网络附加定义(NAD)将工作负载连接到物理网络。

先决条件

- 您可以使用具有 **cluster-admin** 权限的用户访问集群。
- 使用 **nmstate** 将 localnet 配置为 OVS 网桥映射。

流程

1. 在 web 控制台中进入到 **Networking → NetworkAttachmentDefinition**。
2. 点 **Create Network Attachment Definition**。网络附加定义必须与 pod 或虚拟机位于同一个命名空间中。
3. 输入唯一 **Name** 和可选 **Description**。
4. 从 **Network Type** 列表中选择 **OVN Kubernetes secondary localnet network**。
5. 在 **Bridge mapping** 字段中输入预先配置的 localnet 标识符的名称。
6. 可选：您可以将 MTU 明确设置为指定的值。内核选择默认值。
7. 可选：封装 VLAN 中的流量。默认值为 none。
8. 点 **Create**。

8.8.2. 将虚拟机附加到 OVN-Kubernetes 二级网络

您可以使用 OpenShift Container Platform web 控制台或 CLI 将虚拟机(VM)附加到 OVN-Kubernetes 二级网络接口。

8.8.2.1. 使用 CLI 将虚拟机附加到 OVN-Kubernetes 二级网络

您可以通过在虚拟机配置中包含网络详情，将虚拟机 (VM) 连接到 OVN-Kubernetes 二级网络。

先决条件

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

- 已安装 OpenShift CLI(**oc**)。

流程

1. 编辑 **VirtualMachine** 清单以添加 OVN-Kubernetes 二级网络接口详情，如下例所示：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-server
spec:
  running: true
  template:
    spec:
      domain:
        devices:
          interfaces:
          - name: secondary (1)
            bridge: {}
        resources:
          requests:
            memory: 1024Mi
      networks:
      - name: secondary (2)
        multus:
          networkName: <nad_name> (3)
# ...
```

(1) OVN-Kubernetes 二级接口的名称。

(2) 网络的名称。这必须与 **spec.template.spec.domain.devices.interfaces.name** 字段的值匹配。

(3) **NetworkAttachmentDefinition** 对象的名称。

2. 应用 **VirtualMachine** 清单：

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

3. 可选：如果编辑了正在运行的虚拟机，您必须重启它才能使更改生效。

8.8.3. 其他资源

- 配置 OVN-Kubernetes 额外网络
- 关于 Kubernetes NMState Operator
- 配置 OVN-Kubernetes 额外网络映射
- 配置额外网络附加

8.9. 热插二级网络接口

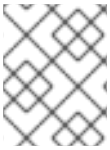
您可以在不停止虚拟机(VM)的情况下添加或删除二级网络接口。OpenShift Virtualization 支持热插使用 VirtIO 设备驱动程序的二级接口。

注意

单根 I/O 虚拟化(SR-IOV) 接口不支持热拔。

8.9.1. virtio 限制

每个 VirtIO 接口使用虚拟机中的有限 Peripheral Connect Interface (PCI) 插槽之一。总共有 32 个插槽可用。PCIe 插槽也被其他设备使用，必须提前保留，因此插槽可能按需提供。OpenShift Virtualization 为热插接口保留最多四个插槽。这包括任何现有插入的网络接口。例如，如果您的虚拟机有两个现有的 plugged 接口，您可以热插两个网络接口。



注意

可用于热插的实际插槽数量也取决于机器类型。例如，q35 机器类型的默认 PCI 拓扑支持热插一个额外的 PCIe 设备。有关 PCI 拓扑和热插支持的更多信息，请参阅 libvirt 文档。

如果在热插接口后重启虚拟机，则该接口就成为标准网络接口的一部分。

8.9.2. 使用 CLI 热插二级网络接口

在虚拟机运行时，热插到虚拟机(VM)的二级网络接口。

先决条件

- 网络附加定义与虚拟机相同的命名空间中配置。
- 已安装 **virtctl** 工具。
- 已安装 OpenShift CLI(**oc**)。

流程

1. 如果要热插网络接口的虚拟机没有运行，请使用以下命令启动它：

```
$ virtctl start <vm_name> -n <namespace>
```

2. 使用以下命令，将新的网络接口添加到正在运行的虚拟机中。编辑虚拟机规格向虚拟机和虚拟机实例(VMI)配置添加新网络接口，但不会将其附加到正在运行的虚拟机中。

```
$ oc edit vm <vm_name>
```

虚拟机配置示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-fedora
template:
  spec:
    domain:
```

```
      devices:
        interfaces:
        - name: defaultnetwork
          masquerade: {}
        # new interface
        - name: <secondary_nic> 1
          bridge: {}
      networks:
      - name: defaultnetwork
        pod: {}
      # new network
      - name: <secondary_nic> 2
        multus:
          networkName: <nad_name> 3
# ...
```

1. 指定新网络接口的名称。
2. 指定网络的名称。这必须与您在 **template.spec.domain.devices.interfaces** 列表中定义的新网络接口的**名称**相同。
3. 指定 **NetworkAttachmentDefinition** 对象的名称。

3. 要将网络接口附加到正在运行的虚拟机中，请运行以下命令来实时迁移虚拟机：

```
$ virtctl migrate <vm_name>
```

验证

1. 使用以下命令验证虚拟机实时迁移是否成功：

```
$ oc get VirtualMachineInstanceMigration -w
```

输出示例

```
NAME                        PHASE            VMI
kubevirt-migrate-vm-lj62q   Scheduling       vm-fedora
kubevirt-migrate-vm-lj62q   Scheduled        vm-fedora
kubevirt-migrate-vm-lj62q   PreparingTarget   vm-fedora
kubevirt-migrate-vm-lj62q   TargetReady      vm-fedora
kubevirt-migrate-vm-lj62q   Running          vm-fedora
kubevirt-migrate-vm-lj62q   Succeeded        vm-fedora
```

2. 通过检查 VMI 状态来验证新接口是否已添加到虚拟机中：

```
$ oc get vmi vm-fedora -ojsonpath="{ @.status.interfaces }"
```

输出示例

```
[
  {
    "infoSource": "domain, guest-agent",
```

```
    "interfaceName": "eth0",
    "ipAddress": "10.130.0.195",
    "ipAddresses": [
      "10.130.0.195",
      "fd02:0:0:3::43c"
    ],
    "mac": "52:54:00:0e:ab:25",
    "name": "default",
    "queueCount": 1
  },
  {
    "infoSource": "domain, guest-agent, multus-status",
    "interfaceName": "eth1",
    "mac": "02:d8:b8:00:00:2a",
    "name": "bridge-interface", 1
    "queueCount": 1
  }
]
```

1 热插接口会出现在 VMI 状态中。

8.9.3. 使用 CLI 热拔二级网络接口

您可以从正在运行的虚拟机(VM)中删除二级网络接口。

注意

单根 I/O 虚拟化(SR-IOV) 接口不支持热拔。

先决条件

- 您的虚拟机必须正在运行。
- 虚拟机必须在运行 OpenShift Virtualization 4.14 或更高版本的集群中创建。
- 虚拟机必须附加了一个桥接网络接口。

流程

1. 编辑虚拟机规格以热拔二级网络接口。将接口状态设置为 **absent** 从客户机中分离网络接口，但接口仍然存在于 pod 中。

```
$ oc edit vm <vm_name>
```

虚拟机配置示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-fedora
template:
  spec:
    domain:
```

```
    devices:
      interfaces:
        - name: defaultnetwork
          masquerade: {}
        # set the interface state to absent
        - name: <secondary_nic>
          state: absent (1)
          bridge: {}
    networks:
      - name: defaultnetwork
        pod: {}
      - name: <secondary_nic>
        multus:
          networkName: <nad_name>
# ...
```

1 将接口状态设置为 **absent**，将其从正在运行的虚拟机中分离。从虚拟机规格中删除接口详情不会热拔二级网络接口。

2. 通过迁移虚拟机从 pod 中删除接口：

```
$ virtctl migrate <vm_name>
```

8.9.4. 其他资源

- 安装 virtctl
- 创建 Linux 网桥网络附加定义
- 将虚拟机连接到 Linux 网桥网络
- 创建 SR-IOV 网络附加定义
- 将虚拟机连接到 SR-IOV 网络

8.10. 将虚拟机连接到服务网格

OpenShift Virtualization 现在与 OpenShift Service Mesh 集成。您可以使用 IPv4 监控、视觉化和控制在默认 pod 网络上运行虚拟机工作负载的 pod 之间的流量。

8.10.1. 将虚拟机添加到服务网格中

要将虚拟机 (VM) 工作负载添加到服务网格中，请在虚拟机配置文件中启用自动 sidecar 注入，方法是将 **sidecar.istio.io/inject** 注解设置为 **true**。然后，将虚拟机公开为服务，以便在网格中查看应用程序。

重要

为了避免端口冲突，请不要使用 Istio sidecar 代理使用的端口。它们包括 15000、15001、15006、15008、15020、15021 和 15090。

先决条件

- 已安装 Service Mesh Operator。

- 已创建 Service Mesh control plane。
- 将 VM 项目添加到 Service Mesh member roll。

流程

1. 编辑虚拟机配置文件以添加 **sidecar.istio.io/inject: "true"** 注解：

配置文件示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm-istio
  name: vm-istio
spec:
  runStrategy: Always
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm-istio
        app: vm-istio (1)
      annotations:
        sidecar.istio.io/inject: "true" (2)
    spec:
      domain:
        devices:
          interfaces:
          - name: default
            masquerade: {} (3)
          disks:
          - disk:
              bus: virtio
            name: containerdisk
          - disk:
              bus: virtio
            name: cloudinitdisk
        resources:
          requests:
            memory: 1024M
      networks:
      - name: default
        pod: {}
      terminationGracePeriodSeconds: 180
      volumes:
      - containerDisk:
          image: registry:5000/kubevirt/fedora-cloud-container-disk-demo:devel
        name: containerdisk
```

(1) 键/值对（标签）必须与 service selector 属性匹配。

(2) 启用自动 sidecar 注入的注解。

(3) 用于默认 pod 网络的绑定方法（伪装模式）。

2. 应用 VM 配置：

```
$ oc apply -f <vm_name>.yaml 1
```

1 虚拟机 YAML 文件的名称。

3. 创建一个 **Service** 对象，将虚拟机公开给服务网格。

```
apiVersion: v1
kind: Service
metadata:
  name: vm-istio
spec:
  selector:
    app: vm-istio 1
  ports:
    - port: 8080
      name: http
      protocol: TCP
```

1 服务选择器，决定服务的目标 pod 集合。此属性对应于虚拟机配置文件中的 **spec.metadata.labels** 字段。在上例中，名为 **vm-istio** 的 **Service** 对象在任何带有标签 **app=vm-istio** 的 pod 上都以 TCP 端口 8080 为目标。

4. 创建服务：

```
$ oc create -f <service_name>.yaml 1
```

1 服务 YAML 文件的名称。

8.10.2. 其他资源

- 安装 Service Mesh Operator
- 创建 Service Mesh control plane
- 在 Service Mesh member roll 中添加项目

8.11. 为实时迁移配置专用网络

您可以为实时迁移配置专用 Multus 网络。专用的网络可最小化实时迁移期间对租户工作负载的网络饱和影响。

8.11.1. 为实时迁移配置专用的二级网络

要为实时迁移配置专用的二级网络，您必须首先使用 CLI 创建桥接网络附加定义(NAD)。然后，您可以将 **NetworkAttachmentDefinition** 对象的名称添加到 **HyperConverged** 自定义资源(CR)。

先决条件

- 已安装 OpenShift CLI（**oc**）。

- 您以具有 **cluster-admin** 角色的用户身份登录到集群。
- 每个节点至少有两个网络接口卡 (NIC)。
- 用于实时迁移的 NIC 连接到同一 VLAN。

流程

1. 根据以下示例创建 **NetworkAttachmentDefinition** 清单：

   配置文件示例

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: my-secondary-network 1
  namespace: openshift-cnv 2
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "migration-bridge",
    "type": "macvlan",
    "master": "eth1", 3
    "mode": "bridge",
    "ipam": {
      "type": "whereabouts", 4
      "range": "10.200.5.0/24" 5
    }
  }'
```

   1 指定 **NetworkAttachmentDefinition** 对象的名称。

   2 3 指定要用于实时迁移的 NIC 名称。

   4 指定为 NAD 提供网络的 CNI 插件名称。

   5 为二级网络指定一个 IP 地址范围。这个范围不得与主网络的 IP 地址重叠。

2. 运行以下命令，在默认编辑器中打开 **HyperConverged** CR：

```
oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

3. 将 **NetworkAttachmentDefinition** 对象的名称添加到 **HyperConverged** CR 的 **spec.liveMigrationConfig** 小节中：

   HyperConverged 清单示例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  liveMigrationConfig:
```

```
    completionTimeoutPerGiB: 800
    network: <network> 1
    parallelMigrationsPerCluster: 5
    parallelOutboundMigrationsPerNode: 2
    progressTimeout: 150
# ...
```

1 指定要用于实时迁移的 Multus **NetworkAttachmentDefinition** 对象的名称。

4. 保存更改并退出编辑器。**virt-handler** Pod 会重启并连接到二级网络。

验证

- 当运行虚拟机的节点置于维护模式时，虚拟机会自动迁移到集群中的另一个节点。您可以通过检查虚拟机实例(VMI)元数据中的目标 IP 地址，验证迁移是否在二级网络中发生，而不是默认 pod 网络。

```
$ oc get vmi <vmi_name> -o jsonpath='{.status.migrationState.targetNodeAddress}'
```

8.11.2. 使用 Web 控制台选择专用网络

您可以使用 OpenShift Container Platform Web 控制台为实时迁移选择一个专用网络。

先决条件

- 为实时迁移配置了 Multus 网络。

流程

1. 在 OpenShift Container Platform web 控制台中进入到 **Virtualization > Overview**。
2. 点 **Settings** 选项卡，然后点 **Live migration**。
3. 从 **Live migration network** 列表中选择网络。

8.11.3. 其他资源

- 配置实时迁移限制和超时

8.12. 配置和查看 IP 地址

您可以在创建虚拟机(VM)时配置 IP 地址。IP 地址使用 cloud-init 置备。

您可以使用 OpenShift Container Platform Web 控制台或命令行查看虚拟机的 IP 地址。QEMU 客户机代理收集网络信息。

8.12.1. 为虚拟机配置 IP 地址

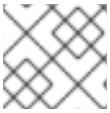
您可以使用 web 控制台或命令行创建虚拟机(VM)时配置静态 IP 地址。

您可以使用命令行在创建虚拟机时配置动态 IP 地址。

IP 地址使用 cloud-init 置备。

8.12.1.1. 使用命令行在创建虚拟机时配置 IP 地址

您可以在创建虚拟机时配置静态或动态 IP 地址。IP 地址使用 cloud-init 置备。



注意

如果虚拟机连接到 pod 网络，pod 网络接口是默认路由，除非您更新它。

先决条件

- 虚拟机连接到第二个网络。
- 在二级网络上有一个 DHCP 服务器，用于为虚拟机配置动态 IP。

流程

- 编辑虚拟机配置的 **spec.template.spec.volumes.cloudInitNoCloud.networkData** 小节：
  - 要配置动态 IP 地址，请指定接口名称并启用 DHCP：

```
kind: VirtualMachine
spec:
# ...
  template:
  # ...
    spec:
      volumes:
      - cloudInitNoCloud:
          networkData: |
            version: 2
            ethernets:
              eth1: 1
                dhcp4: true
```

1 指定接口名称。

  - 要配置静态 IP，请指定接口名称和 IP 地址：

```
kind: VirtualMachine
spec:
# ...
  template:
  # ...
    spec:
      volumes:
      - cloudInitNoCloud:
          networkData: |
            version: 2
            ethernets:
              eth1: 1
                addresses:
                - 10.10.10.14/24 2
```

1 指定接口名称。

2 指定静态 IP 地址。

8.12.2. 查看虚拟机的 IP 地址

您可以使用 OpenShift Container Platform Web 控制台或命令行查看虚拟机的 IP 地址。

QEMU 客户机代理收集网络信息。

8.12.2.1. 使用 web 控制台查看虚拟机的 IP 地址

您可以使用 OpenShift Container Platform web 控制台查看虚拟机的 IP 地址。

注意

您必须在虚拟机上安装 QEMU 客户机代理，以查看二级网络接口的 IP 地址。pod 网络接口不需要 QEMU 客户机代理。

流程

1. 在 OpenShift Container Platform 控制台中，从侧边菜单中点 **Virtualization** → **VirtualMachines**。
2. 选择一个虚拟机以打开 **VirtualMachine 详情**页。
3. 点 **Details** 选项卡查看 IP 地址。

8.12.2.2. 使用命令行查看虚拟机的 IP 地址

您可以使用命令行查看虚拟机的 IP 地址。

注意

您必须在虚拟机上安装 QEMU 客户机代理，以查看二级网络接口的 IP 地址。pod 网络接口不需要 QEMU 客户机代理。

流程

- 运行以下命令来获取虚拟机实例配置：

```
$ oc describe vmi <vmi_name>
```

输出示例

```
# ...
Interfaces:
  Interface Name:  eth0
  Ip Address:      10.244.0.37/24
  Ip Addresses:
    10.244.0.37/24
    fe80::858:aff:fef4:25/64
```

```
Mac:            0a:58:0a:f4:00:25
Name:           default
Interface Name:  v2
Ip Address:     1.1.1.7/24
Ip Addresses:
 1.1.1.7/24
 fe80::f4d9:70ff:fe13:9089/64
Mac:            f6:d9:70:13:90:89
Interface Name:  v1
Ip Address:     1.1.1.1/24
Ip Addresses:
 1.1.1.1/24
 1.1.1.2/24
 1.1.1.4/24
 2001:de7:0:f101::1/64
 2001:db8:0:f101::1/64
 fe80::1420:84ff:fe10:17aa/64
Mac:            16:20:84:10:17:aa
```

8.12.3. 其他资源

- 安装 QEMU 客户机代理

8.13. 使用其外部 FQDN 访问虚拟机

您可以使用其完全限定域名 (FQDN) 访问从集群外部附加到二级网络接口的虚拟机。

重要

使用其 FQDN 从集群外部访问虚拟机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。

8.13.1. 为二级网络配置 DNS 服务器

当您在 **HyperConverged** 自定义资源 (CR) 中启用 **deployKubeSecondaryDNS** 功能门时，Cluster Network Addons Operator (CNAO) 会部署域名服务器 (DNS) 服务器和监控组件。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 已为集群配置负载均衡器。
- 使用 **cluster-admin** 权限登录到集群。

流程

1. 根据以下示例运行 **oc expose** 命令，创建一个负载均衡器服务来公开集群外的 DNS 服务器：

```
$ oc expose -n openshift-cnv deployment/secondary-dns --name=dns-lb \
  --type=LoadBalancer --port=53 --target-port=5353 --protocol='UDP'
```

2. 运行以下命令来检索外部 IP 地址：

```
$ oc get service -n openshift-cnv
```

输出示例

```
NAME      TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)         AGE
dns-lb    LoadBalancer   172.30.27.5    10.46.41.94      53:31829/TCP    5s
```

3. 运行以下命令，在默认编辑器中编辑 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

4. 根据以下示例启用 DNS 服务器和监控组件：

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  featureGates:
    deployKubeSecondaryDNS: true
  kubeSecondaryDNSNameServerIP: "10.46.41.94" (1)
# ...
```

(1) 指定负载均衡器服务公开的外部 IP 地址。

5. 保存文件并退出编辑器。

6. 运行以下命令来检索集群 FQDN：

```
$ oc get dnses.config.openshift.io cluster -o jsonpath='{.spec.baseDomain}'
```

输出示例

```
openshift.example.com
```

7. 使用以下方法之一指向 DNS 服务器：

- 将 **kubeSecondaryDNSNameServerIP** 值添加到本地机器的 **resolv.conf** 文件中。

注意

编辑 **resolv.conf** 文件会覆盖现有的 DNS 设置。

- 将 **kubeSecondaryDNSNameServerIP** 值和集群 FQDN 添加到企业 DNS 服务器记录中。例如：

```
vm.<FQDN>. IN NS ns.vm.<FQDN>.
```

```
ns.vm.<FQDN>. IN A 10.46.41.94
```

8.13.2. 使用集群 FQDN 连接到二级网络上的虚拟机

您可以使用集群的完全限定域名(FQDN)访问附加到二级网络接口的正在运行的虚拟机(VM)。

先决条件

- 您在虚拟机上安装了 QEMU 客户机代理。
- 虚拟机的 IP 地址是公共的。
- 为二级网络配置了 DNS 服务器。
- 您获取了集群的完全限定域名(FQDN)。

流程

1. 运行以下命令，从虚拟机配置检索网络接口名称：

```
$ oc get vm -n <namespace> <vm_name> -o yaml
```

输出示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  running: true
  template:
    spec:
      domain:
        devices:
          interfaces:
            - bridge: {}
              name: example-nic
# ...
      networks:
      - multus:
          networkName: bridge-conf
        name: example-nic 1
```

1 请注意网络接口的名称。

2. 使用 **ssh** 命令连接到虚拟机：

```
$ ssh <user_name>@<interface_name>.<vm_name>.<namespace>.vm.<cluster_fqdn>
```

8.13.3. 其他资源

- 使用负载均衡器配置集群入口流量
- 使用 MetalLB 进行负载平衡
- 为虚拟机配置 IP 地址

8.14. 为网络接口管理 MAC 地址池

KubeMacPool 组件从共享 MAC 地址池为虚拟机(VM)网络接口分配 MAC 地址。这样可确保为每个网络接口分配唯一的 MAC 地址。

从该虚拟机创建的虚拟机实例在重启后保留分配的 MAC 地址。

注意

KubeMacPool 不处理独立于虚拟机创建的虚拟机实例。

8.14.1. 使用命令行管理 KubeMacPool

您可以使用命令行禁用和重新启用 KubeMacPool。

KubeMacPool 默认启用。

流程

- 要在两个命名空间中禁用 KubeMacPool，请运行以下命令：

```
$ oc label namespace <namespace1> <namespace2>
mutatevirtualmachines.kubemacpool.io=ignore
```

- 要在两个命名空间中重新启用 KubeMacPool，请运行以下命令：

```
$ oc label namespace <namespace1> <namespace2>
mutatevirtualmachines.kubemacpool.io-
```

第 9 章 存储

9.1. 存储配置概述

您可以配置默认存储类、存储配置集、Containerized Data Importer (CDI)、数据卷和自动引导源更新。

9.1.1. 存储

以下存储配置任务是必需的：

配置默认存储类

您必须为集群配置默认存储类。否则，集群无法接收自动引导源更新。

配置存储配置集

如果您的存储供应商没有被 CDI 识别，您必须配置存储配置集。存储配置集根据关联的存储类提供推荐的存储设置。

以下存储配置任务是可选的：

为文件系统开销保留额外的 PVC 空间

默认情况下，为开销保留 5.5% 的文件系统 PVC，从而减少了虚拟机磁盘的可用空间。您可以配置不同的开销值。

使用 hostpath 置备程序配置本地存储

您可以使用 hostpath 置备程序(HPP)为虚拟机配置本地存储。安装 OpenShift Virtualization Operator 时，会自动安装 HPP Operator。

配置用户权限以在命名空间间克隆数据卷

您可以配置 RBAC 角色，以便用户在命名空间间克隆数据卷。

9.1.2. 容器化 Data Importer

您可以执行以下 Containerized Data Importer (CDI) 配置任务：

覆盖命名空间的资源请求限制

您可以配置 CDI，将虚拟机磁盘导入、上传并克隆到命名空间中，这可能受 CPU 和内存资源限制。

配置 CDI 涂销空间

CDI 需要涂销空间（临时存储）来完成一些操作，如导入和上传虚拟机镜像。在此过程中，CDI 会提供一个与支持目标数据卷（DV）的 PVC 大小相等的涂销空间 PVC。

9.1.3. 数据卷

您可以执行以下数据卷配置任务：

为数据卷启用预分配

CDI 可以预先分配磁盘空间，以便在创建数据卷时提高写入性能。您可以为特定数据卷启用预分配。

管理数据卷注解

数据卷注解允许您管理 pod 行为。您可以将一个或多个注解添加到数据卷，然后将其传播到创建的导入程序 pod。

9.1.4. 引导源更新

您可以执行以下引导源更新配置任务：

管理自动引导源更新

通过引导源，可让虚拟机 (VM) 的创建更容易和高效。如果启用了自动引导源更新，CDI 导入、轮询和更新镜像，以便为新虚拟机准备好克隆它们。默认情况下，CDI 自动更新红帽引导源。您可以为自定义引导源启用自动更新。

9.2. 配置存储配置集

存储配置集根据关联的存储类提供推荐的存储设置。为每个存储类分配一个存储配置文件。

如果存储供应商已配置为识别和与存储供应商的功能交互，Containerized Data Importer (CDI) 会识别存储供应商。

对于可识别的存储类型，CDI 提供优化 PVC 创建的值。您还可以通过自定义存储配置集来为存储类配置自动设置。如果 CDI 没有识别您的存储供应商，您必须配置存储配置集。

重要

在 Red Hat OpenShift Data Foundation 中使用 OpenShift Virtualization 时，指定创建虚拟机磁盘时 RBD 块模式持久性卷声明(PVC)。RBD 块模式卷效率更高，并且比 Ceph FS 或 RBD 文件系统模式 PVC 提高性能。

要指定 RBD 块模式 PVC，请使用 'ocs-storagecluster-ceph-rbd' 存储类和 **VolumeMode: Block**。

9.2.1. 自定义存储配置集

您可以通过编辑置备程序存储类的 **StorageProfile** 对象来指定默认参数。这些默认参数只有在 **DataVolume** 对象中没有配置持久性卷声明 (PVC) 时才适用。

您无法修改存储类参数。要进行更改，请删除并重新创建存储类。然后，您必须重新应用之前对存储配置集所做的任何自定义。

存储配置文件中的空 **status** 部分表示存储置备程序不被 Containerized Data Interface（CDI）识别。如果您有存储置备程序无法被 CDI 识别，则需要自定义存储配置集。在这种情况下，管理员在存储配置集中设置适当的值以确保分配成功。

警告

如果您创建数据卷并省略 YAML 属性，且存储配置集中没有定义这些属性，则不会分配请求的存储，也不会创建底层持久性卷声明（PVC）。

先决条件

- 确保存储类及其供应商支持您计划的配置。在存储配置集中指定不兼容的配置会导致卷置备失败。

流程

1. 编辑存储配置文件。在本例中，CDI 无法识别置备程序。

```
$ oc edit storageprofile <storage_class>
```

存储配置集示例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <unknown_provisioner_class>
# ...
spec: {}
status:
  provisioner: <unknown_provisioner>
  storageClass: <unknown_provisioner_class>
```

2. 在存储配置集中提供所需的属性值：

存储配置集示例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <unknown_provisioner_class>
# ...
spec:
  claimPropertySets:
  - accessModes:
    - ReadWriteOnce (1)
    volumeMode:
      Filesystem (2)
status:
  provisioner: <unknown_provisioner>
  storageClass: <unknown_provisioner_class>
```

(1) 您选择的 **accessModes**。

(2) 您选择的 **volumeMode**。

保存更改后，所选值将显示在存储配置集的 **status** 项中。

9.2.1.1. 使用存储配置集设置默认克隆策略

您可以使用存储配置集为存储类设置默认克隆方法，从而创建 *克隆策略*。例如，如果您的存储供应商只支持某些克隆方法，设置克隆策略会很有用。它还允许您选择一个限制资源使用或最大化性能的方法。

可以通过将存储配置集中的 **cloneStrategy** 属性设置为以下值之一来指定克隆策略：

- 配置快照时，默认使用 **snapshot**。如果 CDI 识别存储供应商并且供应商支持 Container Storage Interface (CSI) 快照，则使用快照方法。此克隆策略使用临时卷快照来克隆卷。
- **copy** 使用源 pod 和目标 pod 将数据从源卷复制到目标卷。主机辅助克隆是最有效的克隆方法。

- **csi-clone** 使用 CSI 克隆 API 在不使用临时卷快照的情况下高效地克隆现有卷。与 **snapshot** 或 **copy** 不同（它们在没有定义存储配置集时被默认使用），只有在 **StorageProfile** 对象中为置备程序存储类指定它时，才会使用 CSI 卷克隆。

注意

您还可以在不修改 YAML **spec** 部分中的默认 **claimPropertySets** 的情况下使用 CLI 设置克隆策略。

存储配置集示例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <provisioner_class>
# ...
spec:
  claimPropertySets:
  - accessModes:
    - ReadWriteOnce (1)
    volumeMode:
      Filesystem (2)
  cloneStrategy: csi-clone (3)
status:
  provisioner: <provisioner>
  storageClass: <provisioner_class>
```

(1) 指定访问模式。

(2) 指定卷模式。

(3) 指定默认克隆策略。

表 9.1. 存储供应商和默认行为

存储供应商	默认行为
rook-ceph.rbd.csi.ceph.com	Snapshot
openshift-storage.rbd.csi.ceph.com	Snapshot
csi-vxflexos.dellemc.com	CSI 克隆
csi-isilon.dellemc.com	CSI 克隆
csi-powermax.dellemc.com	CSI 克隆
csi-powerstore.dellemc.com	CSI 克隆
hspc.csi.hitachi.com	CSI 克隆

存储供应商	默认行为
csi.hpe.com	CSI 克隆
spectrumscale.csi.ibm.com	CSI 克隆
rook-ceph.rbd.csi.ceph.com	CSI 克隆
openshift-storage.rbd.csi.ceph.com	CSI 克隆
cephfs.csi.ceph.com	CSI 克隆
openshift-storage.cephfs.csi.ceph.com	CSI 克隆

9.3. 管理自动引导源更新

您可以为以下引导源管理自动更新：

- 所有红帽引导源
- 所有自定义引导源
- 独立红帽或自定义引导源

通过引导源，可让虚拟机 (VM) 的创建更容易和高效。如果启用了自动引导源更新，Containerized Data Importer (CDI) 导入、轮询和更新镜像，以便为新虚拟机克隆它们。默认情况下，CDI 自动更新红帽引导源。

9.3.1. 管理红帽引导源更新

您可以通过禁用 **enableCommonBootImageImport** 功能门，选择对所有系统定义的引导源的自动更新。如果您禁用这个功能门，则所有 **DataImportCron** 对象都会被删除。这不会删除之前导入存储操作系统镜像的引导源对象，但管理员可以手动删除它们。

当禁用 **enableCommonBootImageImport** 功能门时，**DataSource** 对象会被重置，以便它们不再指向原始引导源。管理员可以通过为 **DataSource** 对象创建新的持久性卷声明(PVC)或卷快照来手动提供引导源，然后使用操作系统镜像填充它。

9.3.1.1. 为所有系统定义的引导源管理自动更新

禁用自动引导源导入和更新可能会降低资源使用量。在断开连接的环境中，禁用自动引导源更新可防止 **CDIDataImportCronOutdated** 警报填满日志。

要禁用所有系统定义的引导源的自动更新，请通过将值设为 **false** 来关闭 **enableCommonBootImageImport** 功能门。将此值设置为 **true** 可重新启用功能门并重新打开自动更新。

注意

自定义引导源不受此设置的影响。

流程

- 通过编辑 **HyperConverged** 自定义资源 (CR) 为自动引导源更新切换功能门。
  - 要禁用自动引导源更新，请将 **HyperConverged** CR 中的 **spec.featureGates.enableCommonBootImageImport** 字段设置为 **false**。例如：

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op": "replace", "path": \
  "/spec/featureGates/enableCommonBootImageImport", \
  "value": false}]'
```

  - 要重新启用自动引导源更新，请将 **HyperConverged** CR 中的 **spec.featureGates.enableCommonBootImageImport** 字段设置为 **true**。例如：

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op": "replace", "path": \
  "/spec/featureGates/enableCommonBootImageImport", \
  "value": true}]'
```

9.3.2. 管理自定义引导源更新

不是由 OpenShift Virtualization 提供的*自定义*引导源不受功能门控制。您必须通过编辑 **HyperConverged** 自定义资源 (CR) 来单独管理它们。

重要

您必须配置存储类。否则，集群无法接收自定义引导源的自动更新。详情请参阅定义一个存储类。

9.3.2.1. 为自定义引导源更新配置存储类

您可以通过编辑 **HyperConverged** 自定义资源 (CR) 来覆盖默认存储类。

重要

引导源使用默认存储类从存储创建。如果您的集群没有默认存储类，则必须在为自定义引导源配置自动更新前定义一个。

流程

1. 运行以下命令，在默认编辑器中打开 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 通过在 **storageClassName** 字段中输入值来定义新的存储类：

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
```

```
 - metadata:
     name: rhel8-image-cron
   spec:
     template:
       spec:
         storageClassName: <new_storage_class> 1
     schedule: "0 */12 * * *" 2
     managedDataSource: <data_source> 3
# ...
```

1 定义存储类。

2 必需：以 cron 格式指定的作业调度。

3 必需：要使用的数据源。

> For the custom image to be detected as an available boot source, the value of the `spec.dataVolumeTemplates.spec.sourceRef.name` parameter in the VM template must match this value.

3. 从当前的默认存储类中删除 **storageclass.kubernetes.io/is-default-class** 注解。

   a. 运行以下命令，检索当前默认存储类的名称：

   ```
   $ oc get storageclass
   ```

   输出示例

   ```
   NAME                    PROVISIONER                    RECLAIMPOLICY
   VOLUMEBINDINGMODE    ALLOWVOLUMEEXPANSION  AGE
   csi-manila-ceph         manila.csi.openstack.org       Delete        Immediate
   false                11d
   hostpath-csi-basic (default)  kubevirt.io.hostpath-provisioner Delete
   WaitForFirstConsumer false                11d 1
   ```

   1 在本例中，当前的默认存储类名为 **hostpath-csi-basic**。

   b. 运行以下命令，从当前默认存储类中删除注解：

   ```
   $ oc patch storageclass <current_default_storage_class> -p '{"metadata": {"annotations":
   {"storageclass.kubernetes.io/is-default-class":"false"}}}' 1
   ```

   1 将 **<current_default_storage_class>** 替换为默认存储类的 **storageClassName** 值。

4. 运行以下命令，将新存储类设置为默认值：

   ```
   $ oc patch storageclass <new_storage_class> -p '{"metadata":{"annotations":
   {"storageclass.kubernetes.io/is-default-class":"true"}}}' 1
   ```

   1 将 **<new_storage_class>** 替换为添加到 **HyperConverged** CR 中的 **storageClassName** 值。

9.3.2.2. 为自定义引导源启用自动更新

OpenShift Virtualization 默认自动更新系统定义的引导源，但不会自动更新自定义引导源。您必须通过编辑 **HyperConverged** 自定义资源 (CR) 手动启用自动更新。

先决条件

- 集群有一个默认存储类。

流程

1. 运行以下命令，在默认编辑器中打开 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 编辑 **HyperConverged** CR，在 **dataImportCronTemplates** 部分添加适当的模板和引导源。例如：

自定义资源示例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      name: centos7-image-cron
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true" (1)
      labels:
        instancetype.kubevirt.io/default-preference: centos.7
        instancetype.kubevirt.io/default-instancetype: u1.medium
    spec:
      schedule: "0 */12 * * *" (2)
      template:
        spec:
          source:
            registry: (3)
              url: docker://quay.io/containerdisks/centos:7-2009
          storage:
            resources:
              requests:
                storage: 30Gi
      garbageCollect: Outdated
      managedDataSource: centos7 (4)
```

(1) 对于将 **volumeBindingMode** 设置为 **WaitForFirstConsumer** 的存储类来说，这个注解是必需的。

(2) 以 cron 格式指定的作业调度计划。

(3) 用于从 registry 源创建数据卷。使用默认 **pod pullMethod** 而不是 **节点 pullMethod**，这基于 **节点** docker 缓存。当 registry 镜像通过 **Container.Image** 可用时，**节点** docker 缓存很有用，但 CDI 导入程序没有授权访问它。

4 要使自定义镜像被检测到为可用的引导源，镜像的 **managedDataSource** 的名称必须与模板的 **DataSource** 的名称匹配，它在 VM 模板 YAML 文件中的 **spec.dataVolumeTemplates.spec.sourceRef.name** 下找到。

3. 保存该文件。

9.3.2.3. 启用卷快照引导源

通过在与存储操作系统基础镜像的存储类关联的 **StorageProfile** 中设置参数来启用卷快照引导源。虽然 **DataImportCron** 最初被设计为只维护 PVC 源，但 **VolumeSnapshot** 源会比特定存储类型的 PVC 源更好地扩展。

注意

从单个快照克隆时，在存储配置文件中使用卷快照可以更好地扩展。

先决条件

- 您必须有权访问带有操作系统镜像的卷快照。
- 存储必须支持快照。

流程

1. 运行以下命令，打开与用于置备引导源的存储类对应的存储配置集对象：

```
$ oc edit storageprofile <storage_class>
```

2. 查看 **StorageProfile** 的 **dataImportCronSourceFormat** 规格，以确认虚拟机是否默认使用 PVC 或卷快照。
3. 如果需要，通过将 **dataImportCronSourceFormat** 规格更新为 **snapshot** 来编辑存储配置文件。

存储配置集示例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
# ...
spec:
  dataImportCronSourceFormat: snapshot
```

验证

1. 打开与用于置备引导源的存储类对应的存储配置集对象。

```
$ oc get storageprofile <storage_class>  -oyaml
```

2. 确认 **StorageProfile** 的 **dataImportCronSourceFormat** 规格已设置为 'snapshot'，**DataImportCron** 指向的任何 **DataSource** 对象现在引用卷快照。

现在，您可以使用这些引导源来创建虚拟机。

9.3.3. 为单个引导源禁用自动更新

您可以通过编辑 **HyperConverged** 自定义资源 (CR) 禁用单个引导源的自动更新，无论是自定义还是系统定义。

流程

1. 运行以下命令，在默认编辑器中打开 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 通过编辑 **spec.dataImportCronTemplates** 字段来禁用单个引导源的自动更新。

   自定义引导源

   - 从 **spec.dataImportCronTemplates** 字段删除引导源。在默认情况下，自定义引导源禁用了自动更新。

   系统定义的引导源

   a. 将引导源添加到 **spec.dataImportCronTemplates**。

   注意

   对于系统定义的引导源，默认启用自动更新，但除非添加它们，否则这些引导源不会在 CR 中列出。

   b. 将 **dataimportcrontemplate.kubevirt.io/enable** 注解的值设置为 **'false'**。
   例如：

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      annotations:
        dataimportcrontemplate.kubevirt.io/enable: 'false'
      name: rhel8-image-cron
# ...
```

3. 保存该文件。

9.3.4. 验证引导源的状态

您可以通过查看 **HyperConverged** 自定义资源(CR)来确定引导源是否为系统定义或自定义。

流程

1. 运行以下命令，查看 **HyperConverged** CR 的内容：

```
$ oc get hyperconverged kubevirt-hyperconverged -n openshift-cnv -o yaml
```

输出示例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
# ...
status:
# ...
  dataImportCronTemplates:
  - metadata:
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true"
      name: centos-7-image-cron
    spec:
      garbageCollect: Outdated
      managedDataSource: centos7
      schedule: 55 8/12 * * *
      template:
        metadata: {}
        spec:
          source:
            registry:
              url: docker://quay.io/containerdisks/centos:7-2009
          storage:
            resources:
              requests:
                storage: 30Gi
        status: {}
    status:
      commonTemplate: true (1)
# ...
  - metadata:
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true"
      name: user-defined-dic
    spec:
      garbageCollect: Outdated
      managedDataSource: user-defined-centos-stream8
      schedule: 55 8/12 * * *
      template:
        metadata: {}
        spec:
          source:
            registry:
              pullMethod: node
              url: docker://quay.io/containerdisks/centos-stream:8
          storage:
            resources:
              requests:
                storage: 30Gi
```

```
      status: {}
    status: {} 2
# ...
```

1 表示系统定义的引导源。

2 表示自定义引导源。

2. 通过查看 **status.dataImportCronTemplates.status** 字段来验证引导源的状态。
   - 如果字段包含 **commonTemplate: true**，则它是一个系统定义的引导源。
   - 如果 **status.dataImportCronTemplates.status** 字段的值为 **{}**，则它是一个自定义引导源。

9.4. 为文件系统开销保留 PVC 空间

当您将虚拟机磁盘添加到使用 **Filesystem** 卷模式的持久性卷声明 (PVC) 中时，您必须确保 PVC 中有足够的空间用于虚拟机磁盘和文件系统开销，如元数据。

默认情况下，OpenShift Virtualization 为开销保留 5.5% 的 PVC 空间，从而减少了虚拟机磁盘的可用空间。

您可以通过编辑 **HCO** 对象来配置不同的开销值。您可以在全局范围内更改值，也可以为特定存储类指定值。

9.4.1. 覆盖默认文件系统开销值

通过编辑 **HCO** 对象的 **spec.filesystemOverhead** 属性来更改 OpenShift Virtualization 为文件系统开销保留的持久性卷声明 (PVC) 空间量。

先决条件

- 安装 OpenShift CLI (**oc**)。

流程

1. 运行以下命令，打开 **HCO** 对象进行编辑：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 编辑 **spec.filesystemOverhead** 字段，使用您选择的值填充它们：

```
# ...
spec:
  filesystemOverhead:
    global: "<new_global_value>" 1
    storageClass:
      <storage_class_name>: "<new_value_for_this_storage_class>" 2
```

1 任何还没有设置值的存储类使用的默认文件系统开销百分比。例如，**global: "0.07"** 为文件系统开销保留 PVC 的 7%。

2

指定存储类的文件系统开销百分比。例如，**mystorageclass: "0.04"** 将 **mystorageclass** 存储类中 PVC 的默认开销值改为 4%。

3. 保存并退出编辑器以更新 **HCO** 对象。

验证

- 运行以下命令之一查看 **CDIConfig** 状态并验证您的更改：
  通常验证 **CDIConfig** 的更改：

```
$ oc get cdiconfig -o yaml
```

查看您对 **CDIConfig** 的具体更改：

```
$ oc get cdiconfig -o jsonpath='{.items..status.filesystemOverhead}'
```

9.5. 使用 HOSTPATH 置备程序配置本地存储

您可以使用 hostpath 置备程序(HPP)为虚拟机配置本地存储。

安装 OpenShift Virtualization Operator 时，会自动安装 Hostpath Provisioner Operator。HPP 是一个本地存储置备程序，用于由 Hostpath Provisioner Operator 创建的 OpenShift Virtualization。要使用 HPP，您可以使用基本存储池创建 HPP 自定义资源(CR)。

9.5.1. 使用基本存储池创建 hostpath 置备程序

您可以使用 **storagePools** 小节创建 HPP 自定义资源(CR)，以使用基本存储池配置 hostpath 置备程序(HPP)。存储池指定 CSI 驱动程序使用的名称和路径。

重要

不要在与操作系统相同的分区中创建存储池。否则，操作系统分区可能会被填充到容量中，这会影响性能或导致节点不稳定或不可用。

先决条件

- 在 **spec.storagePools.path** 中指定的目录必须具有读/写访问权限。

流程

1. 使用 **storagePools** 小节创建一个 **hpp_cr.yaml** 文件，如下例所示：

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
  storagePools: 1
  - name: any_name
    path: "/var/myvolumes" 2
```

```
workload:
  nodeSelector:
    kubernetes.io/os: linux
```

1. **storagePools** 小节是一个数组，您可以添加多个条目。
2. 指定此节点路径下的存储池目录。

2. 保存文件并退出。
3. 运行以下命令来创建 HPP：

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

9.5.1.1. 关于创建存储类

当您创建存储类时，您将设置参数，它们会影响属于该存储类的持久性卷(PV)的动态置备。您不能在创建 **StorageClass** 对象后更新其参数。

要使用 hostpath 置备程序(HPP)，您必须使用 **storagePools** 小节为 CSI 驱动程序创建关联的存储类。

注意

虚拟机使用基于本地 PV 的数据卷。本地 PV 与特定节点绑定。虽然磁盘镜像准备供虚拟机消耗，但可能不会将虚拟机调度到之前固定本地存储 PV 的节点。

要解决这个问题，使用 Kubernetes pod 调度程序将持久性卷声明(PVC)绑定到正确的节点上的 PV。通过使用 **volumeBindingMode** 参数设置为 **WaitForFirstConsumer** 的 **StorageClass** 值，PV 的绑定和置备会延迟到 pod 使用 PVC。

9.5.1.2. 使用 storagePools 小节为 CSI 驱动程序创建存储类

要使用 hostpath 置备程序 (HPP)，您必须为 Container Storage Interface (CSI) 驱动程序创建关联的存储类。

当您创建存储类时，您将设置参数，它们会影响属于该存储类的持久性卷(PV)的动态置备。您不能在创建 **StorageClass** 对象后更新其参数。

注意

虚拟机使用基于本地 PV 的数据卷。本地 PV 与特定节点绑定。虽然磁盘镜像准备供虚拟机消耗，但可能不会将虚拟机调度到之前固定本地存储 PV 的节点。

要解决这个问题，使用 Kubernetes pod 调度程序将持久性卷声明(PVC)绑定到正确的节点上的 PV。通过使用 **volumeBindingMode** 参数设置为 **WaitForFirstConsumer** 的 **StorageClass** 值，PV 的绑定和置备会延迟到 pod 使用 PVC。

流程

1. 创建 **storageclass_csi.yaml** 文件来定义存储类：

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
```

```
metadata:
  name: hostpath-csi
provisioner: kubevirt.io.hostpath-provisioner
reclaimPolicy: Delete 1
volumeBindingMode: WaitForFirstConsumer 2
parameters:
  storagePool: my-storage-pool 3
```

1. 两个可能的 **reclaimPolicy** 值为 **Delete** 和 **Retain**。如果没有指定值，则默认值为 **Delete**。
2. **volumeBindingMode** 参数决定何时发生动态置备和卷绑定。指定 **WaitForFirstConsumer**，将持久性卷(PV)的绑定和置备延迟到创建使用持久性卷声明(PVC)的 pod 后。这样可确保 PV 满足 pod 的调度要求。
3. 指定 HPP CR 中定义的存储池名称。

2. 保存文件并退出。
3. 运行以下命令来创建 **StorageClass** 对象：

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

9.5.2. 关于使用 PVC 模板创建的存储池

如果您有单个大持久性卷(PV)，可以通过在 hostpath 置备程序(HPP)自定义资源(CR)中定义 PVC 模板来创建存储池。

使用 PVC 模板创建的存储池可以包含多个 HPP 卷。将 PV 拆分为较小的卷，可为数据分配提供更大的灵活性。

PVC 模板基于 **PersistentVolumeClaim** 对象的 **spec** 小节：

PersistentVolumeClaim 对象示例

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: iso-pvc
spec:
  volumeMode: Block 1
  storageClassName: my-storage-class
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

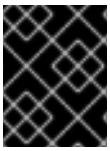
1. 这个值只适用于块卷模式 PV。

您可以使用 HPP CR 中的 **pvcTemplate** 规格来定义存储池。Operator 从包含 **HPP** CSI 驱动程序的每个节点中创建一个 PVC。从 PVC 模板创建的 PVC 使用单个大 PV，允许 HPP 创建较小的动态卷。

您可以将基本存储池与从 PVC 模板中创建的存储池合并。

9.5.2.1. 使用 PVC 模板创建存储池

您可以通过在 HPP 自定义资源(CR)中指定 PVC 模板，为多个 hostpath 置备程序(HPP)卷创建存储池。



重要

不要在与操作系统相同的分区中创建存储池。否则，操作系统分区可能会被填充到容量中，这会影响性能或导致节点不稳定或不可用。

先决条件

- 在 **spec.storagePools.path** 中指定的目录必须具有读/写访问权限。

流程

1. 为 HPP CR 创建 **hpp_pvc_template_pool.yaml** 文件，该文件指定 **storagePools** 小节中的持久性卷(PVC)模板，如下例所示：

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
  storagePools: 1
  - name: my-storage-pool
    path: "/var/myvolumes" 2
    pvcTemplate:
      volumeMode: Block 3
      storageClassName: my-storage-class 4
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 5Gi 5
  workload:
    nodeSelector:
      kubernetes.io/os: linux
```

1. **storagePools** 小节是一个可包含基本和 PVC 模板存储池的数组。
2. 指定此节点路径下的存储池目录。
3. 可选：**volumeMode** 参数可以是 **Block** 或 **Filesystem**，只要它与置备的卷格式匹配。如果没有指定值，则默认为 **Filesystem**。如果 **volumeMode** 是 **Block**，挂载 pod 会在挂载前在块卷中创建一个 XFS 文件系统。
4. 如果省略 **storageClassName** 参数，则使用默认存储类来创建 PVC。如果省略 **storageClassName**，请确保 HPP 存储类不是默认存储类。
5. 您可以指定静态或动态置备的存储。在这两种情况下，确保请求的存储大小适合您要虚拟分割的卷，或者 PVC 无法绑定到大型 PV。如果您使用的存储类使用动态置备的存储，请选择与典型请求大小匹配的分配大小。

2. 保存文件并退出。

3. 运行以下命令，使用存储池创建 HPP：

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

9.6. 启用用户权限跨命名空间克隆数据卷

命名空间的隔离性质意味着用户默认无法在命名空间之间克隆资源。

要让用户将虚拟机克隆到另一个命名空间，具有 **cluster-admin** 角色的用户必须创建新的集群角色。将此集群角色绑定到用户，以便其将虚拟机克隆到目标命名空间。

9.6.1. 创建用于克隆数据卷的 RBAC 资源

创建一个新的集群角色，为 **datavolumes** 资源的所有操作启用权限。

先决条件

- 您必须具有集群管理员特权。

流程

1. 创建 **ClusterRole** 清单：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: <datavolume-cloner> 1
rules:
- apiGroups: ["cdi.kubevirt.io"]
  resources: ["datavolumes/source"]
  verbs: ["*"]
```

1 集群角色的唯一名称。

2. 在集群中创建集群角色：

```
$ oc create -f <datavolume-cloner.yaml> 1
```

1 上一步中创建的 **ClusterRole** 清单的文件名。

3. 创建应用于源和目标命名空间的 **RoleBinding** 清单，并引用上一步中创建的集群角色。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: <allow-clone-to-user> 1
  namespace: <Source namespace> 2
subjects:
- kind: ServiceAccount
```

```
  name: default
  namespace: <Destination namespace> 3
roleRef:
  kind: ClusterRole
  name: datavolume-cloner 4
  apiGroup: rbac.authorization.k8s.io
```

1. 角色绑定的唯一名称。
2. 源数据卷的命名空间。
3. 数据卷克隆到的命名空间。
4. 上一步中创建的集群角色的名称。

4. 在集群中创建角色绑定：

```
$ oc create -f <datavolume-cloner.yaml> 1
```

1. 上一步中创建的 **RoleBinding** 清单的文件名。

9.7. 配置 CDI 来覆盖 CPU 和内存配额

您可以配置 Containerized Data Importer (CDI) 将虚拟机磁盘导入、上传并克隆到命名空间中，这可能受 CPU 和内存资源限制。

9.7.1. 关于命名空间中的 CPU 和内存配额

资源配额 由 **ResourceQuota** 对象定义，对一个命名空间实施限制，该命名空间限制可被该命名空间中资源消耗的计算资源总量。

HyperConverged 自定义资源（CR）定义了 Containerized Data Importer（CDI）的用户配置。CPU 和内存请求和限制值设置为默认值 **0**。这样可确保由 CDI 创建的无需计算资源要求的 Pod 具有默认值，并允许在使用配额限制的命名空间中运行。

启用 **AutoResourceLimits** 功能门时，OpenShift Virtualization 会自动管理 CPU 和内存限值。如果命名空间同时具有 CPU 和内存配额，则内存限值被设置为基本分配的两倍，CPU 限值是针对每个 vCPU 的。

9.7.2. 覆盖 CPU 和内存默认值

通过将 **spec.resourceRequirements.storageWorkloads** 小节添加到 **HyperConverged** 自定义资源（CR），为您的用例修改 CPU 和内存请求和限值的默认设置。

先决条件

- 安装 OpenShift CLI (**oc**)。

流程

1. 运行以下命令来编辑 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 将 **spec.resourceRequirements.storageWorkloads** 小节添加到 CR，根据您的用例设置值。例如：

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  resourceRequirements:
    storageWorkloads:
      limits:
        cpu: "500m"
        memory: "2Gi"
      requests:
        cpu: "250m"
        memory: "1Gi"
```

3. 保存并退出编辑器以更新 **HyperConverged** CR。

9.7.3. 其他资源

- 项目的资源配额

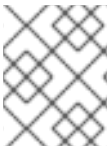
9.8. 准备 CDI 涂销空间

9.8.1. 关于涂销空间

Containerized Data Importer (CDI) 需要涂销空间（临时存储）来完成一些操作，如导入和上传虚拟机镜像。在此过程中，CDI 会提供一个与支持目标数据卷（DV）的 PVC 大小相等的涂销空间 PVC。该涂销空间 PVC 将在操作完成或中止后删除。

您可以在 **HyperConverged** 自定义资源的 **spec.scratchSpaceStorageClass** 字段中定义绑定涂销空间 PVC 的存储类。

如果定义的存储类与集群中的存储类不匹配，则会使用为集群定义的默认存储类。如果没有在集群中定义默认存储类，则会使用置备原始 DV 或 PVC 的存储类。



注意

CDI 需要通过 **file** 卷模式来请求涂销空间，与支持原始数据卷的 PVC 无关。如果 **block** 卷模式支持原始 PVC，则您必须定义一个能够置备 **file** 卷模式 PVC 的 StorageClass。

手动调配
如果没有存储类，CDI 将使用项目中与镜像的大小要求匹配的任何 PVC。如果没有与这些要求匹配的 PVC，则 CDI 导入 Pod 将保持 **Pending** 状态，直至有适当的 PVC 可用或直至超时功能关闭 Pod。

9.8.2. 需要涂销空间的 CDI 操作

类型	原因

类型	原因
registry 导入	CDI 必须下载镜像至涂销空间，并对层进行提取，以查找镜像文件。然后镜像文件传递至 QEMU-IMG 以转换成原始磁盘。
上传镜像	QEMU-IMG 不接受来自 STDIN 的输入。相反，要上传的镜像保存到涂销空间中，然后才可传递至 QEMU-IMG 进行转换。
存档镜像的 HTTP 导入	QEMU-IMG 不知道如何处理 CDI 支持的存档格式。相反，镜像取消存档并保存到涂销空间中，然后再传递至 QEMU-IMG。
经过身份验证的镜像的 HTTP 导入	QEMU-IMG 未充分处理身份验证。相反，镜像保存到涂销空间中并进行身份验证，然后再传递至 QEMU-IMG。
自定义证书的 HTTP 导入	QEMU-IMG 未充分处理 HTTPS 端点的自定义证书。相反，CDI 下载镜像到涂销空间，然后再将文件传递至 QEMU-IMG。

9.8.3. 定义存储类

您可以通过将 **spec.scratchSpaceStorageClass** 字段添加到 **HyperConverged** 自定义资源（CR）来定义 Containerized Data Importer（CDI）在分配涂销空间时使用的存储类。

先决条件

- 安装 OpenShift CLI (**oc**) 。

流程

1. 运行以下命令来编辑 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 将 **spec.scratchSpaceStorageClass** 字段添加到 CR，将值设置为集群中存在的存储类的名称：

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  scratchSpaceStorageClass: "<storage_class>" 1
```

1 如果您没有指定存储类，CDI 将使用正在填充的持久性卷声明的存储类。

3. 保存并退出默认编辑器以更新 **HyperConverged** CR。

9.8.4. CDI 支持的操作列表

此列表针对端点显示内容类型支持的 CDI 操作，以及哪些操作需要涂销空间（scratch space）。

内容类型	HTTP	HTTPS	HTTP 基本身份验证	Registry	上传
KubeVirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* ☐ GZ ☐ XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* ☐ GZ ☐ XZ	✓ RAW* ✓ GZ* ✓ XZ*

✓ 支持的操作

☐ 不支持的操作

* 需要涂销空间

** 如果需要自定义证书颁发机构，则需要涂销空间

9.8.5. 其他资源

- 动态置备

9.9. 对数据卷使用预分配

Containerized Data Importer 可以预先分配磁盘空间，以便在创建数据卷时提高写入性能。

您可以为特定数据卷启用预分配。

9.9.1. 关于预分配

Containerized Data Importer（CDI）可以使用 QEMU 预先分配数据卷模式来提高写入性能。您可以使用预分配模式导入和上传操作，并在创建空白数据卷时使用。

如果启用了预分配，CDI 根据底层文件系统和设备类型使用更好的预分配方法：

fallocate

如果文件系统支持它，CDI 通过使用 **posix_fallocate** 功能（它分配块并将其标记为未初始化），来使用操作系统本身的（**fallocate** 调用来预分配空间。

full

如果无法使用 **fallocate** 模式，则会使用 **full** 模式通过将数据写入底层存储来为镜像分配空间。根据存储位置，所有空分配的空间都可能会为零。

9.9.2. 为数据卷启用预分配

您可以通过在数据卷清单中包含 **spec.preallocation** 字段来为特定数据卷启用预分配。您可以在 web 控制台中或使用 OpenShift CLI (**oc**) 启用预分配模式。

所有 CDI 源类型都支持 Preallocation 模式。

流程

- 指定数据卷清单中的 **spec.preallocation** 字段：

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: preallocated-datavolume
spec:
  source: 1
    registry:
      url: <image_url> 2
  storage:
    resources:
      requests:
        storage: 1Gi
  preallocation: true
# ...
```

1 所有 CDI 源类型都支持预分配。但是，克隆操作会忽略预分配。

2 指定 registry 中数据源的 URL。

9.10. 管理数据卷注解

数据卷（DV）注解允许您管理 pod 行为。您可以将一个或多个注解添加到数据卷，然后将其传播到创建的导入程序 pod。

9.10.1. 示例：数据卷注解

本例演示了如何配置数据卷（DV）注解来控制 importer pod 使用的网络。**v1.multus-cni.io/default-network: bridge-network** 注解会导致 pod 使用名为 **bridge-network** 的 multus 网络作为其默认网络。如果您希望 importer pod 使用集群中的默认网络和从属 multus 网络，请使用 **k8s.v1.cni.cncf.io/networks: <network_name>** 注解。

Multus 网络注解示例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: datavolume-example
  annotations:
    v1.multus-cni.io/default-network: bridge-network 1
# ...
```

1 Multus 网络注解

第 10 章 实时迁移

10.1. 关于实时迁移

实时迁移是在不中断虚拟工作负载的情况下，将正在运行的虚拟机 (VM) 移到集群中的另一节点的过程。默认情况下，实时迁移流量使用传输层安全 (TLS) 加密。

10.1.1. 实时迁移要求

实时迁移有以下要求：

- 集群必须具有 **ReadWriteMany** (RWX) 访问模式的共享存储。
- 集群必须有足够的 RAM 和网络带宽。

注意

您必须确保集群中有足够的内存请求容量来支持节点排空会导致实时迁移。您可以使用以下计算来确定大约所需的备用内存：

> Product of (Maximum number of nodes that can drain in parallel) and (Highest total VM memory request allocations across nodes)

集群中可以并行运行的默认迁移数量为 5。

- 如果虚拟机使用主机模型 CPU，节点必须支持 CPU。
- 强烈建议为实时迁移配置专用的 Multus 网络。专用网络可最小化迁移期间对租户工作负载网络饱和的影响。

10.1.2. 常见实时迁移任务

您可以执行以下实时迁移任务：

- 配置实时迁移设置
- 启动和取消实时迁移
- 在 OpenShift Virtualization web 控制台的 **Migration** 选项卡中监控所有实时迁移的进度。
- 在 web 控制台的 **Metrics** 选项卡中查看虚拟机迁移指标。

10.1.3. 其他资源

- Prometheus 对实时迁移的查询
- VM 迁移调整
- VM 运行策略
- VM 和集群驱除策略

10.2. 配置实时迁移

您可以配置实时迁移设置，以确保迁移过程不会给集群造成大量问题。

您可以配置实时迁移策略，将不同的迁移配置应用到虚拟机组。

10.2.1. 配置实时迁移限制和超时

通过更新位于 **openshift-cnv** 命名空间中的 **HyperConverged** 自定义资源（CR）为集群配置实时迁移限制和超时。

流程

- 编辑 **HyperConverged** CR 并添加必要的实时迁移参数：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

配置文件示例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  liveMigrationConfig:
    bandwidthPerMigration: 64Mi (1)
    completionTimeoutPerGiB: 800 (2)
    parallelMigrationsPerCluster: 5 (3)
    parallelOutboundMigrationsPerNode: 2 (4)
    progressTimeout: 150 (5)
```

1. 每个迁移的带宽限制，其中值为每秒字节数。例如，**2048Mi** 表示 2048 MiB/s。默认：**0**，代表没有限制。
2. 如果迁移未能在此时间内完成则会取消，以每 GiB 内存秒数为单位。例如，如果有 6GiB 内存的虚拟机在 4800 秒内还没有完成，则超时。如果 **Migration Method** 是 **BlockMigration**，则迁移磁盘的大小纳入计算中。
3. 集群中并行运行的迁移数。默认：**5**。
4. 每个节点的最大出站迁移数。默认：**2**。
5. 如果内存复制未能在此时间内取得进展，则会取消迁移，以秒为单位。默认：**150**。

注意

您可以通过删除该键/值对并保存文件来恢复任何 **spec.liveMigrationConfig** 字段的默认值。例如，删除 **progressTimeout: <value>** 以恢复默认的 **progressTimeout: 150**。

10.2.2. 实时迁移策略

您可以创建实时迁移策略，将不同的迁移配置应用到由 VM 或项目标签定义的虚拟机组。

提示

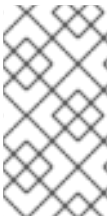
您可以使用 OpenShift Virtualization web 控制台创建实时迁移策略。

10.2.2.1. 使用命令行创建实时迁移策略

您可以使用命令行创建实时迁移策略。kubevirt 使用任意标签组合将实时迁移策略应用到所选虚拟机 (VM)：

- VM 标签，如 **size**, **os**, 或 **gpu**
- 项目标签，如 **priority**, **bandwidth**, 或 **hpc-workload**

要使策略应用到特定的虚拟机组，VM 组的所有标签都必须与策略标签匹配。



注意

如果多个实时迁移策略应用到 VMI，则具有最高匹配标签的策略会优先使用。

如果多个策略满足此条件，则策略按照匹配标签键的字母顺序排序，并且第一个策略具有优先顺序。

流程

1. 编辑要应用实时迁移策略的 VM 对象，并添加对应的 VM 标签。
   a. 打开资源的 YAML 配置：

   ```
   $ oc edit vm <vm_name>
   ```

   b. 调整配置的 **.spec.template.metadata.labels** 部分中所需的标签值。例如，出于迁移策略的目的，将 VM 标记为 **production** VM，添加 **kubevirt.io/environment: production** 行：

   ```
   apiVersion: migrations.kubevirt.io/v1alpha1
   kind: VirtualMachine
   metadata:
     name: <vm_name>
     namespace: default
     labels:
       app: my-app
       environment: production
   spec:
     template:
       metadata:
         labels:
           kubevirt.io/domain: <vm_name>
           kubevirt.io/size: large
           kubevirt.io/environment: production
   # ...
   ```

   c. 保存并退出配置。
2. 使用对应标签配置 **MigrationPolicy** 对象。以下示例配置适用于标记为 **production** 的所有虚拟机的策略：

```
apiVersion: migrations.kubevirt.io/v1alpha1
kind: MigrationPolicy
metadata:
  name: <migration_policy>
spec:
  selectors:
    namespaceSelector: 1
      hpc-workloads: "True"
      xyz-workloads-type: ""
    virtualMachineInstanceSelector: 2
      kubevirt.io/environment: "production"
```

1 指定项目标签。

2 指定 VM 标签。

3. 运行以下命令来创建迁移策略：

```
$ oc create migrationpolicy -f <migration_policy>.yaml
```

10.2.3. 其他资源

- 为实时迁移配置专用的 Multus 网络

10.3. 启动和取消实时迁移

您可以使用 OpenShift Container Platform Web 控制台或命令行启动虚拟机(VM)到另一节点实时迁移。

您可以使用 Web 控制台或命令行取消实时迁移。虚拟机保留在其原始节点上。

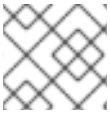
提示

您还可以使用 **virtctl migrate <vm_name>** and **virtctl migrate-cancel <vm_name>** 命令启动和取消实时迁移。

10.3.1. 启动实时迁移

10.3.1.1. 使用 Web 控制台启动实时迁移

您可以使用 OpenShift Container Platform web 控制台将正在运行的虚拟机 (VM) 实时迁移到集群中的不同节点。



注意

Migrate 操作对所有用户可见，但只有集群管理员才能启动实时迁移。

先决条件

- 虚拟机必须是可修改的。
- 如果虚拟机配置了主机模型 CPU，集群必须具有支持 CPU 模型的可用节点。

流程

1. 在 web 控制台中进入到 **Virtualization → VirtualMachines**。
2. 从虚拟机 旁边的 Options 菜单  选择 **Migrate**。
3. 点 **Migrate**。

10.3.1.2. 使用命令行启动实时迁移

您可以使用命令行启动虚拟机(VM)的实时迁移，为虚拟机创建 **VirtualMachineInstanceMigration** 对象。

流程

1. 为您要迁移的虚拟机创建 **VirtualMachineInstanceMigration** 清单：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstanceMigration
metadata:
  name: <migration_name>
spec:
  vmiName: <vm_name>
```

2. 运行以下命令来创建对象：

```
$ oc create -f <migration_name>.yaml
```

VirtualMachineInstanceMigration 对象会触发虚拟机的实时迁移。只要虚拟机实例在运行，该对象便始终存在于集群中，除非手动删除。

验证

- 运行以下命令来获取虚拟机状态：

```
$ oc describe vmi <vm_name> -n <namespace>
```

输出示例

```
# ...
Status:
  Conditions:
    Last Probe Time:       <nil>
    Last Transition Time:  <nil>
    Status:                True
    Type:                  LiveMigratable
  Migration Method:  LiveMigration
  Migration State:
    Completed:                true
    End Timestamp:            2018-12-24T06:19:42Z
    Migration UID:            d78c8962-0743-11e9-a540-fa163e0c69f1
    Source Node:              node2.example.com
    Start Timestamp:          2018-12-24T06:19:35Z
```

```
Target Node:                node1.example.com
Target Node Address:        10.9.0.18:43891
Target Node Domain Detected:  true
```

10.3.2. 取消实时迁移

10.3.2.1. 使用 Web 控制台取消实时迁移

您可以使用 OpenShift Container Platform web 控制台取消虚拟机的实时迁移。

流程

1. 在 web 控制台中进入到 **Virtualization → VirtualMachines**。
2. 在虚拟机的 Options 菜单  中选择 **Cancel Migration**。

10.3.2.2. 使用命令行取消实时迁移

通过删除与迁移关联的 **VirtualMachineInstanceMigration** 对象来取消虚拟机的实时迁移。

流程

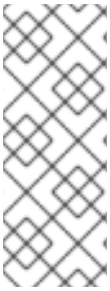
- 删除触发实时迁移的 **VirtualMachineInstanceMigration** 对象，本例中为 **migration-job**：

```
$ oc delete vmim migration-job
```

第 11 章 节点

11.1. 节点维护

节点可以使用 **oc adm** 实用程序或 **NodeMaintenance** 自定义资源 (CR) 置于维护模式。



注意

OpenShift Virtualization 不再提供 **node-maintenance-operator** (NMO)。它被部署为 OpenShift Container Platform Web 控制台中的 **OperatorHub** 的独立 Operator，或使用 OpenShift CLI (**oc**) 部署。

如需有关补救、隔离和维护节点的更多信息，请参阅 Red Hat OpenShift 文档中的工作负载可用性。

重要

虚拟机必须具有一个采用共享 **ReadWriteMany**（RWX）访问模式的 PVC 才能实时迁移。

Node Maintenance Operator 监视是否有新的或删除的 **NodeMaintenance** CR。当检测到新的 **NodeMaintenance** CR 时，不会调度新的工作负载，并且该节点从集群的其余部分中分离。所有可被驱除的 pod 都会从节点上驱除。删除 **NodeMaintenance** CR 时，CR 中引用的节点将可用于新工作负载。

注意

使用 **NodeMaintenance** CR 进行节点维护任务可实现与 **oc adm cordon** 和 **oc adm drain** 命令相同的结果，使用标准 OpenShift Container Platform 自定义资源处理。

11.1.1. 驱除策略

将节点置于维护模式，将节点标记为不可调度，并排空其中的所有虚拟机和 pod。

您可以为虚拟机(VM)或集群配置驱除策略。

VM 驱除策略

VM **LiveMigrate** 驱除策略确保如果节点被置于维护模式或排空，则虚拟机实例(VMI)不会中断。具有驱除策略的 VMI 将实时迁移到另一节点。
您可以使用 命令行为虚拟机 (VM) 配置驱除策略。

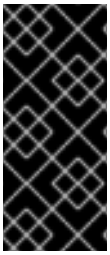
重要

默认驱除策略是 **LiveMigrate**。具有 **LiveMigrate** 驱除策略的不可缓解虚拟机可能会阻止节点排空或阻止基础架构升级，因为虚拟机不会从节点驱除。这种情况会导致迁移处于 **Pending** 或 **Scheduling** 状态，除非您手动关闭虚拟机。

对于不应迁移的虚拟机，您必须将非缓解虚拟机的驱除策略设置为 **LiveMigrateIfPosible**，这不会阻止升级，或设置为 **None**。

集群驱除策略

您可以为集群配置驱除策略，以优先升级工作负载连续性或基础架构升级。



重要

配置集群驱除策略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。

表 11.1. 集群驱除策略

驱除策略	描述	中断工作流	块升级
LiveMigrate[1]	与升级相比，工作负载连续性优先考虑优先级。	否	是[2]
LiveMigrateIfPossible	优先升级工作负载连续性，以确保更新环境。	是	否
None[3]	关闭虚拟机，不使用驱除策略。	是	否

1. 多节点集群的默认驱除策略。
2. 如果虚拟机阻止升级，您必须手动关闭虚拟机。
3. 单节点 OpenShift 的默认驱除策略。

11.1.1.1. 使用命令行配置虚拟机驱除策略

您可以使用命令行为虚拟机配置驱除策略。

重要

默认驱除策略是 **LiveMigrate**。具有 **LiveMigrate** 驱除策略的不可缓解虚拟机可能会阻止节点排空或阻止基础架构升级，因为虚拟机不会从节点驱除。这种情况会导致迁移处于 **Pending** 或 **Scheduling** 状态，除非您手动关闭虚拟机。

对于不应迁移的虚拟机，您必须将非缓解虚拟机的驱除策略设置为 **LiveMigrateIfPosible**，这不会阻止升级，或设置为 **None**。

流程

1. 运行以下命令来编辑 **VirtualMachine** 资源：

```
$ oc edit vm <vm_name> -n <namespace>
```

驱除策略示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: <vm_name>
spec:
```

```
  template:
    spec:
      evictionStrategy: LiveMigrateIfPossible (1)
# ...
```

(1) 指定驱除策略。默认值为 **LiveMigrate**。

2. 重启虚拟机以应用更改：

```
$ virtctl restart <vm_name> -n <namespace>
```

11.1.1.2. 使用命令行配置集群驱除策略

您可以使用命令行为集群配置驱除策略。

重要

配置集群驱除策略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。

流程

1. 运行以下命令来编辑 **hyperconverged** 资源：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

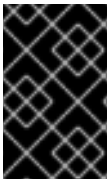
2. 按照以下示例所示设置集群驱除策略：

集群驱除策略示例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  evictionStrategy: LiveMigrate
# ...
```

11.1.2. 运行策略

使用 **spec.running: true** 配置的虚拟机 (VM) 会立即重启。 **spec.runStrategy** 键为决定虚拟机在特定条件下的行为提供了更大的灵活性。



重要

spec.runStrategy 和 **spec.running** 键是互斥的。只能使用其中之一。

带有这两个密钥的虚拟机配置无效。

11.1.2.1. 运行策略

spec.runStrategy 键有四个可能的值：

Always

当在另一个节点上创建虚拟机 (VM) 时，虚拟机实例 (VMI) 始终存在。如果因为某种原因而停止原始 VMI，则会创建新的 VMI。这与 **running: true** 的行为相同。

RerunOnFailure

如果上一个实例失败，则 VMI 会在另一个节点上重新创建。如果虚拟机成功停止，则不会重新创建实例，比如在关闭时。

Manual（手动）

您可以使用 **start**、**stop** 和 **restart** virtctl 客户端命令手动控制 VMI 状态。虚拟机不会自动重启。

Halted

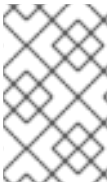
创建虚拟机时不存在 VMI。这与 **running: false** 的行为相同。

virtctl start,**stop** 和 **restart** 命令的不同组合会影响运行策略。

下表描述了虚拟机在状态之间的转换。第一列显示虚拟机的初始运行策略。剩余的列显示 virtctl 命令以及该命令运行后的新运行策略。

表 11.2. 在 virtctl 命令前和后运行策略

初始运行策略	Start	Stop	Restart
Always	-	Halted	Always
RerunOnFailure	-	Halted	RerunOnFailure
Manual	Manual	Manual	Manual
Halted	Always	-	-

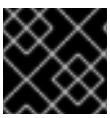


注意

如果使用安装程序置备的基础架构安装的集群中某个节点无法进行机器健康检查且不可用，则带有 **runStrategy: Always** 或 **runStrategy: RerunOnFailure** 的虚拟机会被重新调度到新节点上。

11.1.2.2. 使用命令行配置虚拟机运行策略

您可以使用命令行为虚拟机配置运行策略。



重要

spec.runStrategy 和 **spec.running** 键是互斥的。包含这两个键的值的虚拟机配置无效。

流程

- 运行以下命令来编辑 **VirtualMachine** 资源：

```
$ oc edit vm <vm_name> -n <namespace>
```

run 策略示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  runStrategy: Always
# ...
```

11.1.3. 维护裸机节点

当您在裸机基础架构上部署 OpenShift Container Platform 时，与在云基础架构上部署相比，还需要考虑其他的注意事项。与集群节点被视为临时的云环境中不同，重新置备裸机节点需要大量时间和精力进行维护任务。

当裸机节点出现故障时，例如，如果发生致命内核错误或发生 NIC 卡硬件故障，在修复或替换问题节点时，故障节点上的工作负载需要重启。节点维护模式允许集群管理员安全关闭节点，将工作负载移到集群的其它部分，并确保工作负载不会中断。详细进度和节点状态详情会在维护过程中提供。

11.1.4. 其他资源

- 关于实时迁移

11.2. 为过时的 CPU 型号管理节点标签

只要节点支持 VM CPU 模型和策略，您可以在节点上调度虚拟机（VM）。

11.2.1. 关于过时 CPU 型号的节点标签

OpenShift Virtualization Operator 使用预定义的过时 CPU 型号列表来确保节点只支持调度的虚拟机的有效 CPU 型号。

默认情况下，从为节点生成的标签列表中删除了以下 CPU 型号：

例 11.1. 过时的 CPU 型号

```
"486"
Conroe
athlon
core2duo
coreduo
kvm32
kvm64
n270
pentium
pentium2
pentium3
pentiumpro
phenom
qemu32
qemu64
```

在 **HyperConverged** CR 中无法看到这个预定义列表。您无法从此列表中 *删除* CPU 型号，但您可以通过编辑 **HyperConverged** CR 的 **spec.obsoleteCPUs.cpuModels** 字段来添加到列表中。

11.2.2. 关于 CPU 功能的节点标签

在迭代过程中，从为节点生成的标签列表中删除最小 CPU 模型中的基本 CPU 功能。

例如：

- 一个环境可能有两个支持的 CPU 型号： **Penryn** 和 **Haswell**。
- 如果将 **Penryn** 指定为 **minCPU** 的 CPU 型号， **Penryn** 的每个基本 CPU 功能都会与 **Haswell** 支持的 CPU 功能列表进行比较。

例 11.2. Penryn支持的 CPU 功能

```
apic
clflush
cmov
cx16
cx8
de
fpu
fxsr
lahf_lm
lm
mca
mce
mmx
msr
mtrr
nx
pae
pat
pge
pni
pse
pse36
sep
sse
sse2
sse4.1
ssse3
syscall
tsc
```

例 11.3. Haswell支持的 CPU 功能

```
aes
apic
avx
avx2
bmi1
```

```
bmi2
clflush
cmov
cx16
cx8
de
erms
fma
fpu
fsgsbase
fxsr
hle
invpcid
lahf_lm
lm
mca
mce
mmx
movbe
msr
mtrr
nx
pae
pat
pcid
pclmuldq
pge
pni
popcnt
pse
pse36
rdtscp
rtm
sep
smep
sse
sse2
sse4.1
sse4.2
ssse3
syscall
tsc
tsc-deadline
x2apic
xsave
```

- 如果 **Penryn** 和 **Haswell** 都支持特定的 CPU 功能，则不会为该功能创建一个标签。为仅受 **Haswell** 支持且不受 **Penryn** 支持的 CPU 功能生成标签。

例 11.4. 迭代后为 CPU 功能创建的节点标签

```
aes
avx
avx2
bmi1
```

```
bmi2
erms
fma
fsgsbase
hle
invpcid
movbe
pcid
pclmuldq
popcnt
rdtscp
rtm
sse4.2
tsc-deadline
x2apic
xsave
```

11.2.3. 配置过时的 CPU 型号

您可以通过编辑 **HyperConverged** 自定义资源（CR）来配置过时的 CPU 型号列表。

流程

- 编辑 **HyperConverged** 自定义资源，在 **obsoleteCPUs** 阵列中指定过时的 CPU 型号。例如：

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  obsoleteCPUs:
    cpuModels: 1
      - "<obsolete_cpu_1>"
      - "<obsolete_cpu_2>"
    minCPUModel: "<minimum_cpu_model>" 2
```

1. 将 **cpuModels** 数组中的示例值替换为过时的 CPU 型号。您指定的任何值都会添加到预定义的过时 CPU 型号列表中。预定义的列表在 CR 中不可见。
2. 使用您要用于基本 CPU 功能的最低 CPU 型号替换这个值。如果没有指定值，则默认使用 **Penryn**。

11.3. 防止节点协调

使用 **skip-node** 注解来防止 **node-labeller** 协调节点。

11.3.1. 使用 skip-node 注解

如果您希望 **node-labeller** 跳过节点，请使用 **oc** CLI 注解该节点。

先决条件

- 已安装 OpenShift CLI（**oc**）。

流程

- 运行以下命令来注解您要跳过的节点：

```
$ oc annotate node <node_name> node-labeller.kubevirt.io/skip-node=true 1
```

1 将 **<node_name>** 替换为要跳过的相关节点的名称。

在节点注解被删除或设置为 false 后，协调会在下一个周期中恢复。

11.3.2. 其他资源

- 为过时的 CPU 型号管理节点标签

11.4. 删除失败的节点以触发虚拟机故障切换

如果节点失败，且集群中没有部署 节点健康检查，带有 **runStrategy: Always** 配置的虚拟机(VM)不会自动重新定位到健康的节点。

11.4.1. 先决条件

- 运行虚拟机的节点具有 **NotReady** 条件。
- 在故障节点中运行的虚拟机的 **runStrategy** 设置为 **Always**。
- 已安装 OpenShift CLI(**oc**)。

11.4.2. 从裸机集群中删除节点

当您使用 CLI 删除节点时，节点对象会从 Kubernetes 中删除，但该节点上存在的 pod 不会被删除。任何未由复制控制器支持的裸机 pod 都无法从 OpenShift Container Platform 访问。由复制控制器支持的 Pod 会重新调度到其他可用的节点。您必须删除本地清单 pod。

流程

通过完成以下步骤，从裸机上运行的 OpenShift Container Platform 集群中删除节点：

1. 将节点标记为不可调度：

```
$ oc adm cordon <node_name>
```

2. 排空节点上的所有 pod：

```
$ oc adm drain <node_name> --force=true
```

如果节点离线或者无响应，此步骤可能会失败。即使节点没有响应，它仍然在运行写入共享存储的工作负载。为了避免数据崩溃，请在进行操作前关闭物理硬件。

3. 从集群中删除节点：

-

```
$ oc delete node <node_name>
```

虽然节点对象现已从集群中删除，但它仍然可在重启后或 kubelet 服务重启后重新加入集群。要永久删除该节点及其所有数据，您必须弃用该节点。

4. 如果您关闭了物理硬件，请重新打开它以便节点可以重新加入集群。

11.4.3. 验证虚拟机故障切换

在不健康节点上终止所有资源后，会为每个重新定位的虚拟机在健康的节点上自动创建新虚拟机实例（VMI）。要确认已创建了 VMI，使用 **oc** CLI 查看所有 VMI。

11.4.3.1. 使用 CLI 列出所有虚拟机实例

您可以使用 **oc** 命令行界面（CLI）列出集群中的所有虚拟机实例（VMI），包括独立 VMI 和虚拟机拥有的实例。

流程

- 运行以下命令列出所有 VMI：

```
$ oc get vmis -A
```

第 12 章 监控

12.1. 监控概述

您可以使用以下工具监控集群和虚拟机 (VM) 的健康状况：

监控 OpenShift Virtualization 虚拟机健康状态

在 web 控制台中查看 OpenShift Virtualization 环境的整体健康状况，进入到 OpenShift Container Platform Web 控制台中的 **Home → Overview** 页面。**Status** 卡根据警报和条件显示 OpenShift Virtualization 的整体健康状况。

OpenShift Container Platform 集群检查框架

使用 OpenShift Container Platform 集群检查框架在集群中运行自动测试，以检查以下条件：

- 附加到二级网络接口的两个虚拟机之间的网络连接和延迟
- 运行带有零数据包丢失的 Data Plane Development Kit (DPDK) 工作负载的虚拟机
- 为 OpenShift Virtualization 配置集群存储

Prometheus 对虚拟资源的查询

查询 vCPU、网络、存储和客户机内存交换使用情况和实时迁移进度。

VM 自定义指标

配置 **node-exporter** 服务，以公开内部虚拟机指标和进程。

VM 健康检查

为虚拟机配置就绪度、存活度和客户机代理 ping 探测和 watchdog。

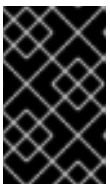
Runbooks

诊断并解决在 OpenShift Container Platform Web 控制台中触发 OpenShift Virtualization 警报的问题。

12.2. OPENSHIFT VIRTUALIZATION 集群检查框架

OpenShift Virtualization 包括以下预定义的检查，可用于集群维护和故障排除：

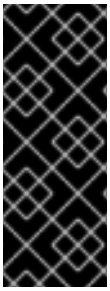
- 延迟检查，验证附加到二级网络接口的两个虚拟机(VM)之间的网络连接和测量延迟。



重要

在运行延迟检查前，您必须首先在集群节点上创建一个桥接接口，以便将虚拟机的二级接口连接到节点上的任何接口。如果您没有创建桥接接口，虚拟机不会启动，作业会失败。

- 存储检查，验证集群存储是否为 OpenShift Virtualization 的最佳配置。
- DPDK 检查，验证节点是否可以运行具有零数据包丢失的 Data Plane Development Kit (DPDK) 工作负载的虚拟机。



重要

OpenShift Virtualization 集群检查框架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。

12.2.1. 关于 OpenShift Virtualization 集群检查框架

checkup 是一个自动测试工作负载，可让您验证特定的集群功能是否按预期工作。集群检查框架使用原生 Kubernetes 资源来配置和执行检查。

通过使用预定义的检查，集群管理员和开发人员可以提高集群可维护性，排除意外行为，最小化错误并节省时间。还可以检查结果并与专家分享，以进一步进行分析。供应商可以编写和发布它们所提供的功能或服务的检查，并验证其客户环境是否已正确配置。

在现有命名空间中运行预定义的检查涉及为检查设置服务帐户，为服务帐户创建 **Role** 和 **RoleBinding** 对象，启用检查权限，以及创建输入配置映射和检查作业。您可以多次运行检查。

重要

您必须始终：

- 在应用之前，验证检查镜像是否来自可信源。
- 在创建 **Role** 和 **RoleBinding** 对象前，查看检查权限。

12.2.2. 使用 Web 控制台运行检查

使用 Web 控制台第一次运行检查时，请使用以下步骤。如需额外的检查，在 checkup 选项卡中点 **Run checkup**，然后从下拉菜单中选择适当的检查。

12.2.2.1. 使用 Web 控制台运行延迟检查

运行延迟检查以验证网络连接并测量附加到二级网络接口的两个虚拟机之间的延迟。

先决条件

- 您必须将 **NetworkAttachmentDefinition** 添加到命名空间。

流程

1. 在 web 控制台中进入到 **Virtualization → Checkups**。
2. 点 **Network latency** 选项卡。
3. 点 **Install permissions**。
4. 点 **Run checkup**。
5. 在 **Name** 字段中输入检查的名称。
6. 从下拉菜单中选择 **NetworkAttachmentDefinition**。

7. 可选：在 **Sample duration (seconds)** 字段中为延迟示例设置一个持续时间。
8. 可选：通过启用 **Set maximum desired latency (milliseconds)** 并定义时间间隔来定义最大延迟时间间隔。
9. 可选：通过启用 **选择节点** 并指定**源节点**和**目标节点**来针对特定的节点。
10. 点 **Run**。

您可以在 **Latency checkup** 选项卡的 **Checkups** 列表中查看延迟检查的状态。点检查的名称以获取更多详细信息。

12.2.2.2. 使用 Web 控制台运行存储检查

运行存储检查，以验证存储是否为虚拟机正常工作。

流程

1. 在 web 控制台中进入到 **Virtualization → Checkups**。
2. 点 **Storage** 选项卡。
3. 点 **Install permissions**。
4. 点 **Run checkup**。
5. 在 **Name** 字段中输入检查的名称。
6. 在 **Timeout (minutes)** 字段中输入检查的超时值。
7. 点 **Run**。

您可以在 **Storage** 选项卡的 **Checkups** 列表中查看存储检查的状态。点检查的名称以获取更多详细信息。

12.2.3. 使用命令行运行检查

第一次使用命令行运行检查时，请使用以下步骤。

12.2.3.1. 使用命令行运行延迟检查

您可以使用预定义的检查来验证附加到二级网络接口的两个虚拟机 (VM) 之间的网络连接和测量延迟。延迟检查使用 ping 工具。

您可以执行以下步骤来运行延迟检查：

1. 创建服务帐户、角色和 rolebindings，以便为延迟检查提供集群访问权限。
2. 创建配置映射以提供运行检查并存储结果的输入。
3. 创建用于运行检查的作业。
4. 查看配置映射中的结果。
5. 可选： 要重新运行检查，请删除现有的配置映射和作业，然后创建新的配置映射和作业。

6. 完成后，删除延迟检查资源。

先决条件

- 已安装 OpenShift CLI（**oc**）。
- 集群至少有两个 worker 节点。
- 为命名空间配置了网络附加定义。

流程

1. 为延迟检查创建一个 **ServiceAccount**、**Role** 和 **RoleBinding** 清单：

例 12.1. 角色清单文件示例

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: vm-latency-checkup-sa
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kubevirt-vm-latency-checker
rules:
- apiGroups: ["kubevirt.io"]
  resources: ["virtualmachineinstances"]
  verbs: ["get", "create", "delete"]
- apiGroups: ["subresources.kubevirt.io"]
  resources: ["virtualmachineinstances/console"]
  verbs: ["get"]
- apiGroups: ["k8s.cni.cncf.io"]
  resources: ["network-attachment-definitions"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kubevirt-vm-latency-checker
subjects:
- kind: ServiceAccount
  name: vm-latency-checkup-sa
roleRef:
  kind: Role
  name: kubevirt-vm-latency-checker
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kiagnose-configmap-access
rules:
- apiGroups: [ "" ]
  resources: [ "configmaps" ]
```

```
  verbs: ["get", "update"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kiagnose-configmap-access
subjects:
- kind: ServiceAccount
  name: vm-latency-checkup-sa
roleRef:
  kind: Role
  name: kiagnose-configmap-access
  apiGroup: rbac.authorization.k8s.io
```

2. 应用 **ServiceAccount**、**Role** 和 **RoleBinding** 清单：

```
$ oc apply -n <target_namespace> -f <latency_sa_roles_rolebinding>.yaml 1
```

1 **<target_namespace>** 是要运行检查的命名空间。这必须是 **NetworkAttachmentDefinition** 对象所在的现有命名空间。

3. 创建包含检查的输入参数的 **ConfigMap** 清单：

输入配置映射示例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kubevirt-vm-latency-checkup-config
  labels:
    kiagnose/checkup-type: kubevirt-vm-latency
data:
  spec.timeout: 5m
  spec.param.networkAttachmentDefinitionNamespace: <target_namespace>
  spec.param.networkAttachmentDefinitionName: "blue-network" 1
  spec.param.maxDesiredLatencyMilliseconds: "10" 2
  spec.param.sampleDurationSeconds: "5" 3
  spec.param.sourceNode: "worker1" 4
  spec.param.targetNode: "worker2" 5
```

1 **NetworkAttachmentDefinition** 对象的名称。

2 可选：虚拟机之间所需最大延迟（以毫秒为单位）。如果测量的延迟超过这个值，则检查会失败。

3 可选：延迟检查的持续时间，以秒为单位。

4 可选：指定时，延迟从此节点测量到目标节点。如果指定了源节点，**spec.param.targetNode** 字段将无法为空。

5 可选：指定时，延迟从源节点测量到这个节点。

4. 应用目标命名空间中的配置映射清单：

```
$ oc apply -n <target_namespace> -f <latency_config_map>.yaml
```

5. 创建一个 **Job** 清单以运行检查：

任务清单示例

```
apiVersion: batch/v1
kind: Job
metadata:
  name: kubevirt-vm-latency-checkup
  labels:
    kiagnose/checkup-type: kubevirt-vm-latency
spec:
  backoffLimit: 0
  template:
    spec:
      serviceAccountName: vm-latency-checkup-sa
      restartPolicy: Never
      containers:
        - name: vm-latency-checkup
          image: registry.redhat.io/container-native-virtualization/vm-network-latency-checkup-
rhel9:v4.16.0
          securityContext:
            allowPrivilegeEscalation: false
            capabilities:
              drop: ["ALL"]
            runAsNonRoot: true
            seccompProfile:
              type: "RuntimeDefault"
          env:
            - name: CONFIGMAP_NAMESPACE
              value: <target_namespace>
            - name: CONFIGMAP_NAME
              value: kubevirt-vm-latency-checkup-config
            - name: POD_UID
              valueFrom:
                fieldRef:
                  fieldPath: metadata.uid
```

6. 应用 **Job** 清单：

```
$ oc apply -n <target_namespace> -f <latency_job>.yaml
```

7. 等待作业完成：

```
$ oc wait job kubevirt-vm-latency-checkup -n <target_namespace> --for condition=complete --
timeout 6m
```

8. 运行以下命令，检查延迟检查的结果。如果测量的最大延迟大于 **spec.param.maxDesiredLatencyMilliseconds** 属性的值，则检查会失败并返回错误。

```
$ oc get configmap kubevirt-vm-latency-checkup-config -n <target_namespace> -o yaml
```

输出配置映射（成功）示例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kubevirt-vm-latency-checkup-config
  namespace: <target_namespace>
  labels:
    kiagnose/checkup-type: kubevirt-vm-latency
data:
  spec.timeout: 5m
  spec.param.networkAttachmentDefinitionNamespace: <target_namespace>
  spec.param.networkAttachmentDefinitionName: "blue-network"
  spec.param.maxDesiredLatencyMilliseconds: "10"
  spec.param.sampleDurationSeconds: "5"
  spec.param.sourceNode: "worker1"
  spec.param.targetNode: "worker2"
  status.succeeded: "true"
  status.failureReason: ""
  status.completionTimestamp: "2022-01-01T09:00:00Z"
  status.startTimestamp: "2022-01-01T09:00:07Z"
  status.result.avgLatencyNanoSec: "177000"
  status.result.maxLatencyNanoSec: "244000" (1)
  status.result.measurementDurationSec: "5"
  status.result.minLatencyNanoSec: "135000"
  status.result.sourceNode: "worker1"
  status.result.targetNode: "worker2"
```

(1) 以纳秒为单位的最大测量延迟。

9. 可选：要在检查失败时查看详细作业日志，请使用以下命令：

```
$ oc logs job.batch/kubevirt-vm-latency-checkup -n <target_namespace>
```

10. 运行以下命令删除之前创建的作业和配置映射：

```
$ oc delete job -n <target_namespace> kubevirt-vm-latency-checkup
```

```
$ oc delete config-map -n <target_namespace> kubevirt-vm-latency-checkup-config
```

11. 可选：如果您不计划运行另一个检查，请删除角色清单：

```
$ oc delete -f <latency_sa_roles_rolebinding>.yaml
```

12.2.3.2. 使用命令行运行存储检查

使用预定义的检查来验证 OpenShift Container Platform 集群存储是否配置为运行 OpenShift Virtualization 工作负载。

先决条件

- 已安装 OpenShift CLI(**oc**)。

- 集群管理员为存储检查服务帐户和命名空间创建了所需的 **cluster-reader** 权限，如下例所示：

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kubevirt-storage-checkup-clustereader
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-reader
subjects:
- kind: ServiceAccount
  name: storage-checkup-sa
  namespace: <target_namespace> (1)
```

(1) 运行检查的命名空间。

流程

1. 为存储检查创建一个 **ServiceAccount**、**Role** 和 **RoleBinding** 清单文件：

例 12.2. 服务帐户、角色和 rolebinding 清单示例

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: storage-checkup-sa
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: storage-checkup-role
rules:
  - apiGroups: [ "" ]
    resources: [ "configmaps" ]
    verbs: ["get", "update"]
  - apiGroups: [ "kubevirt.io" ]
    resources: [ "virtualmachines" ]
    verbs: [ "create", "delete" ]
  - apiGroups: [ "kubevirt.io" ]
    resources: [ "virtualmachineinstances" ]
    verbs: [ "get" ]
  - apiGroups: [ "subresources.kubevirt.io" ]
    resources: [ "virtualmachineinstances/addvolume",
"virtualmachineinstances/removevolume" ]
    verbs: [ "update" ]
  - apiGroups: [ "kubevirt.io" ]
    resources: [ "virtualmachineinstancemigrations" ]
    verbs: [ "create" ]
  - apiGroups: [ "cdi.kubevirt.io" ]
    resources: [ "datavolumes" ]
    verbs: [ "create", "delete" ]
  - apiGroups: [ "" ]
    resources: [ "persistentvolumeclaims" ]
```

```
    verbs: [ "delete" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: storage-checkup-role
subjects:
  - kind: ServiceAccount
    name: storage-checkup-sa
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: storage-checkup-role
```

2. 在目标命名空间中应用 **ServiceAccount**、**Role** 和 **RoleBinding** 清单：

```
$ oc apply -n <target_namespace> -f <storage_sa_roles_rolebinding>.yaml
```

3. 创建 **ConfigMap** 和 **Job** 清单文件。配置映射包含检查作业的输入参数。

输入配置映射和作业清单示例

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: storage-checkup-config
  namespace: $CHECKUP_NAMESPACE
data:
  spec.timeout: 10m
  spec.param.storageClass: ocs-storagecluster-ceph-rbd-virtualization
  spec.param.vmiTimeout: 3m
---
apiVersion: batch/v1
kind: Job
metadata:
  name: storage-checkup
  namespace: $CHECKUP_NAMESPACE
spec:
  backoffLimit: 0
  template:
    spec:
      serviceAccount: storage-checkup-sa
      restartPolicy: Never
      containers:
        - name: storage-checkup
          image: quay.io/kiagnose/kubevirt-storage-checkup:main
          imagePullPolicy: Always
          env:
            - name: CONFIGMAP_NAMESPACE
              value: $CHECKUP_NAMESPACE
            - name: CONFIGMAP_NAME
              value: storage-checkup-config
```

4. 应用目标命名空间中的 **ConfigMap** 和 **Job** 清单文件来运行检查：

```
$ oc apply -n <target_namespace> -f <storage_configmap_job>.yaml
```

5. 等待作业完成：

```
$ oc wait job storage-checkup -n <target_namespace> --for condition=complete --timeout
10m
```

6. 运行以下命令，查看检查的结果：

```
$ oc get configmap storage-checkup-config -n <target_namespace> -o yaml
```

输出配置映射（成功）示例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: storage-checkup-config
  labels:
    kiagnose/checkup-type: kubevirt-storage
data:
  spec.timeout: 10m
  status.succeeded: "true" 1
  status.failureReason: "" 2
  status.startTimestamp: "2023-07-31T13:14:38Z" 3
  status.completionTimestamp: "2023-07-31T13:19:41Z" 4
  status.result.cnvVersion: 4.16.2 5
  status.result.defaultStorageClass: trident-nfs 6
  status.result.goldenImagesNoDataSource: <data_import_cron_list> 7
  status.result.goldenImagesNotUpToDate: <data_import_cron_list> 8
  status.result.ocpVersion: 4.16.0 9
  status.result.pvcBound: "true" 10
  status.result.storageProfileMissingVolumeSnapshotClass: <storage_class_list> 11
  status.result.storageProfilesWithEmptyClaimPropertySets: <storage_profile_list> 12
  status.result.storageProfilesWithSmartClone: <storage_profile_list> 13
  status.result.storageProfilesWithSpecClaimPropertySets: <storage_profile_list> 14
  status.result.storageProfilesWithRWX: |-
    ocs-storagecluster-ceph-rbd
    ocs-storagecluster-ceph-rbd-virtualization
    ocs-storagecluster-cephfs
    trident-iscsi
    trident-minio
    trident-nfs
    windows-vms
  status.result.vmBootFromGoldenImage: VMI "vmi-under-test-dhkb8" successfully booted
  status.result.vmHotplugVolume: |-
    VMI "vmi-under-test-dhkb8" hotplug volume ready
    VMI "vmi-under-test-dhkb8" hotplug volume removed
  status.result.vmLiveMigration: VMI "vmi-under-test-dhkb8" migration completed
```

```
status.result.vmVolumeClone: 'DV cloneType: "csi-clone"'
status.result.vmsWithNonVirtRbdStorageClass: <vm_list> 15
status.result.vmsWithUnsetEfsStorageClass: <vm_list> 16
```

1. 指定检查是否成功(**true**)或不是 (**false**)。
2. 检查失败时失败的原因。
3. 检查启动时的时间，使用 RFC 3339 时间格式。
4. 检查完成后的时间，使用 RFC 3339 时间格式。
5. OpenShift Virtualization 版本。
6. 指定是否有默认存储类。
7. 其数据源未就绪的金级镜像列表。
8. 数据导入 cron 不是最新的金级镜像列表。
9. OpenShift Container Platform 版本。
10. 指定置备程序是否创建并绑定了 10Mi 的 PVC。
11. 使用基于快照的克隆的存储配置集列表，但缺少 VolumeSnapshotClass。
12. 带有未知置备程序的存储配置集列表。
13. 带有智能克隆支持的存储配置文件列表(CSI/snapshot)。
14. 存储配置集 spec-overriden claimPropertySets 列表。
15. 当虚拟化存储类存在时，使用 Ceph RBD 存储类的虚拟机列表。
16. 使用 Elastic File Store (EFS) 存储类的虚拟机列表，其中 GID 和 UID 没有在存储类中设置。

7. 运行以下命令删除之前创建的作业和配置映射：

```
$ oc delete job -n <target_namespace> storage-checkup
```

```
$ oc delete config-map -n <target_namespace> storage-checkup-config
```

8. 可选：如果您不计划运行另一个检查，请删除 **ServiceAccount**、**Role** 和 **RoleBinding** 清单：

```
$ oc delete -f <storage_sa_roles_rolebinding>.yaml
```

12.2.3.3. 使用命令行运行 DPDK 检查

使用预定义的检查来验证 OpenShift Container Platform 集群节点是否可以运行带有零数据包丢失的 Data Plane Development Kit (DPDK) 工作负载的虚拟机 (VM)。DPDK 检查在流量生成器和运行测试 DPDK 应用程序的虚拟机之间运行流量。

您可以执行以下步骤来运行 DPDK 检查：

1. 为 DPDK 检查创建服务帐户、角色和角色绑定。
2. 创建配置映射以提供运行检查并存储结果的输入。
3. 创建用于运行检查的作业。
4. 查看配置映射中的结果。
5. 可选：要重新运行检查，请删除现有的配置映射和作业，然后创建新的配置映射和作业。
6. 完成后，删除 DPDK 检查资源。

先决条件

- 已安装 OpenShift CLI(**oc**)。
- 集群被配置为运行 DPDK 应用程序。
- 该项目被配置为运行 DPDK 应用程序。

流程

1. 为 DPDK 检查创建 **ServiceAccount**、**Role** 和 **RoleBinding** 清单：

例 12.3. 服务帐户、角色和 rolebinding 清单文件示例

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dpdk-checkup-sa
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kiagnose-configmap-access
rules:
  - apiGroups: [ "" ]
    resources: [ "configmaps" ]
    verbs: [ "get", "update" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kiagnose-configmap-access
subjects:
  - kind: ServiceAccount
    name: dpdk-checkup-sa
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kiagnose-configmap-access
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kubevirt-dpdk-checker
```

```
  rules:
    - apiGroups: [ "kubevirt.io" ]
      resources: [ "virtualmachineinstances" ]
      verbs: [ "create", "get", "delete" ]
    - apiGroups: [ "subresources.kubevirt.io" ]
      resources: [ "virtualmachineinstances/console" ]
      verbs: [ "get" ]
    - apiGroups: [ "" ]
      resources: [ "configmaps" ]
      verbs: [ "create", "delete" ]
  ---
  apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    name: kubevirt-dpdk-checker
  subjects:
    - kind: ServiceAccount
      name: dpdk-checkup-sa
  roleRef:
    apiGroup: rbac.authorization.k8s.io
    kind: Role
    name: kubevirt-dpdk-checker
```

2. 应用 **ServiceAccount**、**Role** 和 **RoleBinding** 清单：

```
$ oc apply -n <target_namespace> -f <dpdk_sa_roles_rolebinding>.yaml
```

3. 创建包含检查的输入参数的 **ConfigMap** 清单：

输入配置映射示例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: dpdk-checkup-config
  labels:
    kiagnose/checkup-type: kubevirt-dpdk
data:
  spec.timeout: 10m
  spec.param.networkAttachmentDefinitionName: <network_name> (1)
  spec.param.trafficGenContainerDiskImage: "quay.io/kiagnose/kubevirt-dpdk-checkup-traffic-gen:v0.4.0 (2)
  spec.param.vmUnderTestContainerDiskImage: "quay.io/kiagnose/kubevirt-dpdk-checkup-vm:v0.4.0" (3)
```

(1) **NetworkAttachmentDefinition** 对象的名称。

(2) 流量生成器的容器磁盘镜像。在本例中，镜像是从上游 Project Quay Container Registry 中拉取的。

(3) 测试中的虚拟机的容器磁盘镜像。在本例中，镜像是从上游 Project Quay Container Registry 中拉取的。

4. 应用目标命名空间中的 **ConfigMap** 清单：

```
$ oc apply -n <target_namespace> -f <dpdk_config_map>.yaml
```

5. 创建一个 **Job** 清单以运行检查：

任务清单示例

```
apiVersion: batch/v1
kind: Job
metadata:
  name: dpdk-checkup
  labels:
    kiagnose/checkup-type: kubevirt-dpdk
spec:
  backoffLimit: 0
  template:
    spec:
      serviceAccountName: dpdk-checkup-sa
      restartPolicy: Never
      containers:
        - name: dpdk-checkup
          image: registry.redhat.io/container-native-virtualization/kubevirt-dpdk-checkup-
rhel9:v4.16.0
          imagePullPolicy: Always
          securityContext:
            allowPrivilegeEscalation: false
            capabilities:
              drop: ["ALL"]
            runAsNonRoot: true
            seccompProfile:
              type: "RuntimeDefault"
          env:
            - name: CONFIGMAP_NAMESPACE
              value: <target-namespace>
            - name: CONFIGMAP_NAME
              value: dpdk-checkup-config
            - name: POD_UID
              valueFrom:
                fieldRef:
                  fieldPath: metadata.uid
```

6. 应用 **Job** 清单：

```
$ oc apply -n <target_namespace> -f <dpdk_job>.yaml
```

7. 等待作业完成：

```
$ oc wait job dpdk-checkup -n <target_namespace> --for condition=complete --timeout 10m
```

8. 运行以下命令，查看检查的结果：

```
$ oc get configmap dpdk-checkup-config -n <target_namespace> -o yaml
```

输出配置映射（成功）示例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: dpdk-checkup-config
  labels:
    kiagnose/checkup-type: kubevirt-dpdk
data:
  spec.timeout: 10m
  spec.param.NetworkAttachmentDefinitionName: "dpdk-network-1"
  spec.param.trafficGenContainerDiskImage: "quay.io/kiagnose/kubevirt-dpdk-checkup-traffic-gen:v0.4.0"
  spec.param.vmUnderTestContainerDiskImage: "quay.io/kiagnose/kubevirt-dpdk-checkup-vm:v0.4.0"
  status.succeeded: "true" (1)
  status.failureReason: "" (2)
  status.startTimestamp: "2023-07-31T13:14:38Z" (3)
  status.completionTimestamp: "2023-07-31T13:19:41Z" (4)
  status.result.trafficGenSentPackets: "480000000" (5)
  status.result.trafficGenOutputErrorPackets: "0" (6)
  status.result.trafficGenInputErrorPackets: "0" (7)
  status.result.trafficGenActualNodeName: worker-dpdk1 (8)
  status.result.vmUnderTestActualNodeName: worker-dpdk2 (9)
  status.result.vmUnderTestReceivedPackets: "480000000" (10)
  status.result.vmUnderTestRxDroppedPackets: "0" (11)
  status.result.vmUnderTestTxDroppedPackets: "0" (12)
```

1. 指定检查是否成功(**true**)或不是 (**false**)。
2. 检查失败时失败的原因。
3. 检查启动时的时间，使用 RFC 3339 时间格式。
4. 检查完成后的时间，使用 RFC 3339 时间格式。
5. 从流量生成器发送的数据包数量。
6. 从流量生成器发送的错误数据包数量。
7. 流量生成器接收的错误数据包数量。
8. 调度流量生成器虚拟机的节点。
9. 调度测试过程中虚拟机的节点。
10. 虚拟机在测试下接收的数据包数量。
11. DPDK 应用程序丢弃的入口流量数据包。
12. 从 DPDK 应用程序丢弃的出口流量数据包。

9. 运行以下命令删除之前创建的作业和配置映射：

```
$ oc delete job -n <target_namespace> dpdk-checkup
```

```
$ oc delete config-map -n <target_namespace> dpdk-checkup-config
```

10. 可选：如果您不计划运行另一个检查，请删除 **ServiceAccount**、**Role** 和 **RoleBinding** 清单：

```
$ oc delete -f <dpdk_sa_roles_rolebinding>.yaml
```

12.2.3.3.1. DPDK 检查配置映射参数

下表显示了在运行集群 DPDK 就绪度检查时，您可以在输入 **ConfigMap** 清单的 **data** 小节中设置的强制参数和可选参数：

表 12.1. DPDK 检查配置映射输入参数

参数	描述	是必须的
spec.timeout	检查失败前的时间（以分钟为单位）。	True
spec.param.networkAttachmentDefinitionName	连接 SR-IOV NIC 的 **NetworkAttachmentDefinition** 对象的名称。	True
spec.param.trafficGenContainerDiskImage	流量生成器的容器磁盘镜像。	True
spec.param.trafficGenTargetNodeName	在其中调度流量生成器虚拟机的节点。节点应配置为允许 DPDK 流量。	False
spec.param.trafficGenPacketsPerSecond	每秒数据包数量，单位为(k)或500万(m)。默认值为 8m。	False
spec.param.vmUnderTestContainerDiskImage	测试中的虚拟机的容器磁盘镜像。	True
spec.param.vmUnderTestTargetNodeName	在其中调度测试虚拟机的节点。节点应配置为允许 DPDK 流量。	False
spec.param.testDuration	流量生成器运行的持续时间（以分钟为单位）。默认值为 5 分钟。	False
spec.param.portBandwidthGbps	SR-IOV NIC 的最大带宽。默认值为 10Gbps。	False
spec.param.verbose	当设置为 **true** 时，它会增加检查日志的详细程度。默认值为 **false**。	False

12.2.3.3.2. 为 RHEL 虚拟机构建容器磁盘镜像

您可以使用 **qcow2** 格式构建自定义 Red Hat Enterprise Linux (RHEL) 8 OS 镜像，并使用它来创建容器磁盘镜像。您可以将容器磁盘镜像存储在集群中可访问的 registry 中，并在 DPDK 检查配置映射的 **spec.param.vmContainerDiskImage** 属性中指定镜像位置。

要构建容器镜像，您必须创建一个镜像构建器虚拟机 (VM)。*镜像构建器虚拟机* 是一个 RHEL 8 虚拟机，可用于构建自定义 RHEL 镜像。

先决条件

- 镜像构建器虚拟机必须运行 RHEL 8.7，且必须至少在 **/var** 目录中有 2 个 CPU 内核、4 GiB RAM 和 20 GB 的可用空间。
- 您已在虚拟机上安装了镜像构建器工具及其 CLI (**composer-cli**)。
- 已安装 **virt-customize** 工具：

```
# dnf install libguestfs-tools
```

- 已安装 Podman CLI 工具 (**podman**)。

流程

1. 验证您可以构建 RHEL 8.7 镜像：

```
# composer-cli distros list
```

注意

要以非 root 身份运行 **composer-cli** 命令，请将您的用户添加到 **weldr** 或 **root** 组中：

```
# usermod -a -G weldr user
```

```
$ newgrp weldr
```

2. 输入以下命令以 TOML 格式创建镜像蓝图文件，其中包含要安装的软件包、内核自定义和服务，以及在引导时要禁用的服务：

```
$ cat << EOF > dpdk-vm.toml
name = "dpdk_image"
description = "Image to use with the DPDK checkup"
version = "0.0.1"
distro = "rhel-87"

[[customizations.user]]
name = "root"
password = "redhat"

[[packages]]
name = "dpdk"
```

```
[[packages]]
name = "dpdk-tools"

[[packages]]
name = "driverctl"

[[packages]]
name = "tuned-profiles-cpu-partitioning"

[customizations.kernel]
append = "default_hugepagesz=1GB hugepagesz=1G hugepages=1"

[customizations.services]
disabled = ["NetworkManager-wait-online", "sshd"]
EOF
```

3. 运行以下命令，将蓝图文件推送到镜像构建器工具中：

```
# composer-cli blueprints push dpdk-vm.toml
```

4. 通过指定蓝图名称和输出文件格式来生成系统镜像。在开始 compose 过程时，会显示镜像的通用唯一标识符(UUID)。

```
# composer-cli compose start dpdk_image qcow2
```

5. 等待 compose 进程完成。compose 状态必须先显示 **FINISHED**，然后才能继续下一步。

```
# composer-cli compose status
```

6. 输入以下命令通过指定其 UUID 来下载 **qcow2** 镜像文件：

```
# composer-cli compose image <UUID>
```

7. 运行以下命令来创建自定义脚本：

```
$ cat <<EOF >customize-vm
#!/bin/bash

# Setup hugepages mount
mkdir -p /mnt/huge
echo "hugetlbfs /mnt/huge hugetlbfs defaults,pagesize=1GB 0 0" >> /etc/fstab

# Create vfio-noiommu.conf
echo "options vfio enable_unsafe_noiommu_mode=1" > /etc/modprobe.d/vfio-noiommu.conf

# Enable guest-exec,guest-exec-status on the qemu-guest-agent configuration
sed -i '/^BLACKLIST_RPC=/ { s/guest-exec-status//; s/guest-exec//g }' /etc/sysconfig/qemu-
ga
sed -i '/^BLACKLIST_RPC=/ { s/,\+/,/g; s/^,\|,$//g }' /etc/sysconfig/qemu-ga
EOF
```

8. 使用 **virt-customize** 工具自定义镜像构建器工具生成的镜像：

```
$ virt-customize -a <UUID>-disk.qcow2 --run=customize-vm --selinux-relabel
```

9. 要创建包含构建容器镜像的所有命令的 Dockerfile，请输入以下命令：

```
$ cat << EOF > Dockerfile
FROM scratch
COPY --chown=107:107 <UUID>-disk.qcow2 /disk/
EOF
```

其中：

<UUID>-disk.qcow2

以 **qcow2** 格式指定自定义镜像的名称。

10. 运行以下命令构建并标记容器：

```
$ podman build . -t dpdk-rhel:latest
```

11. 运行以下命令，将容器磁盘镜像推送到集群可访问的 registry：

```
$ podman push dpdk-rhel:latest
```

12. 在 DPDK 检查配置映射中的 **spec.param.vmUnderTestContainerDiskImage** 属性中提供容器磁盘镜像的链接。

12.2.4. 其他资源

- 将虚拟机附加到多个网络
- 在 DPDK 模式中使用 Intel NIC 的虚拟功能
- 使用 SR-IOV 和 Node Tuning Operator 实现 DPDK 行率
- 安装镜像构建器
- 如何使用 Red Hat Subscription Manager 在红帽客户门户网站中注册并订阅 RHEL 系统

12.3. PROMETHEUS 对虚拟资源的查询

OpenShift Virtualization 提供用于监控集群基础架构资源消耗的指标，包括 vCPU、网络、存储和客户机内存交换。您还可以使用指标查询实时迁移状态。

12.3.1. 先决条件

- 要使用 vCPU 指标，必须将 **schedstats=enable** 内核参数应用到 **MachineConfig** 对象。此内核参数启用用于调试和性能调优的调度程序统计，并为调度程序添加较小的额外负载。如需更多信息，请参阅向节点添加内核参数。
- 要进行客户机内存交换查询以返回数据，必须在虚拟客户机上启用内存交换。

12.3.2. 查询指标

OpenShift Container Platform 监控仪表板可供您运行 Prometheus Query Language (PromQL) 查询来查看图表中呈现的指标。此功能提供有关集群以及要监控的任何用户定义工作负载的状态信息。

作为集群管理员，您可以查询所有 OpenShift Container Platform 核心项目和用户定义的项目的指标。

作为开发者，您必须在查询指标时指定项目名称。您必须具有所需权限才能查看所选项目的指标。

12.3.2.1. 以集群管理员身份查询所有项目的指标

作为集群管理员，或具有所有项目的查看权限的用户，您可以在 Metrics UI 中访问所有 OpenShift Container Platform 默认项目和用户定义的项目的指标。

先决条件

- 您可以使用具有 **cluster-admin** 集群角色的用户访问集群，或者具有所有项目的查看权限。
- 已安装 OpenShift CLI(**oc**)。

流程

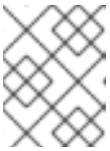
1. 从 OpenShift Container Platform Web 控制台中的 **Administrator** 视角，选择 **Observe → Metrics**。
2. 要添加一个或多个查询，请执行以下操作之一：

选项	描述
创建自定义查询。	将 Prometheus Query Language (PromQL) 查询添加到 **Expression** 字段中。 当您输入 PromQL 表达式时，自动完成建议会出现在下拉列表中。这些建议包括功能、指标、标签和时间令牌。您可以使用键盘箭头选择其中一项建议的项目，然后按 Enter 将项目添加到您的表达式中。您还可以将鼠标指针移到建议的项目上，以查看该项目的简短描述。
添加多个查询。	选择 **Add query**。
复制现有的查询。	选择查询旁边的 Options 菜单 ，然后选择 **Duplicate 查询**。
禁用查询正在运行。	选择查询旁边的 Options 菜单 并选择 **Disable query**。

3. 要运行您创建的查询，请选择 **Run queries**。图表中会直观呈现查询的指标。如果查询无效，则 UI 会显示错误消息。

注意

如果查询对大量数据进行运算，这可能会在绘制时序图时造成浏览器超时或过载。要避免这种情况，请选择 **Hide graph** 并且仅使用指标表来校准查询。然后，在找到可行的查询后，启用图表来绘制图形。



注意

默认情况下，查询表会显示一个展开的视图，列出每个指标及其当前值。您可以选择 ˅ 来最小化查询的展开视图。

4. 可选：页面 URL 现在包含您运行的查询。要在以后再次使用这一组查询，请保存这个 URL。

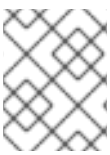
5. 探索视觉化指标。最初，图表中显示所有启用的查询中的所有指标。您可以通过执行以下操作来选择显示哪些指标：

选项	描述
隐藏查询中的所有指标。	点查询的 Options 菜单  并点 **Hide all series**。
隐藏特定指标。	前往查询表，再点指标名称旁边的带颜色的方格。
放大图表并更改时间范围。	任一： • 点击图表并在水平方向上拖动，以可视化方式选择时间范围。 • 使用左上角的菜单来选择时间范围。
重置时间范围。	选择 **Reset zoom**。
在特定时间点显示所有查询的输出。	将鼠标光标悬停在图表上。弹出框中会显示查询输出。
隐藏图表。	选择 **Hide graph**。

12.3.2.2. 以开发者身份查询用户定义的项目的指标

您可以以开发者或具有项目查看权限的用户身份访问用户定义项目的指标。

在 **Developer** 视角中， Metrics UI 包括所选项目的一些预定义 CPU、内存、带宽和网络数据包查询。您还可以对项目的 CPU、内存、带宽、网络数据包和应用程序指标运行自定义 Prometheus Query Language (PromQL) 查询。



注意

开发者只能使用 **Developer** 视角，而不能使用 **Administrator** 视角。作为开发者，您一次只能查询一个项目的指标。

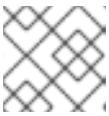
先决条件

- 对于您要查看指标的项目，您可以作为开发者或具有查看权限的用户访问集群。
- 您已为用户定义的项目启用了监控。

- 您已在用户定义的项目中部署了服务。
- 您已为该服务创建了 **ServiceMonitor** 自定义资源定义（CRD），以定义如何监控该服务。

流程

1. 在 OpenShift Container Platform web 控制台中的 **Developer** 视角中，选择 **Observe → Metrics**。
2. 在 **Project:** 列表中选择您要查看指标的项目。
3. 从 **Select query** 列表中选择查询，或者通过选择 **Show PromQL** 根据所选查询创建自定义 PromQL 查询。图表中会直观呈现查询的指标。



注意

在 Developer 视角中，您一次只能运行一个查询。

4. 通过执行以下操作来探索视觉化的指标：

选项	描述
放大图表并更改时间范围。	任一： • 点击图表并在水平方向上拖动，以可视化方式选择时间范围。 • 使用左上角的菜单来选择时间范围。
重置时间范围。	选择 **Reset zoom**。
在特定时间点显示所有查询的输出。	将鼠标光标悬停在图表上。查询输出会出现在弹出窗口中。

12.3.3. 虚拟化指标

以下指标描述包括示例 Prometheus Query Language（PromQL）查询。这些指标不是 API，可能在不同版本之间有所变化。有关虚拟化指标的完整列表，请参阅 KubeVirt 组件指标。

注意

以下示例使用 **topk** 查询来指定时间段。如果在那个时间段内删除虚拟机，它们仍然会显示在查询输出中。

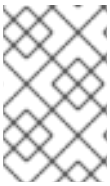
12.3.3.1. vCPU 指标

以下查询可以识别等待输入/输出 (I/O) 的虚拟机：

kubevirt_vmi_vcpu_wait_seconds_total

返回虚拟机的 vCPU 等待时间（以秒为单位）。类型：计数器。

高于"0" 的值表示 vCPU 要运行，但主机调度程序还无法运行它。无法运行代表 I/O 存在问题。



注意

要查询 vCPU 指标，必须首先将 **schedstats=enable** 内核参数应用到 **MachineConfig** 对象。此内核参数启用用于调试和性能调优的调度程序统计，并为调度程序添加较小的额外负载。

vCPU 等待时间查询示例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_vcpu_wait_seconds_total[6m]))) > 0 1
```

1 此查询会返回在六分钟内每次都等待 I/O 的 3 台虚拟机。

12.3.3.2. 网络指标

以下查询可以识别正在饱和网络的虚拟机：

kubevirt_vmi_network_receive_bytes_total

返回虚拟机网络中接收的流量总数（以字节为单位）。类型：计数器。

kubevirt_vmi_network_transmit_bytes_total

返回虚拟机网络上传输的流量总数（以字节为单位）。类型：计数器。

网络流量查询示例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_network_receive_bytes_total[6m])) + sum by (name, namespace) (rate(kubevirt_vmi_network_transmit_bytes_total[6m]))) > 0 1
```

1 此查询会返回每给定时间在六分钟内传输最多流量的 3 台虚拟机。

12.3.3.3. 存储指标

12.3.3.3.1. 与存储相关的流量

以下查询可以识别正在写入大量数据的虚拟机：

kubevirt_vmi_storage_read_traffic_bytes_total

返回虚拟机与存储相关的流量的总量（以字节为单位）。类型：计数器。

kubevirt_vmi_storage_write_traffic_bytes_total

返回虚拟机与存储相关的流量的存储写入总量（以字节为单位）。类型：计数器。

与存储相关的流量查询示例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_storage_read_traffic_bytes_total[6m])) + sum by (name, namespace) (rate(kubevirt_vmi_storage_write_traffic_bytes_total[6m]))) > 0 1
```

1 此查询会返回在六分钟内每给定时间执行最多存储流量的 3 台虚拟机。

12.3.3.3.2. 存储快照数据

kubevirt_vmsnapshot_disks_restored_from_source

返回从源虚拟机中恢复的虚拟机磁盘总数。类型：Gauge。

kubevirt_vmsnapshot_disks_restored_from_source_bytes

返回从源虚拟机恢复的字节空间量。类型：Gauge。

存储快照数据查询示例

```
kubevirt_vmsnapshot_disks_restored_from_source{vm_name="simple-vm",
vm_namespace="default"} 1
```

1 此查询返回从源虚拟机恢复的虚拟机磁盘总数。

```
kubevirt_vmsnapshot_disks_restored_from_source_bytes{vm_name="simple-vm",
vm_namespace="default"} 1
```

1 此查询返回源虚拟机恢复的空间大小，以字节为单位。

12.3.3.3.3. I/O 性能

以下查询可决定存储设备的 I/O 性能：

kubevirt_vmi_storage_iops_read_total

返回虚拟机每秒执行的写入 I/O 操作量。类型：计数器。

kubevirt_vmi_storage_iops_write_total

返回虚拟机每秒执行的读取 I/O 操作量。类型：计数器。

I/O 性能查询示例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_storage_iops_read_total[6m])) + sum by
(name, namespace) (rate(kubevirt_vmi_storage_iops_write_total[6m]))) > 0 1
```

1 此查询返回在六分钟内每给定时间每秒执行最多 I/O 操作数的 3 台虚拟机。

12.3.3.4. 客户机内存交换指标

以下查询可识别启用了交换最多的客户端执行内存交换最多：

kubevirt_vmi_memory_swap_in_traffic_bytes

返回虚拟客户机交换的内存总量（以字节为单位）。类型：Gauge。

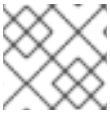
kubevirt_vmi_memory_swap_out_traffic_bytes

返回虚拟 guest 正在交换的内存总量（以字节为单位）。类型：Gauge。

内存交换查询示例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_memory_swap_in_traffic_bytes[6m])) + sum
by (name, namespace) (rate(kubevirt_vmi_memory_swap_out_traffic_bytes[6m]))) > 0 1
```

1 此查询返回前 3 台虚拟机，其中客户机在六分钟内执行每个给定时间段内每个给定时间最多的内存交换。



注意

内存交换表示虚拟机面临内存压力。增加虚拟机的内存分配可以缓解此问题。

12.3.3.5. 实时迁移指标

可以查询以下指标来显示实时迁移状态：

kubevirt_vmi_migration_data_processed_bytes

迁移到新虚拟机(VM)的客户机操作系统数据量。类型：Gauge。

kubevirt_vmi_migration_data_remaining_bytes

要迁移的客户机操作系统数据量。类型：Gauge。

kubevirt_vmi_migration_memory_transfer_rate_bytes

在客户端操作系统中达到脏的速率。脏内存是已更改但还没有写入磁盘的数据。类型：Gauge。

kubevirt_vmi_migrations_in_pending_phase

待处理的迁移数量。类型：Gauge。

kubevirt_vmi_migrations_in_scheduling_phase

调度迁移的数量。类型：Gauge。

kubevirt_vmi_migrations_in_running_phase

正在运行的迁移数量。类型：Gauge。

kubevirt_vmi_migration_succeeded

成功完成迁移的数量。类型：Gauge。

kubevirt_vmi_migration_failed

失败的迁移数量。类型：Gauge。

12.3.4. 其他资源

- 监控概述
- 查询 Prometheus
- Prometheus 查询示例

12.4. 为虚拟机公开自定义指标

OpenShift Container Platform 包括一个预配置、预安装和自我更新的监控堆栈，可为核心平台组件提供监控。此监控堆栈基于 Prometheus 监控系统。Prometheus 是一个时间序列数据库和用于指标的规则评估引擎。

除了使用 OpenShift Container Platform 监控堆栈外，您还可以使用 CLI 启用对用户定义的项目的监控，并查询通过 **node-exporter** 服务为虚拟机公开的自定义指标。

12.4.1. 配置节点导出器服务

node-exporter 代理部署在您要从中收集指标的集群中的每个虚拟机上。将 node-exporter 代理配置为服务，以公开与虚拟机关联的内部指标和进程。

先决条件

- 安装 OpenShift Container Platform CLI **oc**。
- 以具有 **cluster-admin** 权限的用户身份登录集群。
- 在 **openshift-monitoring** 项目中创建 **cluster-monitoring-config ConfigMap** 对象。
- 通过将 **enableUserWorkload** 设置为 **true**，配置 **openshift-user-workload-monitoring** 项目中的 **user-workload-monitoring-config ConfigMap** 对象。

流程

1. 创建 **Service** YAML 文件。在以下示例中，该文件名为 **node-exporter-service.yaml**。

```
kind: Service
apiVersion: v1
metadata:
  name: node-exporter-service 1
  namespace: dynamation 2
  labels:
    servicetype: metrics 3
spec:
  ports:
    - name: exmet 4
      protocol: TCP
      port: 9100 5
      targetPort: 9100 6
  type: ClusterIP
  selector:
    monitor: metrics 7
```

1. 从虚拟机公开指标的 node-exporter 服务。
2. 创建服务的命名空间。
3. 服务的标签。**ServiceMonitor** 使用此标签来匹配此服务。
4. 提供给通过端口 9100 为 **ClusterIP** 服务公开指标的端口的名称。
5. **node-exporter-service** 用来侦听请求的目标端口。
6. 配置有 **monitor** 标签的虚拟机的 TCP 端口号。
7. 用于与虚拟机的 pod 匹配的标签。在本例中，任何具有标签 **monitor**，值为 **metrics** 的虚拟机的 pod 将被匹配。

2. 创建 node-exporter 服务：

```
$ oc create -f node-exporter-service.yaml
```

12.4.2. 配置使用节点 exporter 服务的虚拟机

将 **node-exporter** 文件下载到虚拟机。然后，创建一个在虚拟机引导时运行 node-exporter 服务的 **systemd** 服务。

先决条件

- 该组件的 Pod 在 **openshift-user-workload-monitoring** 项目中运行。
- 向需要监控此用户定义的项目的用户授予 **monitoring-edit** 角色。

流程

1. 登录虚拟机。
2. 使用应用到 **node-exporter** 文件的目录的路径，将 **node-exporter** 文件下载到虚拟机。

```
$ wget
https://github.com/prometheus/node_exporter/releases/download/v1.3.1/node_exporter-
1.3.1.linux-amd64.tar.gz
```

3. 提取可执行文件并将其放置在 **/usr/bin** 目录中。

```
$ sudo tar xvf node_exporter-1.3.1.linux-amd64.tar.gz \
    --directory /usr/bin --strip 1 "*/node_exporter"
```

4. 在此目录路径中创建 **node_exporter.service** 文件： **/etc/systemd/system**。此 **systemd** 服务文件在虚拟机重启时运行 node-exporter 服务。

```
[Unit]
Description=Prometheus Metrics Exporter
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=always
RestartSec=1
User=root
ExecStart=/usr/bin/node_exporter

[Install]
WantedBy=multi-user.target
```

5. 启用并启动 **systemd** 服务。

```
$ sudo systemctl enable node_exporter.service
$ sudo systemctl start node_exporter.service
```

验证

- 验证 node-exporter 代理是否已报告虚拟机的指标。

```
$ curl http://localhost:9100/metrics
```

-

输出示例

```
go_gc_duration_seconds{quantile="0"} 1.5244e-05
go_gc_duration_seconds{quantile="0.25"} 3.0449e-05
go_gc_duration_seconds{quantile="0.5"} 3.7913e-05
```

12.4.3. 为虚拟机创建自定义监控标签

要启用来自单个服务的多个虚拟机的查询，请在虚拟机的 YAML 文件中添加自定义标签。

先决条件

- 安装 OpenShift Container Platform CLI **oc**。
- 以具有 **cluster-admin** 特权的用户身份登录。
- 访问 web 控制台以停止和重启虚拟机。

流程

1. 编辑虚拟机配置文件的**模板** spec。在本例中，标签 **monitor** 具有值 **metrics**。

```
spec:
  template:
    metadata:
      labels:
        monitor: metrics
```

2. 停止并重启虚拟机，以创建具有与 **monitor** 标签给定标签名称的新 pod。

12.4.3.1. 查询 node-exporter 服务以获取指标

指标通过 **/metrics** 规范名称下的 HTTP 服务端点公开。当您查询指标时，Prometheus 会直接从虚拟机公开的指标端点中提取指标，并展示这些指标来查看。

先决条件

- 您可以使用具有 **cluster-admin** 特权或 **monitoring-edit** 角色的用户访问集群。
- 您已通过配置 node-exporter 服务为用户定义的项目启用了监控。

流程

1. 通过为服务指定命名空间来获取 HTTP 服务端点：

```
$ oc get service -n <namespace> <node-exporter-service>
```

2. 要列出 node-exporter 服务的所有可用指标，请查询 **metrics** 资源。

```
$ curl http://<172.30.226.162:9100>/metrics | grep -vE "^#|^$"
```

输出示例

```
node_arp_entries{device="eth0"} 1
node_boot_time_seconds 1.643153218e+09
node_context_switches_total 4.4938158e+07
node_cooling_device_cur_state{name="0",type="Processor"} 0
node_cooling_device_max_state{name="0",type="Processor"} 0
node_cpu_guest_seconds_total{cpu="0",mode="nice"} 0
node_cpu_guest_seconds_total{cpu="0",mode="user"} 0
node_cpu_seconds_total{cpu="0",mode="idle"} 1.10586485e+06
node_cpu_seconds_total{cpu="0",mode="iowait"} 37.61
node_cpu_seconds_total{cpu="0",mode="irq"} 233.91
node_cpu_seconds_total{cpu="0",mode="nice"} 551.47
node_cpu_seconds_total{cpu="0",mode="softirq"} 87.3
node_cpu_seconds_total{cpu="0",mode="steal"} 86.12
node_cpu_seconds_total{cpu="0",mode="system"} 464.15
node_cpu_seconds_total{cpu="0",mode="user"} 1075.2
node_disk_discard_time_seconds_total{device="vda"} 0
node_disk_discard_time_seconds_total{device="vdb"} 0
node_disk_discarded_sectors_total{device="vda"} 0
node_disk_discarded_sectors_total{device="vdb"} 0
node_disk_discards_completed_total{device="vda"} 0
node_disk_discards_completed_total{device="vdb"} 0
node_disk_discards_merged_total{device="vda"} 0
node_disk_discards_merged_total{device="vdb"} 0
node_disk_info{device="vda",major="252",minor="0"} 1
node_disk_info{device="vdb",major="252",minor="16"} 1
node_disk_io_now{device="vda"} 0
node_disk_io_now{device="vdb"} 0
node_disk_io_time_seconds_total{device="vda"} 174
node_disk_io_time_seconds_total{device="vdb"} 0.054
node_disk_io_time_weighted_seconds_total{device="vda"} 259.79200000000003
node_disk_io_time_weighted_seconds_total{device="vdb"} 0.039
node_disk_read_bytes_total{device="vda"} 3.71867136e+08
node_disk_read_bytes_total{device="vdb"} 366592
node_disk_read_time_seconds_total{device="vda"} 19.128
node_disk_read_time_seconds_total{device="vdb"} 0.039
node_disk_reads_completed_total{device="vda"} 5619
node_disk_reads_completed_total{device="vdb"} 96
node_disk_reads_merged_total{device="vda"} 5
node_disk_reads_merged_total{device="vdb"} 0
node_disk_write_time_seconds_total{device="vda"} 240.66400000000002
node_disk_write_time_seconds_total{device="vdb"} 0
node_disk_writes_completed_total{device="vda"} 71584
node_disk_writes_completed_total{device="vdb"} 0
node_disk_writes_merged_total{device="vda"} 19761
node_disk_writes_merged_total{device="vdb"} 0
node_disk_written_bytes_total{device="vda"} 2.007924224e+09
node_disk_written_bytes_total{device="vdb"} 0
```

12.4.4. 为节点 exporter 服务创建 ServiceMonitor 资源

您可以使用 Prometheus 客户端库和从 **/metrics** 端点中提取指标来访问和查看 node-exporter 服务公开的指标。使用 **ServiceMonitor** 自定义资源定义(CRD)来监控节点 exporter 服务。

先决条件

- 您可以使用具有 **cluster-admin** 特权或 **monitoring-edit** 角色的用户访问集群。
- 您已通过配置 node-exporter 服务为用户定义的项目启用了监控。

流程

1. 为 **ServiceMonitor** 资源配置创建一个 YAML 文件。在本例中，服务监控器与带有标签 **metrics** 的任意服务匹配，每 30 秒对 **exmet** 端口进行查询。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    k8s-app: node-exporter-metrics-monitor
  name: node-exporter-metrics-monitor (1)
  namespace: dynamation (2)
spec:
  endpoints:
  - interval: 30s (3)
    port: exmet (4)
    scheme: http
  selector:
    matchLabels:
      servicetype: metrics
```

(1) **ServiceMonitor** 的名称。

(2) 创建 **ServiceMonitor** 的命名空间。

(3) 要查询端口的时间间隔。

(4) 每 30 秒查询的端口名称

2. 为 node-exporter 服务创建 **ServiceMonitor** 配置。

```
$ oc create -f node-exporter-metrics-monitor.yaml
```

12.4.4.1. 访问集群外的节点导出服务

您可以访问集群外的 node-exporter 服务，并查看公开的指标。

先决条件

- 您可以使用具有 **cluster-admin** 特权或 **monitoring-edit** 角色的用户访问集群。
- 您已通过配置 node-exporter 服务为用户定义的项目启用了监控。

流程

1. 公开 node-exporter 服务。

```
$ oc expose service -n <namespace> <node_exporter_service_name>
```

2. 获取路由的 FQDN（全限定域名）。

```
$ oc get route -o=custom-columns=NAME:.metadata.name,DNS:.spec.host
```

输出示例

```
NAME                  DNS
node-exporter-service   node-exporter-service-dynamation.apps.cluster.example.org
```

3. 使用 **curl** 命令显示 node-exporter 服务的指标。

```
$ curl -s http://node-exporter-service-dynamation.apps.cluster.example.org/metrics
```

输出示例

```
go_gc_duration_seconds{quantile="0"} 1.5382e-05
go_gc_duration_seconds{quantile="0.25"} 3.1163e-05
go_gc_duration_seconds{quantile="0.5"} 3.8546e-05
go_gc_duration_seconds{quantile="0.75"} 4.9139e-05
go_gc_duration_seconds{quantile="1"} 0.000189423
```

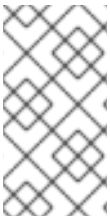
12.4.5. 其他资源

- 配置监控堆栈
- 为用户定义的项目启用监控
- 管理指标
- 查看监控仪表板
- 使用健康检查来监控应用程序的健康状态
- 创建和使用配置映射
- 控制虚拟机状态

12.5. 为虚拟机公开下限指标

作为管理员，您可以通过首先启用 **DownwardMetrics** 功能门，然后配置 **DownwardMetrics** 设备，来向客户虚拟机公开一组有限的主机和虚拟机(VM)指标。

用户可以使用命令行或 **vm-dump-metrics** 工具来查看指标结果。



注意

在 Red Hat Enterprise Linux (RHEL) 9 中，使用命令行查看 Downward 指标。请参阅使用命令行查看 Downward指标。

Red Hat Enterprise Linux (RHEL) 9 平台上不支持 vm-dump-metrics 工具。

12.5.1. 启用或禁用 DownwardMetrics 功能门

您可以通过执行以下操作之一启用或禁用 **DownwardMetrics** 功能门：

- 在默认编辑器中编辑 HyperConverged 自定义资源 (CR)
- 使用命令行

12.5.1.1. 在 YAML 文件中启用或禁用 Downward 指标功能门

要为主机虚拟机公开下限指标，您可以通过编辑 YAML 文件来启用 **DownwardMetrics** 功能门。

先决条件

- 您必须具有管理员特权才能启用功能门。

流程

1. 运行以下命令，在默认编辑器中打开 HyperConverged 自定义资源 (CR)：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 选择启用或禁用 DownwardMetrics 功能门，如下所示：
   - 要启用 **DownwardMetrics** 功能门，请将 **spec.featureGates.downwardMetrics** 设置为 **true**。例如：

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
    featureGates:
      downwardMetrics: true
# ...
```

   - 要禁用 **DownwardMetrics** 功能门，请将 **spec.featureGates.downwardMetrics** 设置为 **false**。例如：

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
    featureGates:
      downwardMetrics: false
# ...
```

12.5.1.2. 从命令行启用或禁用 Downward 指标功能门

要为主机虚拟机公开下限指标，您可以使用命令行启用 **DownwardMetrics** 功能门。

先决条件

- 您必须具有管理员特权才能启用功能门。

流程

- 选择启用或禁用 **DownwardMetrics** 功能门，如下所示：
  - 运行以下命令启用 **DownwardMetrics** 功能门：

```
$ oc patch hco kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op": "replace", "path": \
  "/spec/featureGates/downwardMetrics" \
  "value": true}]'
```

  - 运行以下命令，禁用 **DownwardMetrics** 功能门：

```
$ oc patch hco kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op": "replace", "path": \
  "/spec/featureGates/downwardMetrics" \
  "value": false}]'
```

12.5.2. 配置 Downward 指标设备

您可以通过创建一个包含 **downwardMetrics** 设备的配置文件，为主机虚拟机启用 downward 指标。添加此设备会建立指标通过 **virtio-serial** 端口公开。

先决条件

- 您必须首先启用 **DownwardMetrics** 功能门。

流程

- 编辑或创建包含 **DownwardMetrics** 设备的 YAML 文件，如下例所示：

DownwardMetrics 配置文件示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: fedora
  namespace: default
spec:
  dataVolumeTemplates:
    - metadata:
        name: fedora-volume
      spec:
        sourceRef:
          kind: DataSource
          name: fedora
          namespace: openshift-virtualization-os-images
        storage:
          resources: {}
          storageClassName: hostpath-csi-basic
  instancetype:
    name: u1.medium
```

```
preference:
  name: fedora
running: true
template:
  metadata:
    labels:
      app.kubernetes.io/name: headless
  spec:
    domain:
      devices:
        downwardMetrics: {} (1)
    subdomain: headless
    volumes:
      - dataVolume:
          name: fedora-volume
        name: rootdisk
      - cloudInitNoCloud:
          userData: |
            #cloud-config
            chpasswd:
              expire: false
            password: '<password>' (2)
            user: fedora
        name: cloudinitdisk
```

1. **downwardMetrics** 设备。
2. **fedora** 用户的密码。

12.5.3. 查看 downward 指标

您可以使用以下选项之一查看 downward 指标：

- 命令行界面 (CLI)
- **vm-dump-metrics** 工具

注意

在 Red Hat Enterprise Linux (RHEL) 9 中，使用命令行查看 Downward 指标。Red Hat Enterprise Linux (RHEL) 9 平台上不支持 vm-dump-metrics 工具。

12.5.3.1. 使用命令行查看下限指标

您可以通过从客户虚拟机(VM) 内输入命令来查看 downward 指标。

流程

- 运行以下命令：

```
$ sudo sh -c 'printf "GET /metrics/XML\n\n" > /dev/virtio-ports/org.github.vhostmd.1'
```

```
$ sudo cat /dev/virtio-ports/org.github.vhostmd.1
```

12.5.3.2. 使用 vm-dump-metrics 工具查看 downward 指标

要查看 downward 指标，请安装 **vm-dump-metrics** 工具，然后使用该工具公开指标结果。

注意

在 Red Hat Enterprise Linux (RHEL) 9 中，使用命令行查看 Downward 指标。Red Hat Enterprise Linux (RHEL) 9 平台上不支持 vm-dump-metrics 工具。

流程

1. 运行以下命令安装 **vm-dump-metrics** 工具：

```
$ sudo dnf install -y vm-dump-metrics
```

2. 运行以下命令来检索指标结果：

```
$ sudo vm-dump-metrics
```

输出示例

```
<metrics>
  <metric type="string" context="host">
    <name>HostName</name>
    <value>node01</value>
[...]
  <metric type="int64" context="host" unit="s">
    <name>Time</name>
    <value>1619008605</value>
  </metric>
  <metric type="string" context="host">
    <name>VirtualizationVendor</name>
    <value>kubevirt.io</value>
  </metric>
</metrics>
```

12.6. 虚拟机健康检查

您可以通过在 **VirtualMachine** 资源中定义就绪度和存活度探测来配置虚拟机 (VM) 健康检查。

12.6.1. 关于就绪度和存活度探测

使用就绪度和存活度探测来检测和处理不健康的虚拟机 (VM)。您可以在虚拟机规格中包含一个或多个探测，以确保流量无法访问未就绪的虚拟机，并在虚拟机变得无响应时创建新虚拟机。

*就绪度探测*决定 VMI 是否准备好接受服务请求。如果探测失败，则 VMI 会从可用端点列表中移除，直到 VMI 就绪为止。

*存活度探测*决定 VMI 是否响应。如果探测失败，则会删除虚拟机并创建一个新的虚拟机来恢复响应性。

您可以通过设置 **VirtualMachine** 对象的 **spec.readinessProbe** 和 **spec.livenessProbe** 字段来配置就绪度和存活度探测。这些字段支持以下测试：

HTTP GET

该探测使用 Web hook 确定 VMI 的健康状况。如果 HTTP 响应代码介于 200 和 399 之间，则测试成功。您可以将 HTTP GET 测试用于在完全初始化时返回 HTTP 状态代码的应用程序。

TCP 套接字

该探测尝试为 VMI 打开一个套接字。只有在探测可以建立连接时，VMI 才被视为健康。对于在初始化完成前不会开始监听的应用程序，可以使用 TCP 套接字测试。

客户机代理 ping

该探测使用 **guest-ping** 命令来确定 QEMU 客户机代理是否在虚拟机上运行。

12.6.1.1. 定义 HTTP 就绪度探测

通过设置虚拟机配置的 **spec.readinessProbe.httpGet** 字段来定义 HTTP 就绪度探测。

流程

1. 在虚拟机配置文件中包括就绪度探测的详细信息。

使用 HTTP GET 测试就绪度探测示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      readinessProbe:
        httpGet: 1
          port: 1500 2
          path: /healthz 3
          httpHeaders:
          - name: Custom-Header
            value: Awesome
        initialDelaySeconds: 120 4
        periodSeconds: 20 5
        timeoutSeconds: 10 6
        failureThreshold: 3 7
        successThreshold: 3 8
# ...
```

1. 执行的 HTTP GET 请求以连接 VM。
2. 探测查询的虚拟机端口。在上例中，探测查询端口 1500。
3. 在 HTTP 服务器上访问的路径。在上例中，如果服务器的 /healthz 路径的处理程序返回成功代码，则 VMI 被视为健康。如果处理程序返回失败代码，则虚拟机将从可用端点列表中移除。
4. 虚拟机启动就绪度探测前的时间（以秒为单位）。

5 执行探测之间的延迟（以秒为单位）。默认延迟为 10 秒。这个值必须大于 **timeoutSeconds**。

6 在不活跃的时间（以秒为单位）超过这个值时探测会超时，且假定 VM 失败。默认值为 1。这个值必须小于 **periodSeconds**。

7 探测允许失败的次数。默认值为 3。在进行了指定数量的尝试后，pod 被标记为 **Unready**。

8 在失败后，在探测报告成功的次数达到这个值时才能被视为成功。默认值为 1。

2. 运行以下命令来创建虚拟机：

```
$ oc create -f <file_name>.yaml
```

12.6.1.2. 定义 TCP 就绪度探测

通过设置虚拟机 (VM) 配置的 **spec.readinessProbe.tcpSocket** 字段来定义 TCP 就绪度探测。

流程

1. 在虚拟机配置文件中包括 TCP 就绪度探测的详细信息。

使用 TCP 套接字测试的就绪度探测示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      readinessProbe:
        initialDelaySeconds: 120 1
        periodSeconds: 20 2
        tcpSocket: 3
          port: 1500 4
        timeoutSeconds: 10 5
# ...
```

1 虚拟机启动就绪度探测前的时间（以秒为单位）。

2 执行探测之间的延迟（以秒为单位）。默认延迟为 10 秒。这个值必须大于 **timeoutSeconds**。

3 要执行的 TCP 操作。

4 探测查询的虚拟机端口。

5 在不活跃的时间（以秒为单位）超过这个值时探测会超时，且假定 VM 失败。默认值为 1。这个值必须小于 **periodSeconds**。

2. 运行以下命令来创建虚拟机：

```
$ oc create -f <file_name>.yaml
```

12.6.1.3. 定义 HTTP 存活度探测

通过设置虚拟机 (VM) 配置的 **spec.livenessProbe.httpGet** 字段来定义 HTTP 存活度探测。您可以按照与就绪度探测相同的方式为存活度探测定义 HTTP 和 TCP 测试。此流程使用 HTTP GET 测试配置示例存活度探测。

流程

1. 在虚拟机配置文件中包括 HTTP 存活度探测的详细信息。

使用 HTTP GET 测试的存活度探测示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      livenessProbe:
        initialDelaySeconds: 120 1
        periodSeconds: 20 2
        httpGet: 3
          port: 1500 4
          path: /healthz 5
          httpHeaders:
          - name: Custom-Header
            value: Awesome
        timeoutSeconds: 10 6
# ...
```

1. 虚拟机启动存活度探测前的时间（以秒为单位）。
2. 执行探测之间的延迟（以秒为单位）。默认延迟为 10 秒。这个值必须大于 **timeoutSeconds**。
3. 执行的 HTTP GET 请求以连接 VM。
4. 探测查询的虚拟机端口。在上例中，探测查询端口 1500。VM 通过 cloud-init 在端口 1500 上安装并运行最小 HTTP 服务器。
5. 在 HTTP 服务器上访问的路径。在上例中，如果服务器的 **/healthz** 路径的处理程序返回成功代码，则 VMI 被视为健康。如果处理程序返回失败代码，则会删除虚拟机并创建新虚拟机。
6. 在不活跃的时间（以秒为单位）超过这个值时探测会超时，且假定 VM 失败。默认值为 1。这个值必须小于 **periodSeconds**。

2. 运行以下命令来创建虚拟机：

```
$ oc create -f <file_name>.yaml
```

12.6.2. 定义 watchdog

您可以通过执行以下步骤来定义 watchdog 来监控客户端操作系统的健康状态：

1. 为虚拟机配置 watchdog 设备。
2. 在客户机上安装 watchdog 代理。

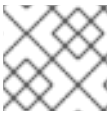
watchdog 设备监控代理，并在客户机操作系统不响应时执行以下操作之一：

- **poweroff** ：虚拟机立即关闭。如果 **spec.running** 设为 **true** 或 **spec.runStrategy** 没有设置为 **manual**，则虚拟机将重启。
- **reset** ：虚拟机重新启动，客户机操作系统无法响应。

注意

重启时间可能会导致存活度探测超时。如果集群级别的保护检测到失败的存活度探测，则虚拟机可能会被强制重新调度，从而增加重启时间。

- **shutdown** ：通过停止所有服务来正常关闭虚拟机。



注意

watchdog 不适用于 Windows 虚拟机。

12.6.2.1. 为虚拟机配置 watchdog 设备

您可以为虚拟机配置 watchdog 设备。

先决条件

- 虚拟机必须具有对 **i6300esb** watchdog 设备的内核支持。Red Hat Enterprise Linux (RHEL) 镜像支持 **i6300esb**。

流程

1. 创建包含以下内容的 **YAML** 文件：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm2-rhel84-watchdog
  name: <vm-name>
spec:
  running: false
  template:
    metadata:
      labels:
```

```
      kubevirt.io/vm: vm2-rhel84-watchdog
  spec:
    domain:
      devices:
        watchdog:
          name: <watchdog>
          i6300esb:
            action: "poweroff" (1)
# ...
```

(1) 指定 **poweroff**, **reset**, 或 **shutdown**.

上面的示例使用 poweroff 操作配置 RHEL8 虚拟机上的 **i6300esb** watchdog 设备，并将设备公开为 **/dev/watchdog**。

现在，watchdog 二进制文件可以使用这个设备。

2. 运行以下命令，将 YAML 文件应用到集群：

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

重要

此流程仅用于测试 watchdog 功能，且不得在生产环境中运行。

1. 运行以下命令来验证虚拟机是否已连接到 watchdog 设备：

```
$ lspci | grep watchdog -i
```

2. 运行以下命令之一以确认 watchdog 处于活跃状态：

- 触发内核 panic：

```
# echo c > /proc/sysrq-trigger
```

- 停止 watchdog 服务：

```
# pkill -9 watchdog
```

12.6.2.2. 在客户机上安装 watchdog 代理

您可以在客户机上安装 watchdog 代理并启动 **watchdog** 服务。

流程

1. 以 root 用户身份登录虚拟机。
2. 安装 **watchdog** 软件包及其依赖项：

```
# yum install watchdog
```

3. 在 **/etc/watchdog.conf** 文件中取消注释以下行并保存更改：

```
#watchdog-device = /dev/watchdog
```

4. 在引导时启用 **watchdog** 服务：

```
# systemctl enable --now watchdog.service
```

12.6.3. 定义客户机代理 ping 探测

通过设置虚拟机 (VM) 配置的 **spec.readinessProbe.guestAgentPing** 字段来定义客户机代理 ping 探测。

重要

客户机代理 ping 探测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。

先决条件

- 必须在虚拟机上安装并启用 QEMU 客户机代理。

流程

1. 在虚拟机配置文件中包括客户机代理 ping 探测的详情。例如：

客户机代理 ping 探测示例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      readinessProbe:
        guestAgentPing: {} 1
        initialDelaySeconds: 120 2
        periodSeconds: 20 3
        timeoutSeconds: 10 4
        failureThreshold: 3 5
        successThreshold: 3 6
# ...
```

1. 客户机代理 ping 探测以连接到虚拟机。
2. 可选：VM 启动客户机代理探测前的时间（以秒为单位）。

3. 可选：执行探测之间的延迟（以秒为单位）。默认延迟为 10 秒。这个值必须大于 **timeoutSeconds**。
4. 可选：在不活跃的时间（以秒为单位）超过这个值时探测会超时，且假定 VM 失败。默认值为 1。这个值必须小于 **periodSeconds**。
5. 可选：探测允许失败的次数。默认值为 3。在进行了指定数量的尝试后，pod 被标记为 **Unready**。
6. 可选：探测在失败后必须报告成功的次数，才被视为成功。默认值为 1。

2. 运行以下命令来创建虚拟机：

```
$ oc create -f <file_name>.yaml
```

12.6.4. 其他资源

- 使用健康检查来监控应用程序的健康状态

12.7. OPENSHIFT VIRTUALIZATION RUNBOOKS

OpenShift Virtualization Operator 的 runbooks 在 openshift/runbooks Git 存储库中维护，您可以在 GitHub 上查看它们。要诊断并解决触发 OpenShift Virtualization 警报的问题，请按照 runbooks 中的步骤操作。

OpenShift Virtualization 警报显示在 web 控制台的 **Virtualization → Overview** 选项卡中。

12.7.1. CDIDataImportCronOutdated

- 查看 **CDIDataImportCronOutdated** 警报的 runbook。

12.7.2. CDIDataVolumeUnusualRestartCount

- 查看 **CDIDataVolumeUnusualRestartCount** 警报的 runbook。

12.7.3. CDIDefaultStorageClassDegraded

- 查看 **CDIDefaultStorageClassDegraded** 警报的 runbook。

12.7.4. CDIMultipleDefaultVirtStorageClasses

- 查看 **CDIMultipleDefaultVirtStorageClasses** 警报的 runbook。

12.7.5. CDINoDefaultStorageClass

- 查看 **CDINoDefaultStorageClass** 警报的 runbook。

12.7.6. CDINotReady

- 查看 **CDINotReady** 警报的 runbook。

12.7.7. CDIOperatorDown

- 查看 **CDIOperatorDown** 警报的 runbook。

12.7.8. CDIStorageProfilesIncomplete

- 查看 **CDIStorageProfilesIncomplete** 警报的 runbook。

12.7.9. CnaoDown

- 查看 **CnaoDown** 警报的 runbook。

12.7.10. CnaoNMstateMigration

- 查看 **CnaoNMstateMigration** 警报的 runbook。

12.7.11. HCOInstallationIncomplete

- 查看 **HCOInstallationIncomplete** 警报的 runbook。

12.7.12. HPPNotReady

- 查看 **HPPNotReady** 警报的 runbook。

12.7.13. HPPOperatorDown

- 查看 **HPPOperatorDown** 警报的 runbook。

12.7.14. HPPSharingPoolPathWithOS

- 查看 **HPPSharingPoolPathWithOS** 警报的 runbook。

12.7.15. KubemacpoolDown

- 查看 **KubemacpoolDown** 警报的 runbook。

12.7.16. KubeMacPoolDuplicateMacsFound

- 查看 **KubeMacPoolDuplicateMacsFound** 警报的 runbook。

12.7.17. KubeVirtComponentExceedsRequestedCPU

- **KubeVirtComponentExceedsRequestedCPU** 警报已弃用。

12.7.18. KubeVirtComponentExceedsRequestedMemory

- **KubeVirtComponentExceedsRequestedMemory** 警报已弃用。

12.7.19. KubeVirtCRModified

- 查看 **KubeVirtCRModified** 警报的 runbook。

12.7.20. KubeVirtDeprecatedAPIRequested

- 查看 **KubeVirtDeprecatedAPIRequested** 警报的 runbook。

12.7.21. KubeVirtNoAvailableNodesToRunVMs

- 查看 **KubeVirtNoAvailableNodesToRunVM** 警报的 runbook。

12.7.22. KubevirtVmHighMemoryUsage

- 查看 **KubevirtVmHighMemoryUsage** 警报的 runbook。

12.7.23. KubeVirtVMIExcessiveMigrations

- 查看 **KubeVirtVMIExcessiveMigrations** 警报的 runbook。

12.7.24. LowKVMNodesCount

- 查看 **LowKVMNodesCount** 警报的 runbook。

12.7.25. LowReadyVirtControllersCount

- 查看 **LowReadyVirtControllersCount** 警报的 runbook。

12.7.26. LowReadyVirtOperatorsCount

- 查看 **LowReadyVirtOperatorsCount** 警报的 runbook。

12.7.27. LowVirtAPICount

- 查看 **LowVirtAPICount** 警报的 runbook。

12.7.28. LowVirtControllersCount

- 查看 **LowVirtControllersCount** 警报的 runbook。

12.7.29. LowVirtOperatorCount

- 查看 **LowVirtOperatorCount** 警报的 runbook。

12.7.30. NetworkAddonsConfigNotReady

- 查看 **NetworkAddonsConfigNotReady** 警报的 runbook。

12.7.31. NoLeadingVirtOperator

- 查看 **NoLeadingVirtOperator** 警报的 runbook。

12.7.32. NoReadyVirtController

- 查看 **NoReadyVirtController** 警报的 runbook。

12.7.33. NoReadyVirtOperator

- 查看 **NoReadyVirtOperator** 警报的 runbook。

12.7.34. OrphanedVirtualMachineInstances

- 查看 **OrphanedVirtualMachineInstances** 警报的 runbook。

12.7.35. OutdatedVirtualMachineInstanceWorkloads

- 查看 **OutdatedVirtualMachineInstanceWorkloads** 警报的 runbook。

12.7.36. SingleStackIPv6Unsupported

- 查看 **SingleStackIPv6Unsupported** 警报的 runbook。

12.7.37. SSPCommonTemplatesModificationReverted

- 查看 **SSPCommonTemplatesModificationReverted** 警报的 runbook。

12.7.38. SSPDown

- 查看 **SSPDown** 警报的 runbook。

12.7.39. SSPFailingToReconcile

- 查看 **SSPFailingToReconcile** 警报的 runbook。

12.7.40. SSPHighRateRejectedVms

- 查看 **SSPHighRateRejectedVms** 警报的 runbook。

12.7.41. SSPTemplateValidatorDown

- 查看 **SSPTemplateValidatorDown** 警报的 runbook。

12.7.42. UnsupportedHCOModification

- 查看 **UnsupportedHCOModification** 警报的 runbook。

12.7.43. VirtAPIDown

- 查看 **VirtAPIDown** 警报的 runbook。

12.7.44. VirtApiRESTErrorsBurst

- 查看 **VirtApiRESTErrorsBurst** 警报的 runbook。

12.7.45. VirtApiRESTErrorsHigh

- 查看 **VirtApiRESTErrorsHigh** 警报的 runbook。

12.7.46. VirtControllerDown

- 查看 **VirtControllerDown** 警报的 runbook。

12.7.47. VirtControllerRESTErrorsBurst

- 查看 **VirtControllerRESTErrorsBurst** 警报的 runbook。

12.7.48. VirtControllerRESTErrorsHigh

- 查看 **VirtControllerRESTErrorsHigh** 警报的 runbook。

12.7.49. VirtHandlerDaemonSetRolloutFailing

- 查看 **VirtHandlerDaemonSetRolloutFailing** 警报的 runbook。

12.7.50. VirtHandlerRESTErrorsBurst

- 查看 **VirtHandlerRESTErrorsBurst** 警报的 runbook。

12.7.51. VirtHandlerRESTErrorsHigh

- 查看 **VirtHandlerRESTErrorsHigh** 警报的 runbook。

12.7.52. VirtOperatorDown

- 查看 **VirtOperatorDown** 警报的 runbook。

12.7.53. VirtOperatorRESTErrorsBurst

- 查看 **VirtOperatorRESTErrorsBurst** 警报的 runbook。

12.7.54. VirtOperatorRESTErrorsHigh

- 查看 **VirtOperatorRESTErrorsHigh** 警报的 runbook。

12.7.55. VirtualMachineCRCErrors

- **VirtualMachineCRCErrors** 警报的 runbook 已被弃用，因为警报已重命名为 **VMStorageClassWarning**。
    - 查看 **VMStorageClassWarning** 警报的 runbook。

12.7.56. VMCannotBeEvicted

- 查看 **VMCannotBeEvicted** 警报的 runbook。

12.7.57. VMStorageClassWarning

- 查看 **VMStorageClassWarning** 警报的 runbook。

第 13 章 支持

13.1. 支持概述

您可以请求红帽支持的帮助，报告错误，收集有关环境的数据，并使用以下工具监控集群和虚拟机(VM)的健康状况。

13.1.1. 打开支持问题单

如果您遇到需要红帽支持立即帮助的问题，您可以提交支持问题单。

要报告程序错误，您可以直接创建一个 Jira 问题。

13.1.1.1. 提交支持问题单

要请求红帽支持的支持，请按照提交支持问题单的说明进行操作。

收集调试数据以与您的支持请求包含非常有用。

13.1.1.1.1. 为红帽支持收集数据

您可以执行以下步骤来收集调试信息：

收集有关环境的数据

配置 Prometheus 和 Alertmanager，并为 OpenShift Container Platform 和 OpenShift Virtualization 收集 **must-gather** 数据。

OpenShift Virtualization 的 must-gather 工具

配置和使用 **must-gather** 工具。

收集虚拟机的数据

从虚拟机收集 **must-gather** 数据和内存转储。

13.1.1.2. 创建 JIRA 问题

要报告错误，您可以通过在 Create Issue 页面上填写表单来直接创建一个 Jira 问题。

13.1.2. Web 控制台监控

您可以使用 OpenShift Container Platform Web 控制台监控集群和虚拟机的健康状态。Web 控制台显示集群的资源使用情况、警报、事件和趋势，以及 OpenShift Virtualization 组件和资源。

表 13.1. 用于监控和故障排除的 Web 控制台页面

页面	描述
概述页面	集群详情、状态、警报、清单和资源使用情况
Virtualization → Overview 标签页	OpenShift Virtualization 资源、使用量、警报和状态
Virtualization → Top consumers 标签页	CPU、内存和存储的主要使用者

页面	描述
Virtualization → Migrations 标签页	实时迁移的进度
VirtualMachines → VirtualMachine → VirtualMachine details → Metrics 标签页	VM 资源使用情况、存储、网络和迁移
VirtualMachines → VirtualMachine → VirtualMachine details → Events 标签页	VM 事件列表
VirtualMachines → VirtualMachine → VirtualMachine details → Diagnostics 标签页	虚拟机状态条件和卷快照状态

13.2. 为红帽支持收集数据

当您向红帽支持 提交支持问题单 时，使用以下工具为 OpenShift Container Platform 和 OpenShift Virtualization 提供调试信息会很有帮助：

must-gather 工具

must-gather 工具收集诊断信息，包括资源定义和服务日志。

Prometheus

Prometheus 是一个时间序列数据库和用于指标的规则评估引擎。Prometheus 将警报发送到 Alertmanager 进行处理。

Alertmanager

Alertmanager 服务处理从 Prometheus 接收的警报。Alertmanager 还负责将警报发送到外部通知系统。

如需有关 OpenShift Container Platform 监控堆栈的信息，请参阅关于 OpenShift Container Platform 监控。

13.2.1. 收集有关环境的数据

收集有关环境的数据可最小化分析和确定根本原因所需的时间。

先决条件

- 将 Prometheus 指标数据的保留时间设置为最少 7 天。
- 配置 Alertmanager 以捕获相关的警报并将其发送到专用邮箱，以便可以在集群外部查看和保留它们。
- 记录受影响的节点和虚拟机的确切数量。

流程

1. 为集群收集 must-gather 数据。
2. 如有必要，为 Red Hat OpenShift Data Foundation 收集 must-gather 数据 。
3. 为 OpenShift Virtualization 收集 must-gather 数据。

4. 收集集群的 Prometheus 指标。

13.2.2. 收集虚拟机的数据

收集有关出现故障的虚拟机 (VM) 的数据可最小化分析和确定根本原因所需的时间。

先决条件

- Linux 虚拟机： 安装最新的 QEMU 客户机代理。
- Windows 虚拟机：
  - 记录 Windows 补丁更新详情。
  - 安装最新的 VirtIO 驱动程序。
  - 安装最新的 QEMU 客户机代理。
  - 如果启用了远程桌面协议(RDP)，使用 桌面查看器 进行连接以确定连接软件是否存在问题。

流程

1. 使用 **/usr/bin/gather** 脚本为虚拟机收集 must-gather 数据。
2. 收集在重启*前*崩溃的虚拟机截图。
3. 在修复尝试*前*，从虚拟机收集内存转储。
4. 记录出现故障的虚拟机通常具有的因素。例如，虚拟机具有相同的主机或网络。

13.2.3. 为 OpenShift Virtualization 使用 must-gather 工具

您可以使用 OpenShift Virtualization 镜像运行 **must-gather** 命令收集有关 OpenShift Virtualization 资源的数据。

默认数据收集包含有关以下资源的信息：

- OpenShift Virtualization Operator 命名空间，包括子对象
- OpenShift Virtualization 自定义资源定义
- 包含虚拟机的命名空间
- 基本虚拟机定义

默认情况下，目前不收集实例类型信息；但是，您可以运行一个命令来选择性地收集它。

流程

- 运行以下命令来收集有关 OpenShift Virtualization 的数据：

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.3 \
  -- /usr/bin/gather
```

13.2.3.1. must-gather 工具选项

您可以运行 **oc adm must-gather** 命令来收集集群中部署的所有 Operator 和产品的 **must gather** 镜像，而无需明确指定所需的镜像。另外，您可以为以下选项指定脚本和环境变量组合：

- 从命名空间收集详细虚拟机 (VM) 信息
- 收集指定虚拟机的详细信息
- 收集镜像、image-stream 和 image-stream-tags 信息
- 限制 **must-gather** 工具使用的最大并行进程数

13.2.3.1.1. 参数

环境变量

您可以为兼容脚本指定环境变量。

NS=<namespace_name>

从您指定的命名空间中收集虚拟机信息，包括 **virt-launcher** pod 详情。为所有命名空间收集 **VirtualMachine** 和 **VirtualMachineInstance** CR 数据。

VM=<vm_name>

收集特定虚拟机的详情。要使用这个选项，还必须使用 **NS** 环境变量指定命名空间。

PROS=<number_of_processes>

修改 **must-gather** 工具使用的最大并行进程数。默认值为 **5**。

重要

使用太多的并行进程可能会导致性能问题。不建议增加并行进程的最大数量。

脚本

每个脚本仅与某些环境变量组合兼容。

/usr/bin/gather

使用默认 **must-gather** 脚本，从所有命名空间中收集集群数据，且仅包含基本的虚拟机信息。此脚本只与 **PROS** 变量兼容。

/usr/bin/gather --vms_details

收集属于 OpenShift Virtualization 资源的虚拟机日志文件、虚拟机定义、control-plane 日志和命名空间。指定命名空间包括其子对象。如果您在指定命名空间或虚拟机的情况下使用这个参数，**must-gather** 工具会为集群中的所有虚拟机收集这个数据。此脚本与所有环境变量兼容，但是如果使用 **VM** 变量，则必须指定一个命名空间。

/usr/bin/gather --images

收集镜像、image-stream 和 image-stream-tags 自定义资源信息。此脚本只与 **PROS** 变量兼容。

/usr/bin/gather --instancetypes

收集实例类型信息。默认情况下，目前不收集这些信息；但是，您可以选择性地收集这些信息。

13.2.3.1.2. 使用和示例

环境变量是可选的。您可以自行运行脚本，也可以使用一个或多个兼容环境变量来运行脚本。

表 13.2. 兼容参数

脚本	兼容环境变量
/usr/bin/gather	* **PROS=<number_of_processes>**
/usr/bin/gather --vms_details	* 对于命名空间：**NS=<namespace_name>** * 对于虚拟机：**VM=<vm_name> NS=<namespace_name>** * **PROS=<number_of_processes>**
/usr/bin/gather --images	* **PROS=<number_of_processes>**

语法

要在单个传递中为集群中的所有 Operator 和产品收集 **must-gather** 日志，请运行以下命令：

```
$ oc adm must-gather --all-images
```

如果需要将额外的参数传递给独立的 **must-gather** 镜像，请使用以下命令：

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.3 \
  -- <environment_variable_1> <environment_variable_2> <script_name>
```

默认数据收集并行进程

默认情况下，五个进程并行运行。

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.3 \
  -- PROS=5 /usr/bin/gather 1
```

1 您可以通过更改默认值来修改并行进程的数量。

详细虚拟机信息

以下命令在 **mynamespace** 命名空间中收集 **my-vm** 虚拟机的详细虚拟机信息：

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.3 \
  -- NS=mynamespace VM=my-vm /usr/bin/gather --vms_details 1
```

1 如果您使用 **VM** 环境变量，则需要 **NS** 环境变量。

image、image-stream 和 image-stream-tags 信息

以下命令从集群中收集镜像、image-stream 和 image-stream-tags 信息：

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.3 \
  /usr/bin/gather --images
```

实例类型信息

以下命令从集群收集实例类型信息：

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.3 \
  /usr/bin/gather --instancetypes
```

13.3. 故障排除

OpenShift Virtualization 为虚拟机 (VM) 和虚拟化组件故障排除提供工具和日志。

您可以使用 web 控制台中提供的工具或使用 **oc** CLI 工具排除 OpenShift Virtualization 组件的问题。

13.3.1. 事件

OpenShift Container Platform 事件是重要生命周期信息的记录，有助于监控虚拟机、命名空间和资源问题。

- VM 事件 ：进入 web 控制台中的 **VirtualMachine 详情**页的 **Events** 选项卡。

  命名空间事件

  您可以运行以下命令来查看命名空间事件：

  ```
  $ oc get events -n <namespace>
  ```

  有关特定事件的详情，请查看事件列表。

  资源事件

  您可以运行以下命令来查看资源事件：

  ```
  $ oc describe <resource> <resource_name>
  ```

13.3.2. Pod 日志

您可以使用 Web 控制台或 CLI 查看 OpenShift Virtualization pod 的日志。您还可以在 web 控制台中使用 LokiStack 查看 聚合的日志。

13.3.2.1. 配置 OpenShift Virtualization pod 日志详细程度

您可以通过编辑 **HyperConverged** 自定义资源 (CR) 来配置 OpenShift Virtualization pod 日志的详细程度。

流程

1. 要为特定组件设置日志详细程度，请运行以下命令在默认文本编辑器中打开 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 通过编辑 **spec.logVerbosityConfig** 小节，为一个或多个组件设置日志级别。例如：

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  logVerbosityConfig:
    kubevirt:
      virtAPI: 5 (1)
      virtController: 4
      virtHandler: 3
      virtLauncher: 2
      virtOperator: 6
```

(1) 日志详细程度值必须是范围 **1-9** 中的一个整数，其中较高的数字表示更详细的日志。在本例中，如果优先级级别为 **5** 或更高版本，则 **virtAPI** 组件日志会公开。

3. 通过保存并退出编辑器来应用您的更改。

13.3.2.2. 使用 web 控制台查看 virt-launcher pod 日志

您可以使用 OpenShift Container Platform web 控制台查看虚拟机的 **virt-launcher** pod 日志。

流程

1. 进入到 **Virtualization → VirtualMachines**。
2. 选择虚拟机以打开 **VirtualMachine 详情**页面。
3. 在 **General** 标题中，点 pod 名称打开 **Pod 详情**页面。
4. 点 **Logs** 选项卡查看日志。

13.3.2.3. 使用 CLI 查看 OpenShift Virtualization pod 日志

您可以使用 **oc** CLI 工具查看 OpenShift Virtualization pod 的日志。

流程

1. 运行以下命令，查看 OpenShift Virtualization 命名空间中的 pod 列表：

```
$ oc get pods -n openshift-cnv
```

例 13.1. 输出示例

```
NAME                               READY   STATUS    RESTARTS   AGE
disks-images-provider-7gqbc        1/1     Running   0          32m
disks-images-provider-vg4kx        1/1     Running   0          32m
virt-api-57fcc4497b-7qfmc          1/1     Running   0          31m
virt-api-57fcc4497b-tx9nc          1/1     Running   0          31m
```

```
virt-controller-76c784655f-7fp6m   1/1     Running   0          30m
virt-controller-76c784655f-f4pbd   1/1     Running   0          30m
virt-handler-2m86x                 1/1     Running   0          30m
virt-handler-9qs6z                 1/1     Running   0          30m
virt-operator-7ccfdbf65f-q5snk     1/1     Running   0          32m
virt-operator-7ccfdbf65f-vllz8     1/1     Running   0          32m
```

2. 运行以下命令来查看 pod 日志：

```
$ oc logs -n openshift-cnv <pod_name>
```

注意

如果 pod 无法启动，您可以使用 **--previous** 选项查看最后一次尝试的日志。

要实时监控日志输出，请使用 **-f** 选项。

例 13.2. 输出示例

```
{"component":"virt-handler","level":"info","msg":"set verbosity to 2","pos":"virt-
handler.go:453","timestamp":"2022-04-17T08:58:37.373695Z"}
{"component":"virt-handler","level":"info","msg":"set verbosity to 2","pos":"virt-
handler.go:453","timestamp":"2022-04-17T08:58:37.373726Z"}
{"component":"virt-handler","level":"info","msg":"setting rate limiter to 5 QPS and 10
Burst","pos":"virt-handler.go:462","timestamp":"2022-04-17T08:58:37.373782Z"}
{"component":"virt-handler","level":"info","msg":"CPU features of a minimum baseline CPU
model: map[apic:true clflush:true cmov:true cx16:true cx8:true de:true fpu:true fxsr:true
lahf_lm:true lm:true mca:true mce:true mmx:true msr:true mtrr:true nx:true pae:true
pat:true pge:true pni:true pse:true pse36:true sep:true sse:true sse2:true sse4.1:true
ssse3:true syscall:true tsc:true]","pos":"cpu_plugin.go:96","timestamp":"2022-04-
17T08:58:37.390221Z"}
{"component":"virt-handler","level":"warning","msg":"host model mode is expected to
contain only one model","pos":"cpu_plugin.go:103","timestamp":"2022-04-
17T08:58:37.390263Z"}
{"component":"virt-handler","level":"info","msg":"node-labeller is
running","pos":"node_labeller.go:94","timestamp":"2022-04-17T08:58:37.391011Z"}
```

13.3.3. 客户端系统日志

查看虚拟机客户机的引导日志可帮助诊断问题。您可以配置对客户机日志的访问，并使用 OpenShift Container Platform Web 控制台或 **oc** CLI 查看它们。

此功能默认为禁用。如果虚拟机没有明确启用或禁用此设置，它会继承集群范围的默认设置。

重要

如果凭据或其他个人可识别信息(PI)等敏感信息被写入串行控制台，则会使用所有其他可见文本记录。红帽建议使用 SSH 发送敏感数据，而不是串行控制台。

13.3.3.1. 使用 web 控制台启用对虚拟机客户机系统日志的默认访问

您可以使用 Web 控制台启用对虚拟机客户机系统日志的默认访问。

流程

1. 在侧边菜单中点 **Virtualization → Overview**。
2. 点 **Settings** 选项卡。
3. 点 **Cluster → Guest Management**。
4. 设置 **Enable guest system log access**为 on。

13.3.3.2. 使用 CLI 启用虚拟机客户机系统日志的默认访问

您可以通过编辑 **HyperConverged** 自定义资源 (CR) 来启用对虚拟机客户端系统日志的默认访问。

流程

1. 运行以下命令，在默认编辑器中打开 **HyperConverged** CR：

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 更新 **disableSerialConsoleLog** 值。例如：

```
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  virtualMachineOptions:
    disableSerialConsoleLog: true 1
#...
```

1 如果您希望默认启用串口控制台访问，请将 **disableSerialConsoleLog** 的值设置为 **false**。

13.3.3.3. 使用 Web 控制台为单个虚拟机设置客户机系统日志访问

您可以使用 Web 控制台为单个虚拟机配置对虚拟机客户机系统日志的访问。此设置优先于集群范围的默认配置。

流程

1. 在侧边菜单中点 **Virtualization → VirtualMachines**。
2. 选择虚拟机以打开 **VirtualMachine 详情**页面。
3. 点 **Configuration** 选项卡。
4. 将 **Guest system log access**设置为 on 或 off。

13.3.3.4. 使用 CLI 为单个虚拟机设置客户机系统日志访问

您可以通过编辑 **VirtualMachine** CR 来为单个虚拟机配置对虚拟机客户机系统日志的访问。此设置优先于集群范围的默认配置。

流程

1. 运行以下命令来编辑虚拟机清单：

```
$ oc edit vm <vm_name>
```

2. 更新 **logSerialConsole** 字段的值。例如：

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          logSerialConsole: true 1
#...
```

1 要启用对客户机的串行控制台日志的访问，请将 **logSerialConsole** 值设置为 **true**。

3. 运行以下命令，将新配置应用到虚拟机：

```
$ oc apply vm <vm_name>
```

4. 可选：如果您编辑了正在运行的虚拟机，重启虚拟机以应用新配置。例如：

```
$ virtctl restart <vm_name> -n <namespace>
```

13.3.3.5. 使用 Web 控制台查看客户机系统日志

您可以使用 web 控制台查看虚拟机(VM)客户机的串行控制台日志。

先决条件

- 启用客户机系统日志访问。

流程

1. 在侧边菜单中点 **Virtualization → VirtualMachines**。
2. 选择虚拟机以打开 **VirtualMachine 详情**页面。
3. 点 **Diagnostics** 选项卡。
4. 点 **Guest system logs** 以加载串行控制台。

13.3.3.6. 使用 CLI 查看客户机系统日志

您可以通过运行 **oc logs** 命令来查看虚拟机客户机的串行控制台日志。

先决条件

- 启用客户机系统日志访问。

流程

- 运行以下命令来查看日志，使用您的值替换 **<namespace>** 和 **<vm_name>**：

```
$ oc logs -n <namespace> -l kubevirt.io/domain=<vm_name> --tail=-1 -c guest-console-log
```

13.3.4. 日志聚合

您可以通过聚合和过滤日志来促进故障排除。

13.3.4.1. 使用 LokiStack 查看聚合的 OpenShift Virtualization 日志

您可以使用 web 控制台中的 LokiStack 查看 OpenShift Virtualization pod 和容器的聚合日志。

先决条件

- 您已部署了 LokiStack。

流程

1. 在 web 控制台中进入 **Observe → Logs**。
2. 从日志类型列表中选择 **应用程序**，对于 **virt-launcher** pod 日志或**基础架构**，为 OpenShift Virtualization control plane pod 和容器选择应用程序。
3. 点 **Show Query** 以显示查询字段。
4. 在查询字段中输入 LogQL 查询，然后点 **Run Query** 以显示过滤的日志。

13.3.4.2. OpenShift Virtualization LogQL 查询

您可以通过在 web 控制台的 **Observe → Logs** 页面中运行 Loki Query Language (LogQL) 查询来查看和过滤 OpenShift Virtualization 组件的聚合日志。

默认日志类型是 *infrastructure*。**virt-launcher** 日志类型是 *application*。

可选：您可以使用行过滤器表达式来包含或排除字符串或正则表达式。

注意

如果查询与大量日志匹配，查询可能会超时。

表 13.3. OpenShift Virtualization LogQL 示例查询

组件	LogQL 查询

组件	LogQL 查询
All	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster"
cdi-apiserver **cdi-deployment** **cdi-operator**	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="storage"
hco-operator	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="deployment"
kubemacpool	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="network"
virt-api **virt-controller** **virt-handler** **virt-operator**	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="compute"
ssp-operator	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="schedule"
Container	{log_type=~".+",kubernetes_container_name=~"<container>|<container>"} (1) |json|kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster"

(1) 指定一个或多个以管道分开的容器 (|)。

组件	LogQL 查询
virt-launcher	在运行此查询前，您必须从日志类型列表中选择 **应用程序**。 {log_type=~".+", kubernetes_container_name="compute"}|json |!= "custom-ga-command" (1) (1) **|!= "custom-ga-command"** 排除包含字符串 **custom-ga-command** 的 libvirt 日志。(BZ#2177684)

您可以使用行过滤器表达式过滤日志行使其包含或排除字符串或正则表达式。

表 13.4. 行过滤器表达式

行过滤器表达式	描述
|= "<string>"	日志行包含字符串
!= "<string>"	日志行不包含字符串
|~ "<regex>"	日志行包含正则表达式
!~ "<regex>"	日志行不包含正则表达式

行过滤器表达式示例

```
{log_type=~".+"}|json
|kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster"
|= "error" != "timeout"
```

LokiStack 和 LogQL 的其他资源

- 关于日志存储
- 部署 Lokistack
- Grafana 文档中的 LogQL 日志查询

13.3.5. 常见错误消息

OpenShift Virtualization 日志中可能会出现以下出错信息：

ErrImagePull 或 ImagePullBackOff

表示引用镜像的部署配置或问题。

13.3.6. 数据卷故障排除

您可以检查 **DataVolume** 对象的 **Conditions** 和 **Events** 部分，以分析并解决问题。

13.3.6.1. 关于数据卷条件和事件

您可以通过检查命令生成的 **Conditions** 和 **Events** 部分的输出来诊断数据卷的问题：

```
$ oc describe dv <DataVolume>
```

Conditions 部分显示以下 类**型** ：

- **Bound**
- **Running**
- **Ready**

Events 部分提供以下额外信息：

- 事件类**型**
- 日志**原因**
- 事件**源**
- 包含其他诊断信息的**消息**。

oc describe 的输出并不总是包含 **Events**。

当 **Status**、**Reason** 或 **Message** 改变时会产生一个事件。条件和事件均响应数据卷状态的变化。

例如，在导入操作中错误拼写了 URL，则导入会生成 404 信息。该消息的更改会生成一个带有原因的事件。**Conditions** 部分中的输出也会更新。

13.3.6.2. 分析数据卷条件和事件

通过检查 **describe** 命令生成的 **Conditions** 和 **Events** 部分，您可以确定与 PVC相关的数据卷的状态，以及某个操作是否正在主动运行或完成。您可能还会收到信息，它们提供了有关数据卷状态的特定详情，以及如何处于当前状态。

有多种条件的组合。对每个条件组合的评估都必须其特定的环境下进行。

下面是各种组合的例子。

- **Bound** - 本示例中会显示成功绑定 PVC。
  请注意, **Type** 是 **Bound**,所以 **Status** 为 **True**。如果 PVC 没有绑定，**Status** 为 **False**。

  当 PVC 被绑定时，会生成一个事件声明 PVC 已被绑定。在本例中, **Reason** 为 **Bound**，**Status** 为 **True**。**Message** 指明了哪个 PVC 拥有数据卷。

  在 **Events** 部分，**Message** 提供了更多详细信息，包括 PVC 被绑定的时间（**Age**）和它的源（**From**），在本例中是 **datavolume-controller**:

  输出示例

  ```
  Status:
    Conditions:
      Last Heart Beat Time:  2020-07-15T03:58:24Z
      Last Transition Time:  2020-07-15T03:58:24Z
  ```

```
    Message:               PVC win10-rootdisk Bound
    Reason:                Bound
    Status:                True
    Type:                  Bound
...
  Events:
    Type     Reason     Age    From                Message
    ----     ------     ----   ----                -------
    Normal   Bound      24s    datavolume-controller  PVC example-dv Bound
```

- **Running** - 在本例中，请注意 **Type** 是 **Running**，**Status** 为 **False**。这表示发生事件导致尝试的操作失败，将 Status 从 **True** 改为 **False**。
  然而，请注意 **Reason** 是 **Completed**，**Message** 显示 **Import Complete**。

  在 **Events** 部分，**Reason** 和 **Message** 包含有关失败操作的额外故障排除信息。在这个示例中，**Message** 显示因为 **404** 无法连接，这在 **Events** 部分的第一个 **Warning** 中列出。

  根据这些信息，您认为导入操作正在运行，并为试图访问数据卷的其他操作创建竞争：

  输出示例

```
Status:
  Conditions:
    Last Heart Beat Time:  2020-07-15T04:31:39Z
    Last Transition Time:  2020-07-15T04:31:39Z
    Message:               Import Complete
    Reason:                Completed
    Status:                False
    Type:                  Running
...
  Events:
    Type     Reason       Age              From                Message
    ----     ------       ----             ----                -------
    Warning  Error        12s (x2 over 14s)  datavolume-controller  Unable to connect
    to http data source: expected status code 200, got 404. Status: 404 Not Found
```

- **Ready** - 如果 **Type** 是 **Ready**，**Status** 为**True**，则代表数据卷已就绪，如下例所示。如果数据卷未就绪，则 **Status** 为 **False**:

  输出示例

```
Status:
  Conditions:
    Last Heart Beat Time: 2020-07-15T04:31:39Z
    Last Transition Time:  2020-07-15T04:31:39Z
    Status:                True
    Type:                  Ready
```

第 14 章 备份和恢复

14.1. 使用虚拟机快照备份和恢复

您可以使用快照备份和恢复虚拟机(VM)。以下存储供应商支持快照：

- Red Hat OpenShift Data Foundation
- 使用支持 Kubernetes 卷快照 API 的 Container Storage Interface（CSI）驱动程序的任何其他云存储供应商

在线快照的默认时间期限为五分钟(**5m**)，可根据需要进行更改。

重要

具有热插虚拟磁盘的虚拟机支持在线快照。但是，没有在虚拟机规格中的热插磁盘不会包含在快照中。

要创建具有最高完整性的在线(Running 状态)虚拟机的快照，请在您的操作系统中没有包括 QEMU 客户机代理时安装它。QEMU 客户机代理包含在默认的红帽模板中。

QEMU 客户机代理通过尝试静止虚拟机的文件系统来尽可能取一个一致的快照，具体取决于系统工作负载。这样可确保在进行快照前将 in-flight I/O 写入磁盘。如果没有客户机代理，则无法静止并生成最佳快照。执行快照的条件反映在 web 控制台或 CLI 中显示的快照声明中。

14.1.1. 关于快照

*快照*代表虚拟机（VM）在特定时间点的状态和数据。您可以使用快照将现有虚拟机恢复到以前的状态（由快照代表）进行备份和恢复，或者快速回滚到以前的开发版本。

虚拟机快照从关机（停止状态）或 powred on（Running 状态）上的虚拟机创建。

在为正在运行的虚拟机执行快照时，控制器将检查 QEMU 客户机代理是否已安装并在运行。如果是这样，它会在拍摄快照前冻结虚拟机文件系统，并在拍摄快照后修改文件系统。

快照存储附加到虚拟机的每个 Container Storage Interface（CSI）卷的副本以及虚拟机规格和元数据的副本。创建后无法更改快照。

您可以执行以下快照操作：

- 创建新快照
- 从快照创建虚拟机的副本
- 列出附加到特定虚拟机的所有快照
- 从快照恢复虚拟机
- 删除现有虚拟机快照

VM 快照控制器和自定义资源

VM 快照功能引入了三个新的 API 对象，定义为自定义资源定义(CRD)来管理快照：

- **VirtualMachineSnapshot**:代表创建快照的用户请求。它包含有关虚拟机当前状态的信息。

- **VirtualMachineSnapshotContent**:代表集群中置备的资源（快照）。它由虚拟机快照控制器创建，其中包含恢复虚拟机所需的所有资源的引用。
- **VirtualMachineRestore**:代表从快照中恢复虚拟机的用户请求。

VM 快照控制器会把一个 **VirtualMachineSnapshotContent** 对象与创建它的 **VirtualMachineSnapshotContent** 对象绑定，并具有一对一的映射。

14.1.2. 关于应用程序一致性快照和备份

您可以通过冻结和解译周期，为 Linux 或 Windows 虚拟机(VM)配置应用程序一致性快照和备份。对于任何应用程序，您可以在 Linux 虚拟机上配置脚本，或者在 Windows 虚拟机上注册，以便在快照或备份开始时获得通知。

在 Linux 虚拟机上，当生成快照或使用备份时自动触发 GFS 和 thaw 进程，例如，来自 Velero 或其他备份供应商的插件。QEMU 客户机代理(QEMU GA)执行的冻结过程会冻结 hook，确保在虚拟机的快照或备份发生前，所有虚拟机的文件系统都会冻结，每个配置的应用程序都会显示快照或备份要启动。此通知可负担每个应用程序获得其状态的机会。根据应用程序，静默可能涉及临时拒绝新请求、完成进行中的操作，以及将数据刷新到磁盘。然后，操作系统会被定向到静止文件系统，方法是清除未完成的写入磁盘并释放新的写入活动。所有新的连接请求都会被拒绝。当所有应用程序都不活跃时，QEMU GA 会冻结文件系统，并生成一个快照或启动备份。在执行快照或开始备份后，解动过程开始。文件系统写入会被重新激活，应用程序会收到通知以恢复正常操作。

Windows 虚拟机上提供了相同的冻结和波动周期。使用卷 Shadow Copy Service (VSS)注册应用程序，以接收通知，因为备份或快照不会清除其数据。在备份或快照完成后，对应用程序进行修复会将它们返回到 active 状态。如需了解更多详细信息，请参阅有关卷 Shadow 复制服务的 Windows 服务器文档。

14.1.3. 创建快照

您可以使用 OpenShift Container Platform web 控制台或命令行创建虚拟机(VM)的快照。

14.1.3.1. 使用 Web 控制台创建快照

您可以使用 OpenShift Container Platform web 控制台为虚拟机(VM)创建快照。

VM 快照包括以下要求的磁盘：

- 数据卷或持久性卷声明
- 属于支持容器存储接口（CSI）卷快照的存储类

流程

1. 在 web 控制台中进入到 **Virtualization → VirtualMachines**。
2. 选择一个虚拟机以打开 **VirtualMachine 详情**页。
3. 点 **Snapshots** 标签页，然后点 **Take Snapshot**。
4. 输入快照名称。
5. 扩展 **Disks included in this Snapshot**以查看快照中包含的存储卷。
6. 如果您的虚拟机有无法包含在快照中的磁盘，并且您希望继续，请选择 **I am aware of this warning and wish to proceed**。

7. 点击 **Save**。

14.1.3.2. 使用命令行创建快照

您可以通过创建一个 **VirtualMachineSnapshot** 对象来为离线或在线虚拟机创建虚拟机快照。

先决条件

- 确保持久性卷声明（PVC）位于支持 Container Storage Interface（CSI）卷快照的存储类中。
- 安装 OpenShift CLI (**oc**) 。
- 可选：关闭您要为其创建快照的虚拟机。

流程

1. 创建一个 YAML 文件来定义 **VirtualMachineSnapshot** 对象，用于指定新 **VirtualMachineSnapshot** 的名称和源虚拟机的名称，如下例所示：

```
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineSnapshot
metadata:
  name: <snapshot_name>
spec:
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: <vm_name>
```

2. 创建 **VirtualMachineSnapshot** 对象：

```
$ oc create -f <snapshot_name>.yaml
```

快照控制器会创建一个 **VirtualMachineSnapshotContent** 对象，将其绑定到 **VirtualMachineSnapshot**，并更新 **VirtualMachineSnapshot** 对象的 **status** 和 **readyToUse** 字段。

3. 可选： 如果要执行在线快照，您可以使用 **wait** 命令并监控快照的状态：

   a. 输入以下命令：

```
$ oc wait <vm_name> <snapshot_name> --for condition=Ready
```

   b. 验证快照的状态：

   - **InProgress** - 在线快照操作仍在进行中。
   - **succeeded** - 在线快照操作成功完成。
   - **Failed** - 在线快照操作失败。

注意

在线快照的默认时间期限为五分钟(**5m**)。如果快照在五分钟内没有成功完成，其状态将设为 **failed**。之后，文件系统将被"解冻"，虚拟机将取消冻结，但状态会一直 **failed**，直到您删除失败的快照镜像。

要更改默认时间期限，在 VM 快照 spec 中添加 **FailureDeadline** 属性，指定在快照超时前的时间，以分钟(**m**)或秒(**s**)为单位。

要设置截止时间，您可以指定 **0**，但通常不建议这样做，因为它可能会导致虚拟机没有响应。

如果您没有指定时间单位，如 **m** 或 **s**，则默认为 秒(**s**)。

验证

1. 验证 **VirtualMachineSnapshot** 对象是否已创建并绑定到 **VirtualMachineSnapshotContent**，并且 **readyToUse** 标志设为 **true** ：

```
$ oc describe vmsnapshot <snapshot_name>
```

输出示例

```
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineSnapshot
metadata:
  creationTimestamp: "2020-09-30T14:41:51Z"
  finalizers:
  - snapshot.kubevirt.io/vmsnapshot-protection
  generation: 5
  name: mysnap
  namespace: default
  resourceVersion: "3897"
  selfLink:
/apis/snapshot.kubevirt.io/v1alpha1/namespaces/default/virtualmachinesnapshots/my-
vmsnapshot
  uid: 28eedf08-5d6a-42c1-969c-2eda58e2a78d
spec:
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: my-vm
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2020-09-30T14:42:03Z"
    reason: Operation complete
    status: "False" (1)
    type: Progressing
  - lastProbeTime: null
    lastTransitionTime: "2020-09-30T14:42:03Z"
    reason: Operation complete
    status: "True" (2)
    type: Ready
  creationTime: "2020-09-30T14:42:03Z"
```

```
    readyToUse: true 3
    sourceUID: 355897f3-73a0-4ec4-83d3-3c2df9486f4f
    virtualMachineSnapshotContentName: vmsnapshot-content-28eedf08-5d6a-42c1-969c-
  2eda58e2a78d 4
```

1. **Progressing** 的 **status** 字段指定快照是否仍然在创建。
2. **Ready** 条件的 **status** 字段指定快照创建过程是否完成。
3. 指定快照是否准备就绪可用被使用。
4. 指定快照被绑定到快照控制器创建的 **VirtualMachineSnapshotContent** 对象。

2. 检查 **VirtualMachineSnapshotContent** 资源的 **spec:volumeBackups** 属性，以验证快照中包含了预期的 PVC。

14.1.4. 使用快照指示验证在线快照

快照表示是有关在线虚拟机 (VM) 快照操作的上下文信息。对于离线虚拟机 (VM) 快照操作，提示不可用。暗示有助于描述在线快照创建的详细信息。

先决条件

- 您必须已尝试创建在线虚拟机快照。

流程

1. 通过执行以下操作之一来显示快照的输出：
   - 使用命令行查看 **VirtualMachineSnapshot** 对象 YAML 的 **status** 小节中的指示符输出。
   - 在 web 控制台中，在 **Snapshot details** 屏幕中点 **VirtualMachineSnapshot → Status**。
2. 通过查看 **status.indications** 参数的值来验证在线虚拟机快照的状态：
   - **Online** 代表虚拟机在在线快照创建期间运行。
   - **GuestAgent** 表示 QEMU 客户机代理在在线快照创建过程中运行。
   - **NoGuestAgent** 表示 QEMU 客户机代理在在线快照创建过程中没有运行。QEMU 客户机代理无法用于冻结和构建文件系统，要么因为 QEMU 客户机代理尚未安装或正在运行，要么是因为另一个错误。

14.1.5. 从快照中恢复虚拟机

您可以使用 OpenShift Container Platform web 控制台或命令行从快照中恢复虚拟机(VM)。

14.1.5.1. 使用 Web 控制台从快照中恢复虚拟机

您可以将虚拟机(VM)恢复到 OpenShift Container Platform web 控制台中的快照代表的以前的配置。

流程

1. 在 web 控制台中进入到 **Virtualization → VirtualMachines**。

2. 选择一个虚拟机以打开 VirtualMachine 详情页。
3. 如果虚拟机正在运行，点选项菜单  并选择 Stop 关闭它。
4. 点 Snapshots 选项卡查看与虚拟机关联的快照列表。
5. 选择快照以打开 Snapshot Details 屏幕。
6. 点选项菜单  并选择 Restore VirtualMachine from snapshot。
7. 单击 Restore。

14.1.5.2. 使用命令行从快照中恢复虚拟机

您可以使用命令行将现有虚拟机(VM)恢复到以前的配置。您只能从离线虚拟机快照中恢复。

先决条件

- 关闭您要恢复的虚拟机。

流程

1. 创建一个 YAML 文件来定义 **VirtualMachineRestore** 对象，用于指定您要恢复的虚拟机的名称以及要用作源的快照名称，如下例所示：

```
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineRestore
metadata:
  name: <vm_restore>
spec:
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: <vm_name>
  virtualMachineSnapshotName: <snapshot_name>
```

2. 创建 **VirtualMachineRestore** 对象：

```
$ oc create -f <vm_restore>.yaml
```

快照控制器更新了 **VirtualMachineRestore** 对象的 status 字段，并将现有虚拟机配置替换为快照内容。

验证

- 验证虚拟机是否已恢复到快照代表的以前的状态，并将 **complete** 标志设置为 **true** ：

```
$ oc get vmrestore <vm_restore>
```

输出示例

```
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineRestore
metadata:
creationTimestamp: "2020-09-30T14:46:27Z"
generation: 5
name: my-vmrestore
namespace: default
ownerReferences:
- apiVersion: kubevirt.io/v1
  blockOwnerDeletion: true
  controller: true
  kind: VirtualMachine
  name: my-vm
  uid: 355897f3-73a0-4ec4-83d3-3c2df9486f4f
  resourceVersion: "5512"
  selfLink:
/apis/snapshot.kubevirt.io/v1alpha1/namespaces/default/virtualmachinerestores/my-
vmrestore
  uid: 71c679a8-136e-46b0-b9b5-f57175a6a041
  spec:
    target:
      apiGroup: kubevirt.io
      kind: VirtualMachine
      name: my-vm
  virtualMachineSnapshotName: my-vmsnapshot
  status:
  complete: true (1)
  conditions:
  - lastProbeTime: null
  lastTransitionTime: "2020-09-30T14:46:28Z"
  reason: Operation complete
  status: "False" (2)
  type: Progressing
  - lastProbeTime: null
  lastTransitionTime: "2020-09-30T14:46:28Z"
  reason: Operation complete
  status: "True" (3)
  type: Ready
  deletedDataVolumes:
  - test-dv1
  restoreTime: "2020-09-30T14:46:28Z"
  restores:
  - dataVolumeName: restore-71c679a8-136e-46b0-b9b5-f57175a6a041-datavolumedisk1
  persistentVolumeClaim: restore-71c679a8-136e-46b0-b9b5-f57175a6a041-
datavolumedisk1
  volumeName: datavolumedisk1
  volumeSnapshotName: vmsnapshot-28eedf08-5d6a-42c1-969c-2eda58e2a78d-volume-
datavolumedisk1
```

1. 指定将虚拟机恢复到快照代表的状态的进程是否已完成。
2. **Progressing** 条件的 **status** 字段指定 VM 是否仍然被恢复。
3. **Ready** 条件的 **status** 字段指定 VM 恢复过程是否完成。

14.1.6. 删除快照

您可以使用 OpenShift Container Platform web 控制台或命令行删除虚拟机快照。

14.1.6.1. 使用 Web 控制台删除快照

您可以使用 web 控制台删除现有虚拟机(VM)快照。

流程

1. 在 web 控制台中进入到 **Virtualization → VirtualMachines**。
2. 选择一个虚拟机以打开 **VirtualMachine 详情**页。
3. 点 **Snapshots** 选项卡查看与虚拟机关联的快照列表。
4. 点快照旁的选项菜单  并选择 **Delete snapshot**。
5. 点击 **Delete**。

14.1.6.2. 通过 CLI 删除虚拟机快照

您可以通过删除正确的 **VirtualMachineSnapshot** 对象来删除现有虚拟机（VM）快照。

先决条件

- 安装 OpenShift CLI (**oc**) 。

流程

- 删除 **VirtualMachineSnapshot** 对象：

```
$ oc delete vmsnapshot <snapshot_name>
```

快照控制器会删除 **VirtualMachineSnapshot** 和关联的 **VirtualMachineSnapshotContent** 对象。

验证

- 验证快照是否已删除，且不再附加到此虚拟机：

```
$ oc get vmsnapshot
```

14.1.7. 其他资源

- CSI 卷快照

14.2. 备份和恢复虚拟机

重要

红帽支持在 OADP 1.3.x 或更高版本中使用 OpenShift Virtualization 4.14 或更高版本。

1.3.0 之前的 OADP 版本不支持备份和恢复 OpenShift Virtualization。

使用 OpenShift API for Data Protection 来备份和恢复虚拟机。

您可以通过安装 OADP Operator 并配置备份位置，使用 OpenShift Virtualization 安装 OpenShift API for Data Protection (OADP)。然后您可以安装数据保护应用程序。

注意

OpenShift API for Data Protection with OpenShift Virtualization 支持以下备份和恢复存储选项：

- 容器存储接口 (CSI) 备份
- 使用 DataMover 进行容器存储接口 (CSI) 备份

排除以下存储选项：

- 文件系统备份和恢复
- 卷快照备份和恢复

如需更多信息，请参阅使用文件系统备份备份应用程序： Kopia 或 Restic。

要在受限网络环境中安装 OADP Operator，您必须首先禁用默认的 OperatorHub 源并镜像 Operator 目录。

详情请参阅在受限网络中使用 Operator Lifecycle Manager。

14.2.1. 使用 OpenShift Virtualization 安装和配置 OADP

作为集群管理员，您可以通过安装 OADP Operator 来安装 OADP。

OADP Operator 的最新版本会安装 Velero 1.14。

先决条件

- 使用具有 **cluster-admin** 角色的用户访问集群。

流程

1. 根据您的存储供应商说明安装 OADP Operator。
2. 使用 **kubevirt** 和 **openshift** OADP 插件安装数据保护应用程序(DPA)。
3. 通过创建 **Backup** 自定义资源(CR) 来备份虚拟机。

警告

红帽支持仅限于以下选项：

- CSI 备份
- 使用 DataMover 的 CSI 备份。

您可以通过创建一个 **Restore** CR来恢复 **Backup** CR。

其他资源

- OADP 插件
- **Backup** 自定义资源 (CR)
- **恢复** CR
- 在受限网络中使用 Operator Lifecycle Manager

14.2.2. 安装数据保护应用程序 1.3

您可以通过创建 **DataProtectionApplication** API 的实例来安装数据保护应用程序(DPA)。

先决条件

- 您必须安装 OADP Operator。
- 您必须将对象存储配置为备份位置。
- 如果使用快照来备份 PV，云供应商必须支持原生快照 API 或 Container Storage Interface(CSI) 快照。
- 如果备份和快照位置使用相同的凭证，您必须创建带有默认名称 **cloud-credentials** 的 **Secret**。
- 如果备份和快照位置使用不同的凭证，您必须创建两个 **Secret** ：
  - 带有备份位置的自定义名称的 **secret**。您可以将此 **Secret** 添加到 **DataProtectionApplication** CR 中。
  - 带有快照位置的另一个自定义名称的 **Secret**。您可以将此 **Secret** 添加到 **DataProtectionApplication** CR 中。

注意

如果您不想在安装过程中指定备份或快照位置，您可以使用空 **credentials-velero** 文件创建默认 **Secret**。如果没有默认 **Secret**，安装将失败。

流程

1. 点 **Operators → Installed Operators** 并选择 OADP Operator。

2. 在 **Provided APIs** 下，点 **DataProtectionApplication** 框中的 **Create 实例**。

3. 点 **YAML View** 并更新 **DataProtectionApplication** 清单的参数：

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
  namespace: openshift-adp (1)
spec:
  configuration:
    velero:
      defaultPlugins:
        - kubevirt (2)
        - gcp (3)
        - csi (4)
        - openshift (5)
      resourceTimeout: 10m (6)
    nodeAgent: (7)
      enable: true (8)
      uploaderType: kopia (9)
      podConfig:
        nodeSelector: <node_selector> (10)
  backupLocations:
    - velero:
        provider: gcp (11)
        default: true
        credential:
          key: cloud
          name: <default_secret> (12)
        objectStorage:
          bucket: <bucket_name> (13)
          prefix: <prefix> (14)
```

(1) OADP 的默认命名空间是 **openshift-adp**。命名空间是一个变量，可配置。

(2) OpenShift Virtualization 需要 **kubevirt** 插件。

(3) 为备份供应商指定插件，如 **gcp** （如果存在）。

(4) **csi** 插件是使用 CSI 快照备份 PV 所必需的。 **csi** 插件使用 Velero CSI beta 快照 API。您不需要配置快照位置。

(5) **openshift** 插件是必需的。

(6) 指定在超时发生前等待多个 Velero 资源的分钟，如 Velero CRD 可用、volumeSnapshot 删除和备份存储库可用。默认值为 10m。

(7) 将管理请求路由到服务器的管理代理。

(8) 如果要启用 **nodeAgent** 并执行文件系统备份，则将此值设置为 **true**。

(9) 输入 **kopia** 作为您的上传程序，以使用 Built-in DataMover。 **nodeAgent** 部署守护进程集，这意味着 **nodeAgent** pod 在每个工作节点上运行。您可以通过在 **Backup** CR 中添加 **spec.defaultVolumesToFsBackup: true** 来配置文件系统备份。

10 指定 Kopia 可用的节点。默认情况下，Kopia 在所有节点上运行。

11 指定备份供应商。

12 如果备份供应商使用一个默认插件，为 **Secret** 指定正确的默认名称，如 **cloud-credentials-gcp**。如果指定了一个自定义名称，则使用自定义名称用于备份位置。如果没有指定 **Secret** 名称，则使用默认名称。

13 指定存储桶作为备份存储位置。如果存储桶不是 Velero 备份的专用存储桶，您必须指定一个前缀。

14 如果存储桶用于多个目的，请为 Velero 备份指定一个前缀，如 **velero**。

4. 点 **Create**。

验证

1. 运行以下命令，查看 OpenShift API for Data Protection (OADP) 资源来验证安装：

```
$ oc get all -n openshift-adp
```

输出示例

```
NAME                                                 READY   STATUS    RESTARTS   AGE
pod/oadp-operator-controller-manager-67d9494d47-6l8z8   2/2     Running   0          2m8s
pod/node-agent-9cq4q                                 1/1     Running   0          94s
pod/node-agent-m4lts                                 1/1     Running   0          94s
pod/node-agent-pv4kr                                 1/1     Running   0          95s
pod/velero-588db7f655-n842v                          1/1     Running   0          95s

NAME                                                     TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)    AGE
service/oadp-operator-controller-manager-metrics-service   ClusterIP   172.30.70.140
<none>        8443/TCP   2m8s
service/openshift-adp-velero-metrics-svc                 ClusterIP   172.30.10.0      <none>
8085/TCP   8h

NAME                         DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR   AGE
daemonset.apps/node-agent    3         3         3       3            3           <none>          96s

NAME                                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/oadp-operator-controller-manager   1/1     1            1           2m9s
deployment.apps/velero                             1/1     1            1           96s

NAME                                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/oadp-operator-controller-manager-67d9494d47   1         1         1       2m9s
replicaset.apps/velero-588db7f655                          1         1         1       96s
```

2. 运行以下命令，验证 **DataProtectionApplication** (DPA) 是否已协调：

```
$ oc get dpa dpa-sample -n openshift-adp -o jsonpath='{.status}'
```

输出示例

```
{"conditions":[{"lastTransitionTime":"2023-10-27T01:23:57Z","message":"Reconcile
complete","reason":"Complete","status":"True","type":"Reconciled"}]}
```

3. 验证 **type** 被设置为 **Reconciled**。

4. 运行以下命令，验证备份存储位置并确认 **PHASE** 为 **Available** ：

```
$ oc get backupStorageLocation -n openshift-adp
```

输出示例

```
NAME          PHASE       LAST VALIDATED   AGE     DEFAULT
dpa-sample-1  Available   1s               3d16h   true
```

14.3. 灾难恢复

OpenShift Virtualization 支持使用灾难恢复 (DR) 解决方案来确保您的环境可在站点中断后恢复。要使用这些方法，您必须提前规划 OpenShift Virtualization 部署。

14.3.1. 关于灾难恢复方法

有关灾难恢复(DR)概念、架构和规划注意事项的概述，请参阅红帽知识库中的 Red Hat OpenShift Virtualization 灾难恢复指南。

OpenShift Virtualization 两种主要 DR 方法是 Metropolitan Disaster Recovery (Metro-DR) 和 Regional-DR。

14.3.1.1. Metro-DR

Metro-DR 使用同步复制。它会在主站点和次站点写入存储，以便数据始终在站点间同步。由于存储提供程序负责确保同步成功，环境必须满足存储提供程序的吞吐量和延迟要求。

14.3.1.2. Regional-DR

Regional-DR 使用异步复制。主站点中的数据会定期与二级站点同步。对于这种类型的复制，您可以在主站点和次站点之间具有更高的延迟连接。

重要

Regional-DR 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。

14.3.2. 为灾难恢复定义应用程序

使用 Red Hat Advanced Cluster Management (RHACM) 管理或发现的虚拟机为灾难恢复定义应用程序。

14.3.2.1. 定义 RHACM 管理的虚拟机时的最佳实践

必须使用 GitOps 工作流和创建 RHACM 应用程序或 **ApplicationSet** 来创建包含虚拟机的 RHACM 管理的应用程序。

在定义 RHACM 管理虚拟机时，您可以采取几个操作来改进您的体验和成功的机会。

使用 PVC 和填充器为虚拟机定义存储
因为数据卷隐式创建持久性卷声明 (PVC)，所以带有数据卷模板的数据卷和虚拟机不适合 GitOps 模型。

在为您的虚拟机磁盘选择填充源时使用导入方法
使用导入方法解决 Regional-DR 中的限制，以防止您保护使用克隆的 PVC 的虚拟机。

从软件目录中选择一个 RHEL 镜像以使用导入方法。红帽建议使用特定版本的镜像而不是浮动标签来获得一致的结果。KubeVirt 社区为 Quay 存储库中的其他操作系统维护容器磁盘。

使用 pullMethod: node
从 registry 源创建数据卷时使用 pod **pullMethod: node**，以利用 OpenShift Container Platform pull secret，这是从 Red Hat registry 中拉取容器镜像所必需的。

14.3.2.2. 定义 RHACM 发现的虚拟机时的最佳实践

您可以将不是 RHACM 管理的应用程序的任何虚拟机配置为 RHACM 发现的应用程序。这包括使用 Migration Toolkit for Virtualization (MTV)、使用 OpenShift Virtualization web 控制台创建的虚拟机或由任何其他方法创建的虚拟机（如 CLI）导入的虚拟机。

在定义 RHACM 发现的虚拟机时，您可以采取几个操作来改进您的体验和成功的机会。

在使用 MTV、OpenShift Virtualization web 控制台或自定义虚拟机时保护虚拟机
因为当前还不可用自动标记，应用程序所有者必须在使用 MTV、OpenShift Virtualization web 控制台或自定义虚拟机时手动标记虚拟机应用程序的组件。

创建虚拟机后，将 common 标签应用到与虚拟机关联的以下资源：**VirtualMachine**, **DataVolume**, **PersistentVolumeClaim**, **Service**, **Route**, **Secret**, 和 **ConfigMap**。不要标记虚拟机实例 (VMI) 或 pod，因为 OpenShift Virtualization 自动创建和管理它们。

在虚拟机中包含多于 VirtualMachine 对象
工作虚拟机通常还包含数据卷、持久性卷声明(PVC)、服务、路由、secret、**ConfigMap** 对象和 **VirtualMachineSnapshot** 对象。

将虚拟机作为较大逻辑应用程序的一部分包含
这包括其他基于 pod 的工作负载和虚拟机。

14.3.3. 在灾难恢复场景中虚拟机行为

在重新定位和故障转移灾难恢复流程中，虚拟机通常会与基于 pod 的工作负载类似。

重新定位
当主环境仍可访问时，使用重新定位将应用程序从主环境移到二级环境中。在重新定位过程中，虚拟机可以安全地终止，任何未复制的数据都会同步到二级环境中，虚拟机在二级环境中启动。

导致安全终止，在这种情况下不会有数据丢失。因此，虚拟机操作系统不需要执行崩溃恢复。

故障切换
当主环境中出现关键故障时，请使用故障转移，使其不现实或使用重新定位将工作负载移到二级环境中。当执行故障转移时，存储从主环境中隔离，到虚拟机磁盘的 I/O 会完全停止，虚拟机使用复制的数据在次要环境中重新启动。

您应该预期因为故障转移而造成的数据丢失。丢失的程度取决于您是否使用 Metro-DR，它使用同步复制，或 Regional-DR （使用异步复制）。因为 Regional-DR 使用基于快照的复制间隔，所以数据丢失的窗口与复制间隔长度成比例。当虚拟机重启时，操作系统可能会执行崩溃恢复。

14.3.4. Red Hat OpenShift Data Foundation 的 Metro-DR

OpenShift Virtualization 支持 OpenShift Data Foundation 的 Metro-DR 解决方案，它在主站点和次站点上安装的受管 OpenShift Virtualization 集群之间提供双向同步数据复制。此解决方案结合了 Red Hat Advanced Cluster Management (RHACM)、Red Hat Ceph Storage 和 OpenShift Data Foundation 组件。

在站点灾难期间使用此解决方案，将应用程序从主站点故障转移到次站点，并在恢复灾难站点后将应用程序重新重新定位到主站点。

这个同步解决方案仅适用于具有 10 毫秒的延迟或更短的 metropolitan distance 数据中心。

有关在 OpenShift Virtualization 中使用 OpenShift Data Foundation 的 Metro-DR 解决方案的更多信息，请参阅 红帽知识库 或 IBM 的 OpenShift Data Foundation Metro-DR 文档。

其他资源

- 为 OpenShift Workloads 配置 OpenShift Data Foundation 灾难恢复

其他资源

- Red Hat Advanced Cluster Management for Kubernetes 2.10