OpenShift Container Platform 4.20

Virtualization

OpenShift Virtualization installation, usage, and release notes

Last Updated: 2025-11-06

OpenShift Container Platform 4.20 Virtualization

OpenShift Virtualization installation, usage, and release notes

Legal Notice

Abstract

This document provides information about how to use OpenShift Virtualization in OpenShift Container Platform.

Table of Contents

CHAPTER 1. ABOUT

1.1. ABOUT OPENSHIFT VIRTUALIZATION

Learn about OpenShift Virtualization's capabilities and support scope.

1.1.1. What you can do with OpenShift Virtualization

OpenShift Virtualization provides the scalable, enterprise-grade virtualization functionality in Red Hat OpenShift. You can use it to manage virtual machines (VMs) exclusively or alongside container workloads.

NOTE

If you have a Red Hat OpenShift Virtualization Engine subscription, you can run unlimited VMs on subscribed hosts, but you cannot run application instances in containers. For more information, see the subscription guide section about Red Hat OpenShift Virtualization Engine and related products.

OpenShift Virtualization adds new objects into your OpenShift Container Platform cluster by using Kubernetes custom resources to enable virtualization tasks. These tasks include:

- Creating and managing Linux and Windows VMs
- Running pod and VM workloads alongside each other in a cluster
- Connecting to VMs through a variety of consoles and CLI tools
- Importing and cloning existing VMs
- Managing network interface controllers and storage disks attached to VMs
- Live migrating VMs between nodes

You can manage your cluster and virtualization resources by using the **Virtualization** perspective of the OpenShift Container Platform web console, and by using the OpenShift CLI (**oc**).

OpenShift Virtualization is designed and tested to work well with Red Hat OpenShift Data Foundation features.

IMPORTANT

When you deploy OpenShift Virtualization with OpenShift Data Foundation, you must create a dedicated storage class for Windows virtual machine disks. See Optimizing ODF PersistentVolumes for Windows VMs for details.

You can use OpenShift Virtualization with OVN-Kubernetes or one of the other certified network plugins listed in Certified OpenShift CNI Plug-ins.

You can check your OpenShift Virtualization cluster for compliance issues by installing the Compliance Operator and running a scan with the **ocp4-moderate** and **ocp4-moderate-node** profiles. The Compliance Operator uses OpenSCAP, a NIST-certified tool, to scan and enforce security policies.

For information about partnering with Independent Software Vendors (ISVs) and Services partners for specialized storage, networking, backup, and additional functionality, see the Red Hat Ecosystem Catalog.

1.1.2. Comparing OpenShift Virtualization to VMware vSphere

If you are familiar with VMware vSphere, the following table lists OpenShift Virtualization components that you can use to accomplish similar tasks. However, because OpenShift Virtualization is conceptually different from vSphere, and much of its functionality comes from the underlying OpenShift Container Platform, OpenShift Virtualization does not have direct alternatives for all vSphere concepts or components.

Table 1.1. Mapping of vSphere concepts to their closest OpenShift Virtualization counterparts

vSphere concept	OpenShift Virtualization	Explanation
Datastore	Persistent volume (PV) + Persistent volume claim (PVC)	Stores VM disks. A PV represents existing storage and is attached to a VM through a PVC. When created with the **ReadWriteMany** (RWX) access mode, PVCs can be mounted by multiple VMs simultaneously.
Dynamic Resource Scheduling (DRS)	Pod eviction policy + Descheduler	Provides active resource balancing. A combination of pod eviction policies and a descheduler allows VMs to be live migrated to more appropriate nodes to keep node resource utilization manageable.
NSX	Multus + OVN-Kubernetes + Third-party container network interface (CNI) plug-ins	Provides an overlay network configuration. There is no direct equivalent for NSX in OpenShift Virtualization, but you can use the OVN-Kubernetes network provider or install certified third-party CNI plug-ins.
Storage Policy Based Management (SPBM)	Storage class	Provides policy-based storage selection. Storage classes represent various storage types and describe storage capabilities, such as quality of service, backup policy, reclaim policy, and whether volume expansion is allowed. A PVC can request a specific storage class to satisfy application requirements.
vCenter vRealize Operations	OpenShift Metrics and Monitoring	Provides host and VM metrics. You can view metrics and monitor the overall health of the cluster and VMs by using the OpenShift Container Platform web console.

vSphere concept	OpenShift Virtualization	Explanation
vMotion	Live migration	Moves a running VM to another node without interruption. For live migration to be available, the PVC attached to the VM must have the **ReadWriteMany** (RWX) access mode.
vSwitch DvSwitch	NMState Operator + Multus	Provides a physical network configuration. You can use the NMState Operator to apply state-driven network configuration and manage various network interface types, including Linux bridges and network bonds. With Multus, you can attach multiple network interfaces and connect VMs to external networks.

1.1.3. Supported cluster versions for OpenShift Virtualization

The latest stable release of OpenShift Virtualization 4.20 is 4.20.0.

OpenShift Virtualization 4.20 is supported for use on OpenShift Container Platform 4.20 clusters. To use the latest z-stream release of OpenShift Virtualization, you must first upgrade to the latest version of OpenShift Container Platform.

1.1.4. About volume and access modes for virtual machine disks

If you use the storage API with known storage providers, the volume and access modes are selected automatically. However, if you use a storage class that does not have a storage profile, you must configure the volume and access mode.

For a list of known storage providers for OpenShift Virtualization, see the Red Hat Ecosystem Catalog .

For best results, use the **ReadWriteMany** (RWX) access mode and the **Block** volume mode. This is important for the following reasons:

- **ReadWriteMany** (RWX) access mode is required for live migration.
- The **Block** volume mode performs significantly better than the **Filesystem** volume mode. This is because the **Filesystem** volume mode uses more storage layers, including a file system layer and a disk image file. These layers are not necessary for VM disk storage.
  For example, if you use Red Hat OpenShift Data Foundation, Ceph RBD volumes are preferable to CephFS volumes.

IMPORTANT

You cannot live migrate virtual machines with the following configurations:

- Storage volume with **ReadWriteOnce** (RWO) access mode
- Passthrough features such as GPUs

Set the **evictionStrategy** field to **None** for these virtual machines. The **None** strategy powers down VMs during node reboots.

1.1.5. Single-node OpenShift differences

You can install OpenShift Virtualization on single-node OpenShift.

However, you should be aware that Single-node OpenShift does not support the following features:

- High availability
- Pod disruption
- Live migration
- Virtual machines or templates that have an eviction strategy configured

1.1.6. Additional resources

- OpenShift Virtualization supported limits
- Glossary of common terms for OpenShift Container Platform storage
- About single-node OpenShift
- Assisted installer
- Pod disruption budgets
- About live migration
- Eviction strategies
- Tuning & Scaling Guide in the Red Hat Knowledgebase

1.2. SUPPORTED LIMITS

You can refer to tested object maximums when planning your OpenShift Container Platform environment for OpenShift Virtualization. However, approaching the maximum values can reduce performance and increase latency. Ensure that you plan for your specific use case and consider all factors that can impact cluster scaling.

For more information about cluster configuration and options that impact performance, see the OpenShift Virtualization - Tuning & Scaling Guide in the Red Hat Knowledgebase.

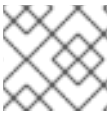
1.2.1. Tested maximums for OpenShift Virtualization

The following limits apply to a large-scale OpenShift Virtualization 4.x environment. They are based on a single cluster of the largest possible size. When you plan an environment, remember that multiple smaller clusters might be the best option for your use case.

1.2.1.1. Virtual machine maximums

The following maximums apply to virtual machines (VMs) running on OpenShift Virtualization. These values are subject to the limits specified in Virtualization limits for Red Hat Enterprise Linux with KVM .

Objective (per VM)	Tested limit	Theoretical limit
Virtual CPUs	216 vCPUs	255 vCPUs
Memory	6 TB	16 TB
Single disk size	20 TB	100 TB
Hot-pluggable disks	255 disks	N/A



NOTE

Each VM must have at least 512 MB of memory.

1.2.1.2. Host maximums

The following maximums apply to the OpenShift Container Platform hosts used for OpenShift Virtualization.

Objective (per host)	Tested limit	Theoretical limit
Logical CPU cores or threads	Same as Red Hat Enterprise Linux (RHEL)	N/A
RAM	Same as RHEL	N/A
Simultaneous live migrations	Defaults to 2 outbound migrations per node, and 5 concurrent migrations per cluster	Depends on NIC bandwidth
Live migration bandwidth	No default limit	Depends on NIC bandwidth

1.2.1.3. Cluster maximums

The following maximums apply to objects defined in OpenShift Virtualization.

Objective (per cluster)	Tested limit	Theoretical limit
Number of attached PVs per node	N/A	CSI storage provider dependent

Objective (per cluster)	Tested limit	Theoretical limit
Maximum PV size	N/A	CSI storage provider dependent
Hosts	500 hosts (100 or fewer recommended) [1]	Same as OpenShift Container Platform
Defined VMs	10,000 VMs [2]	Same as OpenShift Container Platform

1. If you use more than 100 nodes, consider using Red Hat Advanced Cluster Management (RHACM) to manage multiple clusters instead of scaling out a single control plane. Larger clusters add complexity, require longer updates, and depending on node size and total object density, they can increase control plane stress.
   Using multiple clusters can be beneficial in areas like per-cluster isolation and high availability.

2. The maximum number of VMs per node depends on the host hardware and resource capacity. It is also limited by the following parameters:

   - Settings that limit the number of pods that can be scheduled to a node. For example: **maxPods**.

   - The default number of KVM devices. For example: **devices.kubevirt.io/kvm: 1k**.

1.2.2. Additional resources

- OpenShift Virtualization - Tuning & Scaling Guide
- Planning your environment according to object maximums
- Managing the maximum number of pods per node
- Red Hat Advanced Cluster Management documentation

1.3. SECURITY POLICIES

Learn about OpenShift Virtualization security and authorization.

Key points

- OpenShift Virtualization adheres to the **restricted** Kubernetes pod security standards profile, which aims to enforce the current best practices for pod security.
- Virtual machine (VM) workloads run as unprivileged pods.
- Security context constraints (SCCs) are defined for the **kubevirt-controller** service account.
- TLS certificates for OpenShift Virtualization components are renewed and rotated automatically.

1.3.1. About workload security

By default, virtual machine (VM) workloads do not run with root privileges in OpenShift Virtualization, and there are no supported OpenShift Virtualization features that require root privileges.

For each VM, a **virt-launcher** pod runs an instance of **libvirt** in *session mode* to manage the VM process. In session mode, the **libvirt** daemon runs as a non-root user account and only permits connections from clients that are running under the same user identifier (UID). Therefore, VMs run as unprivileged pods, adhering to the security principle of least privilege.

1.3.2. TLS certificates

TLS certificates for OpenShift Virtualization components are renewed and rotated automatically. You are not required to refresh them manually.

Automatic renewal schedules

TLS certificates are automatically deleted and replaced according to the following schedule:

- KubeVirt certificates are renewed daily.
- Containerized Data Importer controller (CDI) certificates are renewed every 15 days.
- MAC pool certificates are renewed every year.

Automatic TLS certificate rotation does not disrupt any operations. For example, the following operations continue to function without any disruption:

- Migrations
- Image uploads
- VNC and console connections

1.3.3. Authorization

OpenShift Virtualization uses role-based access control (RBAC) to define permissions for human users and service accounts. The permissions defined for service accounts control the actions that OpenShift Virtualization components can perform.

You can also use RBAC roles to manage user access to virtualization features. For example, an administrator can create an RBAC role that provides the permissions required to launch a virtual machine. The administrator can then restrict access by binding the role to specific users.

1.3.3.1. Default cluster roles for OpenShift Virtualization

By using cluster role aggregation, OpenShift Virtualization extends the default OpenShift Container Platform cluster roles to include permissions for accessing virtualization objects. Roles unique to OpenShift Virtualization are not aggregated with OpenShift Container Platform roles.

Table 1.2. OpenShift Virtualization cluster roles

Default cluster role	OpenShift Virtualization cluster role	OpenShift Virtualization cluster role description

Default cluster role	OpenShift Virtualization cluster role	OpenShift Virtualization cluster role description
view	**kubevirt.io:view**	A user that can view all OpenShift Virtualization resources in the cluster but cannot create, delete, modify, or access them. For example, the user can see that a virtual machine (VM) is running but cannot shut it down or gain access to its console.
edit	**kubevirt.io:edit**	A user that can modify all OpenShift Virtualization resources in the cluster. For example, the user can create VMs, access VM consoles, and delete VMs.
admin	**kubevirt.io:admin**	A user that has full permissions to all OpenShift Virtualization resources, including the ability to delete collections of resources. The user can also view and modify the OpenShift Virtualization runtime configuration, which is located in the **HyperConverged** custom resource in the **openshift-cnv** namespace.
N/A	**kubevirt.io:migrate**	A user that can create, delete, and update VM live migration requests, which are represented by namespaced **VirtualMachineInstanceMigration** (VMIM) objects. This role is specific to OpenShift Virtualization.

1.3.3.2. RBAC roles for storage features in OpenShift Virtualization

The following permissions are granted to the Containerized Data Importer (CDI), including the **cdi-operator** and **cdi-controller** service accounts.

1.3.3.2.1. Cluster-wide RBAC roles

Table 1.3. Aggregated cluster roles for the cdi.kubevirt.io API group

CDI cluster role	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 cluster role	Resources	Verbs
cdi.kubevirt.io:config-reader	**cdiconfigs**, **storageprofiles**	**get**, **list**, **watch**

Table 1.4. Cluster-wide roles for the **cdi-operator** service account

API group	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** Allow list: **cdi-api-dataimportcron-validate, cdi-api-populator-validate, cdi-api-datavolume-validate, cdi-api-validate, objecttransfer-api-validate**	**get**, **update**, **delete**

API group	Resources	Verbs
admissionregistration.k8s.io	**mutatingwebhookconfigurations** Allow list: **cdi-api-datavolume-mutate**	**get**, **update**, **delete**
apiregistration.k8s.io	**apiservices**	**get**, **list**, **watch**, **create**, **update**, **delete**

Table 1.5. Cluster-wide roles for the **cdi-controller** service account

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

API group	Resources	Verbs
scheduling.k8s.io	**priorityclasses**	**get**, **list**, **watch**
image.openshift.io	**imagestreams**	**get**, **list**, **watch**
"" (core)	**secrets**	**create**
kubevirt.io	**virtualmachines/finalizers**	**update**

1.3.3.2.2. Namespaced RBAC roles

Table 1.6. Namespaced roles for the **cdi-operator** service account

API group	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**

Table 1.7. Namespaced roles for the **cdi-controller** service account

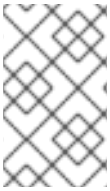
API group	Resources	Verbs
"" (core)	**configmaps**	**get**, **list**, **watch**, **create**, **update**, **delete**
"" (core)	**secrets**	**get**, **list**, **watch**

API group	Resources	Verbs
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**

1.3.3.3. Additional SCCs and permissions for the kubevirt-controller service account

Security context constraints (SCCs) control permissions for pods. These permissions include actions that a pod, a collection of containers, can perform and what resources it can access. You can use SCCs to define a set of conditions that a pod must run with to be accepted into the system.

The **virt-controller** is a cluster controller that creates the **virt-launcher** pods for virtual machines in the cluster.



NOTE

By default, **virt-launcher** pods run with the **default** service account in the namespace. If your compliance controls require a unique service account, assign one to the VM. The setting applies to the **VirtualMachineInstance** object and the **virt-launcher** pod.

The **kubevirt-controller** service account is granted additional SCCs and Linux capabilities so that it can create **virt-launcher** pods with the appropriate permissions. These extended permissions allow virtual machines to use OpenShift Virtualization features that are beyond the scope of typical pods.

The **kubevirt-controller** service account is granted the following SCCs:

- **scc.AllowHostDirVolumePlugin = true**
  This allows virtual machines to use the hostpath volume plugin.
- **scc.AllowPrivilegedContainer = false**
  This ensures the **virt-launcher** pod is not run as a privileged container.
- **scc.AllowedCapabilities = []corev1.Capability{"SYS_NICE", "NET_BIND_SERVICE"}**
  - **SYS_NICE** allows setting the CPU affinity.
  - **NET_BIND_SERVICE** allows DHCP and Slirp operations.

Viewing the SCC and RBAC definitions for the kubevirt-controller

You can view the **SecurityContextConstraints** definition for the **kubevirt-controller** by using the **oc** tool:

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

You can view the RBAC definition for the **kubevirt-controller** clusterrole by using the **oc** tool:

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

1.3.4. Additional resources

- Managing security context constraints
- Using RBAC to define and apply permissions
- Creating a cluster role
- Cluster role binding commands
- Enabling user permissions to clone data volumes across namespaces

1.4. OPENSHIFT VIRTUALIZATION ARCHITECTURE

The Operator Lifecycle Manager (OLM) deploys operator pods for each component of OpenShift Virtualization:

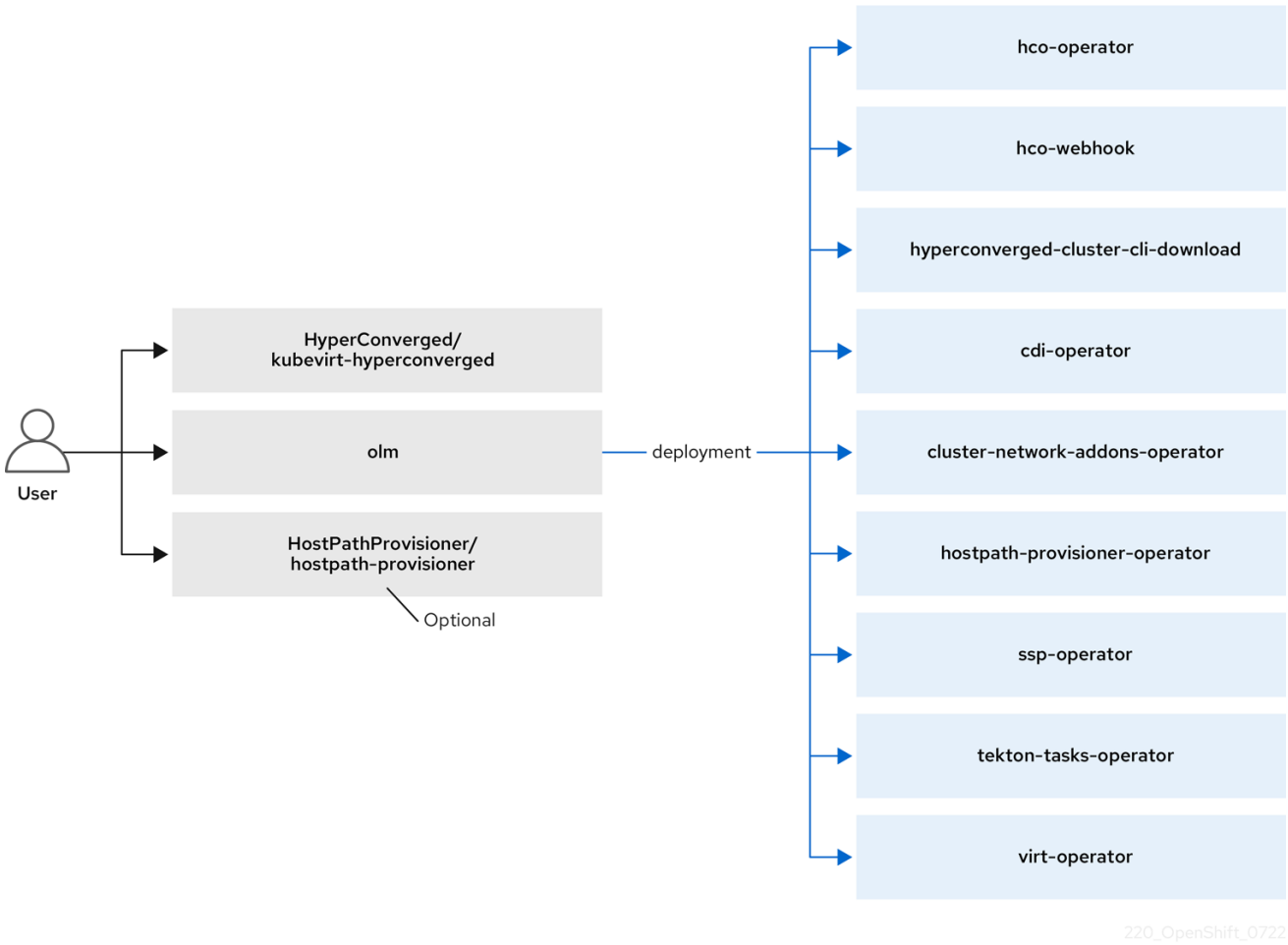
- Compute: **virt-operator**
- Storage: **cdi-operator**
- Network: **cluster-network-addons-operator**
- Scaling: **ssp-operator**

OLM also deploys the **hyperconverged-cluster-operator** pod, which is responsible for the deployment, configuration, and life cycle of other components, and several helper pods: **hco-webhook**, and **hyperconverged-cluster-cli-download**.

After all operator pods are successfully deployed, you should create the **HyperConverged** custom resource (CR). The configurations set in the **HyperConverged** CR serve as the single source of truth and the entrypoint for OpenShift Virtualization, and guide the behavior of the CRs.

The **HyperConverged** CR creates corresponding CRs for the operators of all other components within its reconciliation loop. Each operator then creates resources such as daemon sets, config maps, and additional components for the OpenShift Virtualization control plane. For example, when the HyperConverged Operator (HCO) creates the **KubeVirt** CR, the OpenShift Virtualization Operator reconciles it and creates additional resources such as **virt-controller**, **virt-handler**, and **virt-api**.

The OLM deploys the Hostpath Provisioner (HPP) Operator, but it is not functional until you create a **hostpath-provisioner** CR.



- Virtctl client commands

1.4.1. About the HyperConverged Operator (HCO)

The HCO, **hco-operator**, provides a single entry point for deploying and managing OpenShift Virtualization and several helper operators with opinionated defaults. It also creates custom resources (CRs) for those operators.

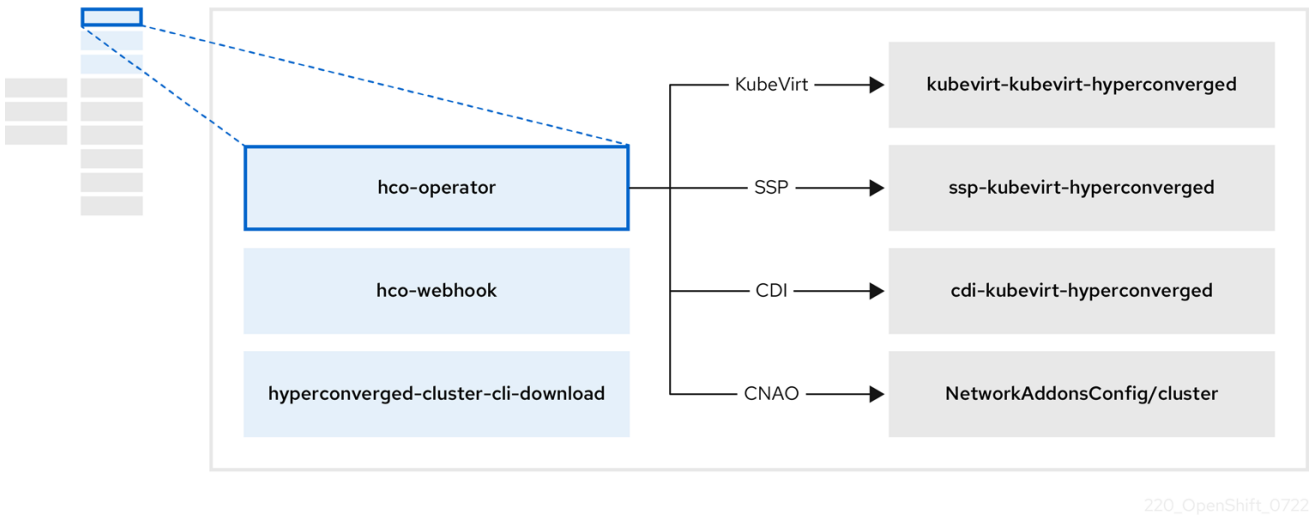


Table 1.8. HyperConverged Operator components

Component	Description
deployment/hco-webhook	Validates the **HyperConverged** custom resource contents.
deployment/hyperconverged-cluster-cli-download	Provides the **virtctl** tool binaries to the cluster so that you can download them directly from the cluster.
KubeVirt/kubevirt-kubevirt-hyperconverged	Contains all operators, CRs, and objects needed by OpenShift Virtualization.
SSP/ssp-kubevirt-hyperconverged	A Scheduling, Scale, and Performance (SSP) CR. This is automatically created by the HCO.
CDI/cdi-kubevirt-hyperconverged	A Containerized Data Importer (CDI) CR. This is automatically created by the HCO.
NetworkAddonsConfig/cluster	A CR that instructs and is managed by the **cluster-network-addons-operator**.

1.4.2. About the Containerized Data Importer (CDI) Operator

The CDI Operator, **cdi-operator**, manages CDI and its related resources, which imports a virtual machine (VM) image into a persistent volume claim (PVC) by using a data volume.

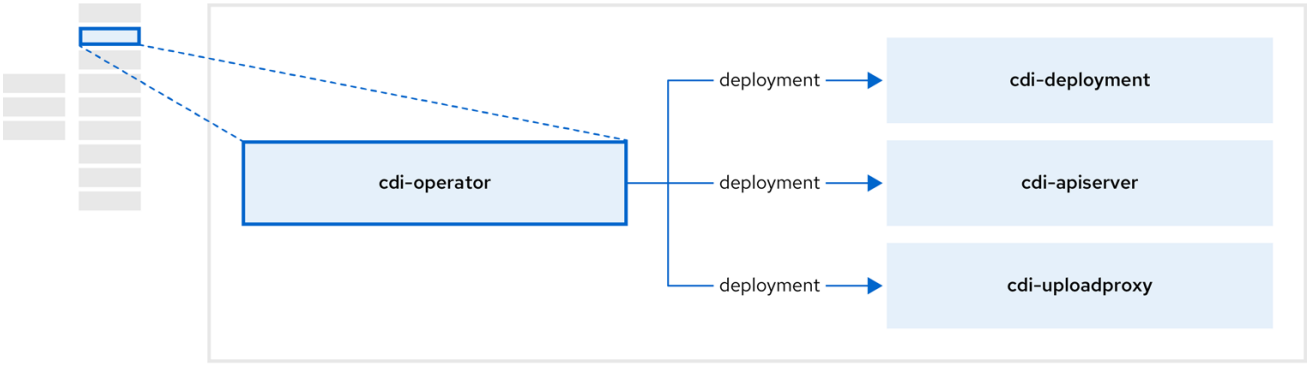


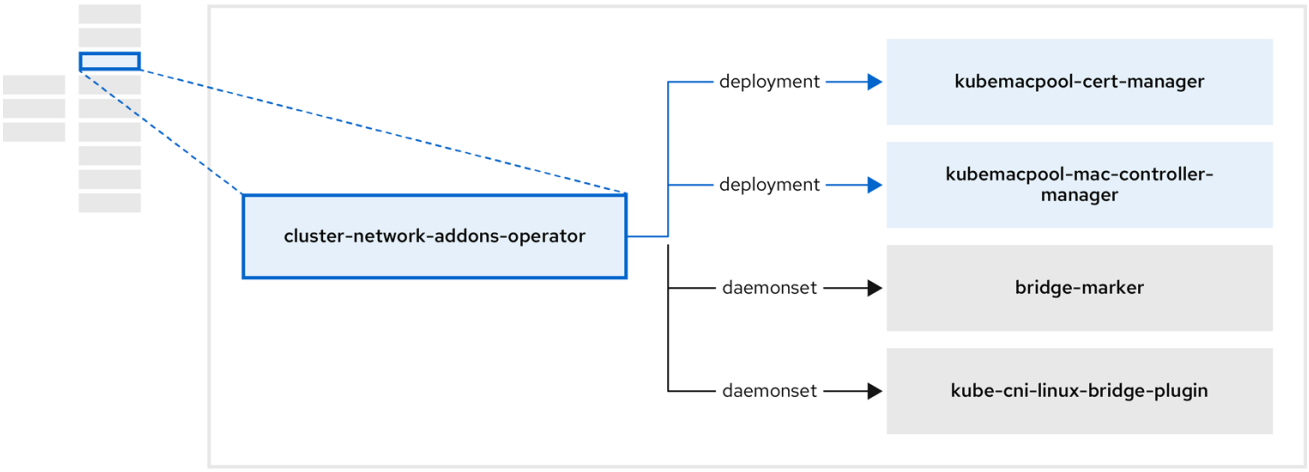
Table 1.9. CDI Operator components

Component	Description
deployment/cdi-apiserver	Manages the authorization to upload VM disks into PVCs by issuing secure upload tokens.

Component	Description
deployment/cdi-uploadproxy	Directs external disk upload traffic to the appropriate upload server pod so that it can be written to the correct PVC. Requires a valid upload token.
pod/cdi-importer	Helper pod that imports a virtual machine image into a PVC when creating a data volume.

1.4.3. About the Cluster Network Addons Operator

The Cluster Network Addons Operator, **cluster-network-addons-operator**, deploys networking components on a cluster and manages the related resources for extended network functionality.



220_OpenShift_0722

Table 1.10. Cluster Network Addons Operator components

Component	Description
deployment/kubemacpool-cert-manager	Manages TLS certificates of Kubemacpool's webhooks.
deployment/kubemacpool-mac-controller-manager	Provides a MAC address pooling service for virtual machine (VM) network interface cards (NICs).
daemonset/bridge-marker	Marks network bridges available on nodes as node resources.
daemonset/kube-cni-linux-bridge-plugin	Installs Container Network Interface (CNI) plugins on cluster nodes, enabling the attachment of VMs to Linux bridges through network attachment definitions.

1.4.4. About the Hostpath Provisioner (HPP) Operator

The HPP Operator, **hostpath-provisioner-operator**, deploys and manages the multi-node HPP and related resources.

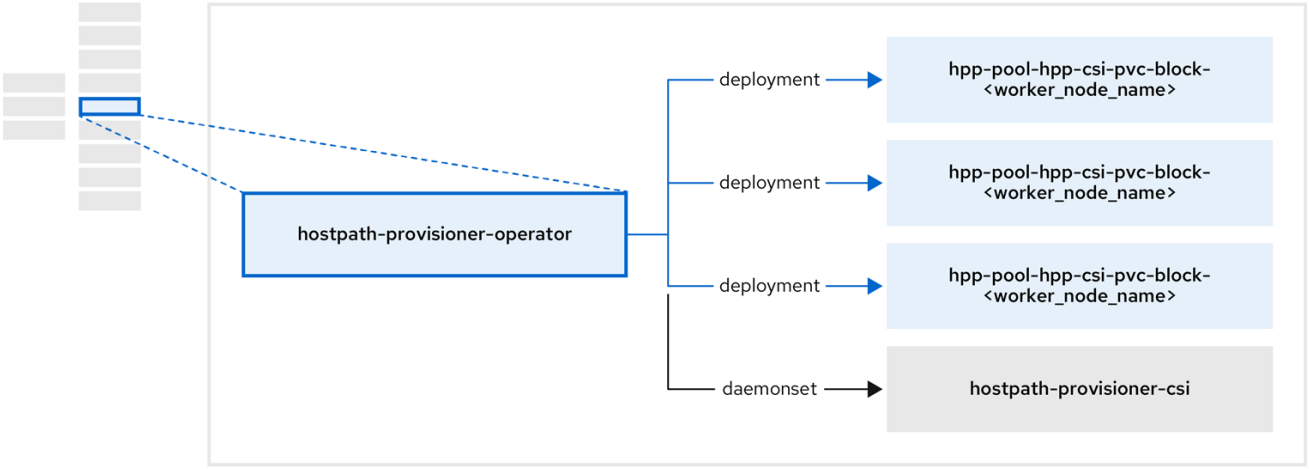


Table 1.11. HPP Operator components

Component	Description
deployment/hpp-pool-hpp-csi-pvc-block-<worker_node_name>	Provides a worker for each node where the HPP is designated to run. The pods mount the specified backing storage on the node.
daemonset/hostpath-provisioner-csi	Implements the Container Storage Interface (CSI) driver interface of the HPP.
daemonset/hostpath-provisioner	Implements the legacy driver interface of the HPP.

1.4.5. About the Scheduling, Scale, and Performance (SSP) Operator

The SSP Operator, **ssp-operator**, deploys the common templates, the related default boot sources, the pipeline tasks, and the template validator.

1.4.6. About the OpenShift Virtualization Operator

The OpenShift Virtualization Operator, **virt-operator**, deploys, upgrades, and manages OpenShift Virtualization without disrupting current virtual machine (VM) workloads. In addition, the OpenShift Virtualization Operator deploys the common instance types and common preferences.

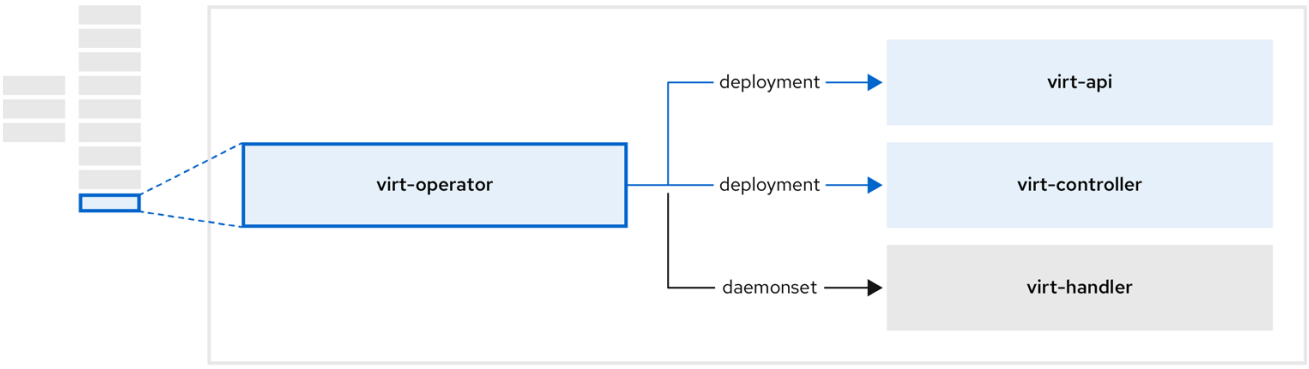


Table 1.12. virt-operator components

Component	Description
deployment/virt-api	HTTP API server that serves as the entry point for all virtualization-related flows.
deployment/virt-controller	Observes the creation of a new VM instance object and creates a corresponding pod. When the pod is scheduled on a node, **virt-controller** updates the VM with the node name.
daemonset/virt-handler	Monitors any changes to a VM and instructs **virt-launcher** to perform the required operations. This component is node-specific.
pod/virt-launcher	Contains the VM that was created by the user as implemented by **libvirt** and **qemu**.

CHAPTER 2. RELEASE NOTES

2.1. OPENSHIFT VIRTUALIZATION RELEASE NOTES

2.1.1. Providing documentation feedback

To report an error or to improve our documentation, log in to your Red Hat Jira account and submit a Jira issue.

2.1.2. About Red Hat OpenShift Virtualization

With Red Hat OpenShift Virtualization, you can bring traditional virtual machines (VMs) into OpenShift Container Platform and run them alongside containers. In OpenShift Virtualization, VMs are native Kubernetes objects that you can manage by using the OpenShift Container Platform web console or the command line.

OpenShift Virtualization is represented by the  icon.

You can use OpenShift Virtualization the OVN-Kubernetes Container Network Interface (CNI) network provider.

Learn more about what you can do with OpenShift Virtualization.

Learn more about OpenShift Virtualization architecture and deployments.

Prepare your cluster for OpenShift Virtualization.

2.1.2.1. Supported cluster versions for OpenShift Virtualization

The latest stable release of OpenShift Virtualization 4.20 is 4.20.0.

OpenShift Virtualization 4.20 is supported for use on OpenShift Container Platform 4.20 clusters. To use the latest z-stream release of OpenShift Virtualization, you must first upgrade to the latest version of OpenShift Container Platform.

2.1.2.2. Supported guest operating systems

To view the supported guest operating systems for OpenShift Virtualization, see Certified Guest Operating Systems in Red Hat OpenStack Platform, Red Hat Virtualization, OpenShift Virtualization and Red Hat Enterprise Linux with KVM.

2.1.2.3. Microsoft Windows SVVP certification

OpenShift Virtualization is certified in Microsoft's Windows Server Virtualization Validation Program (SVVP) to run Windows Server workloads.

The SVVP certification applies to:

- Red Hat Enterprise Linux CoreOS workers. In the Microsoft SVVP Catalog, they are named *Red Hat OpenShift Container Platform 4.20*.
- Intel and AMD CPUs.

2.1.3. Quick starts

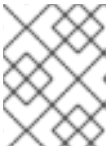
Quick start tours are available for several OpenShift Virtualization features. To view the tours, click the **Help** icon **?** in the menu bar on the header of the OpenShift Container Platform web console and then select **Quick Starts**. You can filter the available tours by entering the keyword **virtualization** in the **Filter** field.

2.1.4. New and changed features

This release adds new features and enhancements related to the following components and concepts:

2.1.4.1. Installation and update

- You can now directly update OpenShift Virtualization to a later z-stream (x.y.z) release without applying each intermediate z-stream version.



NOTE

Ensure that you update to the latest z-stream release of your current minor (x.y) version before updating to the next minor version.

- Installing OpenShift Virtualization on Oracle Cloud Infrastructure (OCI) is now generally available. For more information, see OpenShift Virtualization and Oracle Cloud Infrastructure known issues and limitations in the Red Hat Knowledgebase, and Installing OpenShift Virtualization on OCI on GitHub.

- Using OpenShift Virtualization on a bare-metal cluster installed on an ARM64 (AARCH64) system is now generally available. For more information, see ARM64 compatibility.

2.1.4.2. Virtualization

- The descheduler profile **DevKubeVirtRelieveAndMigrate** has been renamed to **KubeVirtRelieveAndMigrate** and is now generally available. The updated profile improves VM eviction stability during live migrations by enabling background evictions and reducing oscillatory behavior. For more information, see Configuring descheduler evictions for virtual machines.

- vNUMA topology for VMs is now generally available (GA). By enabling this feature, you opt in to an improved NUMA configuration for VMs, with better performance and optimal resource allocation. For more information, see Working with NUMA topology for virtual machines.

- You can now use the **kube_application_aware_resourcequota** and **kube_application_aware_resourcequota_creation_timestamp** metrics to query the current usage and creation times of the Application-Aware Quota (AAQ) Operator resources. For more information, see AAQ Operator metrics.

2.1.4.3. Networking

- You can now hot plug and hot unplug a secondary network interface to a VM without manually triggering live migration. You do not need permission to create and list **VirtualMachineInstanceMigration** objects. For more information, see Hot plugging secondary network interfaces.

- Managing the link state of a virtual machine interface is now generally available. In previous releases this was a Technology Preview feature.

- You can now use the Border Gateway Protocol (BGP) to configure dynamic ingress and egress routing for VMs that are connected to primary user-defined networks. Importing routes from provider networks into OVN-Kubernetes eliminates the need to manually configure routes on hosts. With dynamic egress, you can export VM IP addresses to provider networks, making the VMs directly reachable from outside the cluster. For more information, see Advertise cluster network routes with Border Gateway Protocol.

2.1.4.4. Web console

- In the OpenShift Container Platform web console, the **Migrations** tab of the **Virtualization** page now displays a progress bar for each migrating virtual machine.

- When performing live migration of a VM, you can now specify the particular node for the VM to migrate to.

- The procedure for hot plugging disks now includes an optional step for selecting a bus type. You can select the **virtio-blk** or the **virtio-scsi** bus type. The **virtio-blk** type is the default. For more information, see Hot plugging VM disks .

- The **InstanceTypes** tab on the **Create new VirtualMachine** page now includes options for selecting huge pages. These options appear in the **M** and **CX** series of instance types. They are accessible both through the **Select InstanceType** tiles and in the **Default InstanceType** menu of the **Add volume** dialog box.
  For more information about selecting huge pages for an instance type, see "Creating a VM from an instance type by using the web console".

- You can now easily identify if NUMA is enabled on your virtual machines. With this update, the **vNUMA** attribute is displayed in the VM details next to the **CPU | Memory** section.

2.1.4.5. Monitoring

- Added documentation for the **kubevirt_vmi_vcpu_delay_seconds_total** Prometheus metric. This metric reports the time a virtual CPU (vCPU) was queued by the host scheduler but was not running. The updated documentation helps users better understand vCPU queue delays in OpenShift Virtualization environments.

- The following alerts for the OpenShift Virtualization Operator are now included in the OpenShift Virtualization runbooks:

  - **HighNodeCPUFrequency**
  - **VirtualMachineStuckInUnhealthyState**
  - **VirtualMachineStuckOnNode**
  - **PersistentVolumeFillingUp**
  - **DeprecatedMachineType**
  - **HCOGoldenImageWithNoSupportedArchitecture**
  - **HCOGoldenImageWithNoArchitectureAnnotation**

- **HCOMultiArchGoldenImagesDisabled**

For a complete list of virtualization metrics, see the openshift/runbooks Git repository.

- Using the guest agent ping probe to determine if the QEMU guest agent is running on the VM is now generally available. Previously, this feature was provided as a Technology Preview.
- Using Microsoft Azure Boost with OpenShift Virtualization on Azure Red Hat OpenShift (ARO) is now generally available.

2.1.4.6. Notable technical changes

- Before this update, only the **virtio-scsi** bus type could be used for hot plugging disks. With this update, the **virtio-blk** bus type is supported as well.

2.1.5. Deprecated and removed features

2.1.5.1. Deprecated features

Deprecated features are included in the current release and supported. However, they will be removed in a future release and are not recommended for new deployments.

- The **OperatorConditionsUnhealthy** alert is deprecated. You can safely silence it.
- All hot plugged disks are persistent by default. The use of non-persistent hot plugged disks is deprecated. They will not be supported in future releases.

2.1.5.2. Removed features

Removed features are no longer supported in OpenShift Virtualization.

- With this release, support for the Data Plane Development Kit (DPDK) checkup has been removed. You can no longer run a predefined checkup to verify if your OpenShift Container Platform cluster node can run a VM with a DPDK workload with zero packet loss.

2.1.6. Technology Preview features

Some features in this release are currently in Technology Preview. These experimental features are not intended for production use. Note the following scope of support on the Red Hat Customer Portal for these features:

Technology Preview Features Support Scope

- You can use OpenShift Virtualization on Microsoft Azure Boost.
- Golden image support for heterogeneous clusters is now available.
- You can now use the Plug a Simple Socket Transport (passt) network binding plugin to connect a VM to a primary user-defined network (UDN). For more information, see Attaching a virtual machine to the primary user-defined network.

2.1.7. Bug fixes

- Restoring a snapshot of a VM after a storage migration no longer fails because of unreferenced

dataVolumeTemplate objects. The snapshot process now refreshes the data volume templates in the controller revision to match the **volumes** list, ensuring consistent data recovery. (CNV-61279)

- The migration controller in the **virt-handler** pod was redesigned to separate source, target, and VM responsibilities, ensure deterministic completion, and use a unified **VirtualMachineInstance** (VMI) cache. (CNV-48348)
- Virtual Trusted Platform Module (vTPM) persistence is now enabled by default in VM templates. BitLocker system checks in Windows VMs no longer pass with non-persistent vTPM devices. (CNV-36448)
- On s390x systems, VMs created from a template with the **Boot from CD** option now boot correctly. CD-ROM devices are attached as SCSI instead of SATA, which is not supported on s390x architecture. (CNV-61740)

2.1.8. Known issues

2.1.8.1. Networking

- When you update from OpenShift Container Platform 4.12 to a newer minor version, VMs that use the **cnv-bridge** Container Network Interface (CNI) fail to live migrate. (https://access.redhat.com/solutions/7069807)
  - As a workaround, change the **spec.config.type** field in your **NetworkAttachmentDefinition** manifest from **cnv-bridge** to **bridge** before performing the update.
- Red Hat OpenShift Service Mesh 3.1.1 and Istio versions 1.25 and later are incompatible with OpenShift Virtualization 4.20 because the annotation **traffic.sidecar.istio.io/kubevirtInterfaces** is deprecated. (OSSM-10883)
  - As a workaround, when installing Service Mesh for integration with OpenShift Virtualization, select version 3.0.4 and Istio 1.24.4 instead of the default versions that are displayed in the web console.

2.1.8.2. Nodes

- Uninstalling OpenShift Virtualization does not remove the **feature.node.kubevirt.io** node labels created by OpenShift Virtualization. You must remove the labels manually. (CNV-38543)

2.1.8.3. Storage

- Attempting a storage live migration from the OpenShift Container Platform web console might hang and fail to create a destination **PersistentVolumeClaim** (PVC). This issue occurs because the web console does not detect a label that marks source PVCs previously used for migration. When this label is present, the migration cannot proceed successfully. (CNV-70866)
  - As a workaround, use the Migration Toolkit for Containers (MTC) web console or create the **MigPlan** resource manually by using the CLI to perform the migration.

2.1.8.4. Virtualization

- Live migration fails if the VM name exceeds 47 characters. (CNV-61066)

- Live migration might fail if you are migrating a VM which has vNUMA enabled, and the **topologyManagerPolicy** setting in the KubeletConfig is configured with **none**. This is due to conflicting NUMA cells in the Topology Manager policy. (CNV-70330)
    - As a workaround, configure the **topologyManagerPolicy** setting in the KubeletConfig to use either the **best-effort** or **single-numa-node** policies.
- OpenShift Virtualization links a service account token in use by a pod to that specific pod. OpenShift Virtualization implements a service account volume by creating a disk image that contains a token. If you migrate a VM, then the service account volume becomes invalid. (CNV-33835)
    - As a workaround, use user accounts rather than service accounts because user account tokens are not bound to a specific pod.

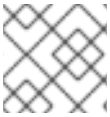
2.1.8.5. IBM Z and IBM LinuxONE

- VMs based on s390x architecture can only use the **IPL** boot mode. However, in the OpenShift Container Platform web console, the **Boot mode** list for s390x VMs incorrectly includes **BIOS**, **UEFI**, and **UEFI (secure)** boot modes. If you select one of these modes for an s390x-based VM, the operation fails. (CNV-56889)

CHAPTER 3. GETTING STARTED

3.1. GETTING STARTED WITH OPENSHIFT VIRTUALIZATION

You can explore the features and functionalities of OpenShift Virtualization by installing and configuring a basic environment.



NOTE

Cluster configuration procedures require **cluster-admin** privileges.

3.1.1. Tours and quick starts

You can start exploring OpenShift Virtualization by taking tours in the OpenShift Container Platform web console.

Getting started tour
This short guided tour introduces several key aspects of using OpenShift Virtualization. There are two ways to start the tour:

- On the **Welcome to OpenShift Virtualization** dialog, click **Start Tour**.
- Go to **Virtualization → Overview → Settings → User → Getting started resources** and click **Guided tour**.

Quick starts
Quick start tours are available for several OpenShift Virtualization features. To access quick starts, complete the following steps:

1. Click the **Help** icon **?** in the menu bar on the header of the OpenShift Container Platform web console.
2. Select **Quick Starts**.

You can filter the available tours by entering the keyword **virtual** in the **Filter** field.

3.1.2. Planning and installing OpenShift Virtualization

Plan and install OpenShift Virtualization on an OpenShift Container Platform cluster:

- Plan your bare metal cluster for OpenShift Virtualization .
- Prepare your cluster for OpenShift Virtualization .
- Install the OpenShift Virtualization Operator .
- Install the **virtctl** command-line interface (CLI) tool .

Planning and installation resources

- About storage volumes for virtual machine disks .
- Using a CSI-enabled storage provider .
- Configuring local storage for virtual machines .

- Installing the Kubernetes NMState Operator .
- Specifying nodes for virtual machines.
- **Virtctl** commands.

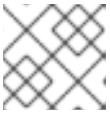
3.1.3. Creating and managing virtual machines

Create a virtual machine (VM):

- Create a VM from a Red Hat image .
  You can create a VM by using a Red Hat template or an instance type.
- You can create a VM by importing a custom image from a container registry or a web page , by uploading an image from your local machine , or by cloning a persistent volume claim (PVC) .

Connect a VM to a secondary network:

- Linux bridge network .
- Open Virtual Network (OVN)-Kubernetes secondary network .
- Single Root I/O Virtualization (SR-IOV) network.



NOTE

VMs are connected to the pod network by default.

Connect to a VM:

- Connect to the serial console or VNC console of a VM.
- Connect to a VM by using SSH .
- Connect to the desktop viewer for Windows VMs .

Manage a VM:

- Manage a VM by using the web console .
- Manage a VM by using the **virtctl** CLI tool.
- Export a VM .

3.1.4. Migrating to OpenShift Virtualization

To migrate virtual machines from an external provider such as VMware vSphere, Red Hat OpenStack Platform (RHOSP), Red Hat Virtualization, or another OpenShift Container Platform cluster, use the Migration Toolkit for Virtualization (MTV). You can also migrate Open Virtual Appliance (OVA) files created by VMware vSphere.

NOTE

Migration Toolkit for Virtualization is not part of OpenShift Virtualization and requires separate installation. For this reason, all links in this procedure lead outside of OpenShift Virtualization documentation.

Prerequisites

- The Migration Toolkit for Virtualization Operator is installed.

Procedure

- Migrate virtual machines from VMware vSphere.
- Migrate virtual machines from Red Hat OpenStack Platform (RHOSP).
- Migrate virtual machines from Red Hat Virtualization.
- Migrate virtual machines from OpenShift Virtualization.
- Migrate virtual machines from OVA files created by VMware vSphere.

3.1.5. Next steps

- Review postinstallation configuration options.
- Configure storage options and automatic boot source updates.
- Learn about monitoring and health checks.
- Learn about live migration.
- Back up and restore VMs by using the OpenShift API for Data Protection (OADP).
- Tune and scale your cluster.

3.2. USING THE CLI TOOLS

You can manage OpenShift Virtualization resources by using the **virtctl** command-line tool.

You can access and modify virtual machine (VM) disk images by using the **libguestfs** command-line tool. You deploy **libguestfs** by using the **virtctl libguestfs** command.

3.2.1. Installing virtctl

To install **virtctl** on Red Hat Enterprise Linux (RHEL) 9, Linux, Windows, and MacOS operating systems, you download and install the **virtctl** binary file.

To install **virtctl** on RHEL 8, you enable the OpenShift Virtualization repository and then install the **kubevirt-virtctl** package.

3.2.1.1. Installing the virtctl binary on RHEL 9, Linux, Windows, or macOS

You can download the **virtctl** binary for your operating system from the OpenShift Container Platform web console and then install it.

Procedure

1. Navigate to the **Virtualization → Overview** page in the web console.
2. Click the **Download virtctl** link to download the **virtctl** binary for your operating system.
3. Install **virtctl**:
   - For RHEL 9 and other Linux operating systems:
     a. Decompress the archive file:

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

     b. Run the following command to make the **virtctl** binary executable:

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

     c. Move the **virtctl** binary to a directory in your **PATH** environment variable. You can check your path by running the following command:

     ```
     $ echo $PATH
     ```

     d. Set the **KUBECONFIG** environment variable:

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

   - For Windows:
     a. Decompress the archive file.
     b. Navigate the extracted folder hierarchy and double-click the **virtctl** executable file to install the client.
     c. Move the **virtctl** binary to a directory in your **PATH** environment variable. You can check your path by running the following command:

     ```
     C:\> path
     ```

   - For macOS:
     a. Decompress the archive file.
     b. Move the **virtctl** binary to a directory in your **PATH** environment variable. You can check your path by running the following command:

     ```
     echo $PATH
     ```

3.2.1.2. Installing the virtctl RPM on RHEL 8

You can install the **virtctl** RPM package on Red Hat Enterprise Linux (RHEL) 8 by enabling the OpenShift Virtualization repository and installing the **kubevirt-virtctl** package.

Prerequisites

- Each host in your cluster must be registered with Red Hat Subscription Manager (RHSM) and have an active OpenShift Container Platform subscription.

Procedure

1. Enable the OpenShift Virtualization repository by using the **subscription-manager** CLI tool to run the following command:

```
# subscription-manager repos --enable cnv-4.20-for-rhel-8-x86_64-rpms
```

2. Install the **kubevirt-virtctl** package by running the following command:

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

3.2.2. virtctl commands

The **virtctl** client is a command-line utility for managing OpenShift Virtualization resources.

NOTE

The virtual machine (VM) commands also apply to virtual machine instances (VMIs) unless otherwise specified.

3.2.2.1. virtctl information commands

You can use the following **virtctl** information commands to view information about the **virtctl** client.

Table 3.1. Information commands

Command	Description
virtctl version	View the **virtctl** client and server versions.
virtctl help	View a list of **virtctl** commands.
virtctl <command> -h|--help	View a list of options for a specific command.
virtctl options	View a list of global command options for any **virtctl** command.

3.2.2.2. VM information commands

You can use **virtctl** to view information about virtual machines (VMs) and virtual machine instances (VMIs).

Table 3.2. VM information commands

Command	Description
virtctl fslist <vm_name>	View the file systems available on a guest machine.
virtctl guestosinfo <vm_name>	View information about the operating systems on a guest machine.
virtctl userlist <vm_name>	View the logged-in users on a guest machine.

3.2.2.3. VM manifest creation commands

You can use the following **virtctl create** commands to create manifests for virtual machines, instance types, and preferences.

Table 3.3. VM manifest creation commands

Command	Description
virtctl create vm	Create a **VirtualMachine** (VM) manifest.
virtctl create vm --name <vm_name>	Create a VM manifest, specifying a name for the VM.
virtctl create vm --user <user_name> --ssh-key|password-file= <value>	Create a VM manifest with a cloud-init configuration to create the selected user and either add an SSH public key from the supplied string, or a password from a file.
virtctl create vm --access-cred type:password,src:<secret>	Create a VM manifest with a user and password combination injected from the selected secret.
virtctl create vm --access-cred type:ssh,src:<secret>,user: <user_name>	Create a VM manifest with an SSH public key injected from the selected secret.
virtctl create vm --volume-sysprep src:<config_map>	Create a VM manifest, specifying a config map to use as the sysprep volume. The config map must contain a valid answer file named **unattend.xml** or **autounattend.xml**.
virtctl create vm --instancetype <instancetype_name>	Create a VM manifest that uses an existing cluster-wide instance type.

Command	Description
virtctl create vm --instancetype=virtualmachineinstancetype/<instancetype_name>	Create a VM manifest that uses an existing namespaced instance type.
virtctl create instancetype --cpu <cpu_value> --memory <memory_value> --name <instancetype_name>	Create a manifest for a cluster-wide instance type.
virtctl create instancetype --cpu <cpu_value> --memory <memory_value> --name <instancetype_name> --namespace <namespace_value>	Create a manifest for a namespaced instance type.
virtctl create preference --name <preference_name>	Create a manifest for a cluster-wide VM preference, specifying a name for the preference.
virtctl create preference --namespace <namespace_value>	Create a manifest for a namespaced VM preference.

3.2.2.4. VM management commands

You can use the following **virtctl** commands to manage and migrate virtual machines (VMs) and VM instances (VMIs).

Table 3.4. VM management commands

Command	Description
virtctl start <vm_name>	Start a VM.
virtctl start --paused <vm_name>	Start a VM in a paused state. This option enables you to interrupt the boot process from the VNC console.
virtctl stop <vm_name>	Stop a VM.
virtctl stop <vm_name> --grace-period 0 --force	Force stop a VM. This option might cause data inconsistency or data loss.
virtctl pause vm <vm_name>	Pause a VM. The machine state is kept in memory.
virtctl unpause vm <vm_name>	Unpause a VM.
virtctl migrate <vm_name>	Migrate a VM.

Command	Description
virtctl migrate-cancel <vm_name>	Cancel a VM migration.
virtctl restart <vm_name>	Restart a VM.

3.2.2.5. VM connection commands

You use can use the following **virtctl** commands to expose ports and connect to virtual machines (VMs) and VM instances (VMIs).

Table 3.5. VM connection commands

Command	Description
virtctl console <vm_name>	Connect to the serial console of a VM.
virtctl expose vm <vm_name> --name <service_name> --type <ClusterIP|NodePort|LoadBalancer> --port <port>	Create a service that forwards a designated port of a VM and expose the service on the specified port of the node. Example: **virtctl expose vm rhel9_vm --name rhel9-ssh --type NodePort --port 22**
virtctl scp -i <ssh_key> <file_name> <user_name>@vm/<vm_name>	Copy a file from your machine to a VM. This command uses the private key of an SSH key pair. The VM must be configured with the public key.
virtctl scp -i <ssh_key> <user_name@vm/<vm_name>:<file_name> .	Copy a file from a VM to your machine. This command uses the private key of an SSH key pair. The VM must be configured with the public key.
virtctl ssh -i <ssh_key> <user_name>@vm/<vm_name>	Open an SSH connection with a VM. This command uses the private key of an SSH key pair. The VM must be configured with the public key.
virtctl vnc <vm_name>	Connect to the VNC console of a VM. You must have **virt-viewer** installed.
virtctl vnc --proxy-only=true <vm_name>	Display the port number and connect manually to a VM by using any viewer through the VNC connection.
virtctl vnc --port=<port-number> <vm_name>	Specify a port number to run the proxy on the specified port, if that port is available. If a port number is not specified, the proxy runs on a random port.

3.2.2.6. VM export commands

Use **virtctl vmexport** commands to create, download, or delete a volume exported from a VM, VM snapshot, or persistent volume claim (PVC). Certain manifests also contain a header secret, which grants access to the endpoint to import a disk image in a format that OpenShift Virtualization can use.

Table 3.6. VM export commands

Command	Description
virtctl vmexport create <vmexport_name> --vm|snapshot|pvc= <object_name>	Create a **VirtualMachineExport** custom resource (CR) to export a volume from a VM, VM snapshot, or PVC. • **--vm**: Exports the PVCs of a VM. • **--snapshot**: Exports the PVCs contained in a **VirtualMachineSnapshot** CR. • **--pvc**: Exports a PVC. • Optional: **--ttl=1h** specifies the time to live. The default duration is 2 hours.
virtctl vmexport delete <vmexport_name>	Delete a **VirtualMachineExport** CR manually.
virtctl vmexport download <vmexport_name> --output= <output_file> --volume= <volume_name>	Download the volume defined in a **VirtualMachineExport** CR. • **--output** specifies the file format. Example: **disk.img.gz**. • **--volume** specifies the volume to download. This flag is optional if only one volume is available. Optional: • **--keep-vme** retains the **VirtualMachineExport** CR after download. The default behavior is to delete the **VirtualMachineExport** CR after download. • **--insecure** enables an insecure HTTP connection.
virtctl vmexport download <vmexport_name> --vm|snapshot|pvc= <object_name> --output= <output_file> --volume= <volume_name>	Create a **VirtualMachineExport** CR and then download the volume defined in the CR.
virtctl vmexport download export --manifest	Retrieve the manifest for an existing export. The manifest does not include the header secret.

Command	Description
virtctl vmexport download export --manifest --vm=example	Create a VM export for a VM example, and retrieve the manifest. The manifest does not include the header secret.
virtctl vmexport download export --manifest --snap=example	Create a VM export for a VM snapshot example, and retrieve the manifest. The manifest does not include the header secret.
virtctl vmexport download export --manifest --include-secret	Retrieve the manifest for an existing export. The manifest includes the header secret.
virtctl vmexport download export --manifest --manifest-output-format=json	Retrieve the manifest for an existing export in json format. The manifest does not include the header secret.
virtctl vmexport download export --manifest --include-secret --output=manifest.yaml	Retrieve the manifest for an existing export. The manifest includes the header secret and writes it to the file specified.

3.2.2.7. Hot plug and hot unplug commands

You can use the following **virtctl** commands to add or remove resources from running virtual machines (VMs) and VM instances (VMIs).

Table 3.7. Hot plug and hot unplug commands

Command	Description
virtctl addvolume <vm_name> --volume-name=<datavolume_or_PVC> [--persist] [--serial=<label>]	Hot plug a data volume or persistent volume claim (PVC). Optional: • **--persist** mounts the virtual disk permanently on a VM.**This flag does not apply to VMIs.** • **--serial=<label>** adds a label to the VM. If you do not specify a label, the default label is the data volume or PVC name.
virtctl removevolume <vm_name> --volume-name=<virtual_disk>	Hot unplug a virtual disk.

3.2.2.8. Image upload commands

You can use the following **virtctl image-upload** commands to upload a VM image to a data volume.

Table 3.8. Image upload commands

Command	Description
virtctl image-upload dv <datavolume_name> --image-path= </path/to/image> --no-create	Upload a VM image to a data volume that already exists.
virtctl image-upload dv <datavolume_name> --size= <datavolume_size> --image-path=</path/to/image>	Upload a VM image to a new data volume of a specified requested size.
virtctl image-upload dv <datavolume_name> --datasource --size= <datavolume_size> --image-path=</path/to/image>	Upload a VM image to a new data volume and create an associated **DataSource** object for it.

3.2.3. Deploying libguestfs by using virtctl

You can use the **virtctl guestfs** command to deploy an interactive container with **libguestfs-tools** and a persistent volume claim (PVC) attached to it.

Procedure

- To deploy a container with **libguestfs-tools**, mount the PVC, and attach a shell to it, run the following command:

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

1 The PVC name is a required argument. If you do not include it, an error message appears.

3.2.3.1. Libguestfs and virtctl guestfs commands

Libguestfs tools help you access and modify virtual machine (VM) disk images. You can use **libguestfs** tools to view and edit files in a guest, clone and build virtual machines, and format and resize disks.

You can also use the **virtctl guestfs** command and its sub-commands to modify, inspect, and debug VM disks on a PVC. To see a complete list of possible sub-commands, enter **virt-** on the command line and press the Tab key. For example:

Command	Description
virt-edit -a /dev/vda /etc/motd	Edit a file interactively in your terminal.

Command	Description
virt-customize -a /dev/vda --ssh-inject root:string:<public key example>	Inject an ssh key into the guest and create a login.
virt-df -a /dev/vda -h	See how much disk space is used by a VM.
virt-customize -a /dev/vda --run-command 'rpm -qa > /rpm-list'	See the full list of all RPMs installed on a guest by creating an output file containing the full list.
virt-cat -a /dev/vda /rpm-list	Display the output file list of all RPMs created using the **virt-customize -a /dev/vda --run-command 'rpm -qa > /rpm-list'** command in your terminal.
virt-sysprep -a /dev/vda	Seal a virtual machine disk image to be used as a template.

By default, **virtctl guestfs** creates a session with everything needed to manage a VM disk. However, the command also supports several flag options if you want to customize the behavior:

Flag Option	Description
--h or **--help**	Provides help for **guestfs**.
-n <namespace> option with a **<pvc_name>** argument	To use a PVC from a specific namespace. If you do not use the **-n <namespace>** option, your current project is used. To change projects, use **oc project <namespace>**. If you do not include a **<pvc_name>** argument, an error message appears.
--image string	Lists the **libguestfs-tools** container image. You can configure the container to use a custom image by using the **--image** option.

Flag Option	Description
--kvm	Indicates that **kvm** is used by the **libguestfs-tools** container. By default, **virtctl guestfs** sets up **kvm** for the interactive container, which greatly speeds up the **libguest-tools** execution because it uses QEMU. If a cluster does not have any **kvm** supporting nodes, you must disable **kvm** by setting the option **--kvm=false**. If not set, the **libguestfs-tools** pod remains pending because it cannot be scheduled on any node.
--pull-policy string	Shows the pull policy for the **libguestfs** image. You can also overwrite the image's pull policy by setting the **pull-policy** option.

The command also checks if a PVC is in use by another pod, in which case an error message appears. However, once the **libguestfs-tools** process starts, the setup cannot avoid a new pod using the same PVC. You must verify that there are no active **virtctl guestfs** pods before starting the VM that accesses the same PVC.

NOTE

The **virtctl guestfs** command accepts only a single PVC attached to the interactive pod.

3.2.4. Using Ansible

To use the Ansible collection for OpenShift Virtualization, see Red Hat Ansible Automation Hub (Red Hat Hybrid Cloud Console).

CHAPTER 4. INSTALLING

4.1. PREPARING YOUR CLUSTER FOR OPENSHIFT VIRTUALIZATION

Before you install OpenShift Virtualization, review this section to ensure that your cluster meets the requirements.

4.1.1. Compatible platforms

You can use the following platforms with OpenShift Virtualization:

- On-premise bare metal servers. See Planning a bare metal cluster for OpenShift Virtualization .
- Bare metal clusters installed on ARM64-based (**arm64**, also known as **aarch64**) systems.
- IBM Z® or IBM® LinuxONE (s390x architecture) systems where an OpenShift Container Platform cluster is installed in logical partitions (LPARs). See Preparing to install on IBM Z and IBM LinuxONE.

Cloud platforms

OpenShift Virtualization is also compatible with a variety of public cloud platforms. Each cloud platform has specific storage provider options available. The following table outlines which platforms are fully supported (GA) and which are currently offered as Technology Preview features.

IMPORTANT

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

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

Vendor	Status	Storage	Related links
Amazon Web Services (AWS)	GA	Elastic Block Store (EBS), Red Hat OpenShift Data Foundation (ODF), Portworx, FSx (NetApp)	• Installing a cluster on AWS with customizations

Vendor	Status	Storage	Related links
Red Hat OpenShift Service on AWS (ROSA)	GA	EBS, Portworx, FSx (Q3), ODF	• OpenShift Virtualization in the Red Hat OpenShift Service on AWS documentation • What is Red Hat OpenShift Service on AWS? in the AWS documentation
Oracle Cloud Infrastructure (OCI)	GA	OCI native storage	• OpenShift Virtualization and Oracle Cloud Infrastructure known issues and limitations in the Red Hat Knowledgebase • Installing OpenShift Virtualization on OCI in the **oracle-quickstart/oci-openshift** GitHub repository
Azure Red Hat OpenShift (ARO)	GA	ODF	• OpenShift Virtualization for Azure Red Hat OpenShift (preview) in the Microsoft documentation
Google Cloud	Technology Preview	Google Cloud native storage	• OpenShift Virtualization and Google Cloud known storage issues and limitations in the Red Hat Knowledgebase

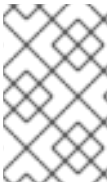
TIP

For platform-specific networking information, see the networking overview.

Bare metal instances or servers offered by other cloud providers are not supported.

4.1.1.1. OpenShift Virtualization on AWS bare metal

You can run OpenShift Virtualization on an Amazon Web Services (AWS) bare metal OpenShift Container Platform cluster.



NOTE

OpenShift Virtualization is also supported on Red Hat OpenShift Service on AWS (ROSA) Classic clusters, which have the same configuration requirements as AWS bare-metal clusters.

Before you set up your cluster, review the following summary of supported features and limitations:

Installing

- You can install the cluster by using installer-provisioned infrastructure, ensuring that you specify bare-metal instance types for the worker nodes. For example, you can use the **c5n.metal** type value for a machine based on x86_64 architecture. You specify bare-metal instance types by editing the **install-config.yaml** file.
  For more information, see the OpenShift Container Platform documentation about installing on AWS.

Accessing virtual machines (VMs)

- There is no change to how you access VMs by using the **virtctl** CLI tool or the OpenShift Container Platform web console.
- You can expose VMs by using a **NodePort** or **LoadBalancer** service.

NOTE

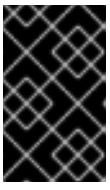
The load balancer approach is preferable because OpenShift Container Platform automatically creates the load balancer in AWS and manages its lifecycle. A security group is also created for the load balancer, and you can use annotations to attach existing security groups. When you remove the service, OpenShift Container Platform removes the load balancer and its associated resources.

Networking

- You cannot use Single Root I/O Virtualization (SR-IOV) or bridge Container Network Interface (CNI) networks, including virtual LAN (VLAN). If your application requires a flat layer 2 network or control over the IP pool, consider using OVN-Kubernetes secondary overlay networks.

Storage

- You can use any storage solution that is certified by the storage vendor to work with the underlying platform.



IMPORTANT

AWS bare metal, Red Hat OpenShift Service on AWS, and Red Hat OpenShift Service on AWS classic architecture clusters might have different supported storage solutions. Ensure that you confirm support with your storage vendor.

- Using Amazon Elastic File System (EFS) or Amazon Elastic Block Store (EBS) with OpenShift Virtualization might cause performance and functionality limitations as shown in the following table:

Table 4.1. EFS and EBS performance and functionality limitations

Feature	EBS volume			EFS volume	Shared storage solutions
	gp2	gp3	io2		
VM live migration	Not available	Not available	Available	Available	Available
Fast VM creation by using cloning	Available			Not available	Available
VM backup and restore by using snapshots	Available			Not available	Available

Consider using CSI storage, which supports ReadWriteMany (RWX), cloning, and snapshots to enable live migration, fast VM creation, and VM snapshots capabilities.

Hosted control planes (HCPs)

- HCPs for OpenShift Virtualization are not currently supported on AWS infrastructure.

Additional resources

- Connecting a virtual machine to an OVN-Kubernetes secondary network
- Exposing a virtual machine by using a service

4.1.1.2. ARM64 compatibility

Using OpenShift Virtualization on an OpenShift Container Platform cluster installed on an ARM64 system is generally available (GA).

Before using OpenShift Virtualization on an ARM64-based system, consider the following limitations:

Operating system

- Only Linux-based guest operating systems are supported.
- All virtualization limitations for RHEL also apply to OpenShift Virtualization. For more information, see How virtualization on ARM64 differs from AMD64 and Intel 64 in the RHEL documentation.

Live migration

- Live migration is **not supported** on ARM64-based OpenShift Container Platform clusters.
- Hotplug is not supported on ARM64-based clusters because it depends on live migration.

VM creation

- RHEL 10 supports instance types and preferences, but not templates.
- RHEL 9 supports templates, instance types, and preferences.

4.1.1.3. IBM Z and IBM LinuxONE compatibility

You can use OpenShift Virtualization in an OpenShift Container Platform cluster that is installed in logical partitions (LPARs) on an IBM Z® or IBM® LinuxONE (**s390x** architecture) system.

Some features are not currently available on **s390x** architecture, while others require workarounds or procedural changes. These lists are subject to change.

Currently unavailable features
The following features are currently not available on **s390x** architecture:

- Memory hot plugging and hot unplugging
- Node Health Check Operator
- SR-IOV Operator
- PCI passthrough
- OpenShift Virtualization cluster checkup framework
- OpenShift Virtualization on a cluster installed in FIPS mode
- IPv6
- IBM® Storage scale
- Hosted control planes for OpenShift Virtualization
- VM pages using HugePages

The following features are not applicable on **s390x** architecture:

- virtual Trusted Platform Module (vTPM) devices
- UEFI mode for VMs
- USB host passthrough
- Configuring virtual GPUs
- Creating and managing Windows VMs
- Hyper-V

Functionality differences
The following features are available for use on s390x architecture but function differently or require procedural changes:

- When deleting a virtual machine by using the web console , the **grace period** option is ignored.

- When configuring the default CPU model, the **spec.defaultCPUModel** value is **"gen15b"** for an IBM Z cluster.
- When configuring a downward metrics device, if you use a VM preference, the **spec.preference.name** value must be set to **rhel.9.s390x** or another available preference with the format ***.s390x**.
- When creating virtual machines from instance types, you are not allowed to set **spec.domain.memory.maxGuest** because memory hot plugging is not supported on IBM Z®.
- Prometheus queries for VM guests could have inconsistent outcome in comparison to **x86**.

4.1.2. Important considerations for any platform

Before you install OpenShift Virtualization on any platform, note the following caveats and considerations.

Installation method considerations

You can use any installation method, including user-provisioned, installer-provisioned, or Assisted Installer, to deploy OpenShift Container Platform. However, the installation method and the cluster topology might affect OpenShift Virtualization functionality, such as snapshots or live migration.

Red Hat OpenShift Data Foundation

If you deploy OpenShift Virtualization with Red Hat OpenShift Data Foundation, you must create a dedicated storage class for Windows virtual machine disks. See Optimizing ODF PersistentVolumes for Windows VMs for details.

IPv6

OpenShift Virtualization support for single-stack IPv6 clusters is limited to the OVN-Kubernetes localnet and Linux bridge Container Network Interface (CNI) plugins.

IMPORTANT

Deploying OpenShift Virtualization on a single-stack IPv6 cluster is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

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

FIPS mode

If you install your cluster in FIPS mode, no additional setup is required for OpenShift Virtualization.

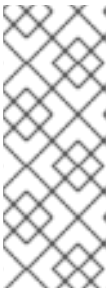
4.1.3. Hardware and operating system requirements

Review the following hardware and operating system requirements for OpenShift Virtualization.

4.1.3.1. CPU requirements

- Supported by Red Hat Enterprise Linux (RHEL) 9.

See Red Hat Ecosystem Catalog for supported CPUs.



NOTE

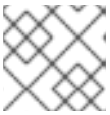
If your worker nodes have different CPUs, live migration failures might occur because different CPUs have different capabilities. You can mitigate this issue by ensuring that your worker nodes have CPUs with the appropriate capacity and by configuring node affinity rules for your virtual machines.

See Configuring a required node affinity rule for details.

- Supports AMD64, Intel 64-bit (x86-64-v2), IBM Z® (**s390x**), or ARM64-based (**arm64** or **aarch64**) architectures and their respective CPU extensions.
- Intel VT-x, AMD-V, or ARM virtualization extensions are enabled, or **s390x** virtualization support is enabled.
- NX (no execute) flag is enabled.
- If you use **s390x** architecture, the default CPU model is set to **gen15b**.

4.1.3.2. Operating system requirements

- Red Hat Enterprise Linux CoreOS (RHCOS) installed on worker nodes.
  See About RHCOS for details.

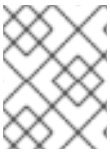


NOTE

RHEL worker nodes are not supported.

4.1.3.3. Storage requirements

- Supported by OpenShift Container Platform. See Optimizing storage.
- You must create a default OpenShift Virtualization or OpenShift Container Platform storage class. The purpose of this is to address the unique storage needs of VM workloads and offer optimized performance, reliability, and user experience. If both OpenShift Virtualization and OpenShift Container Platform default storage classes exist, the OpenShift Virtualization class takes precedence when creating VM disks.



NOTE

To mark a storage class as the default for virtualization workloads, set the annotation **storageclass.kubevirt.io/is-default-virt-class** to **"true"**.

- If the storage provisioner supports snapshots, you must associate a **VolumeSnapshotClass** object with the default storage class.

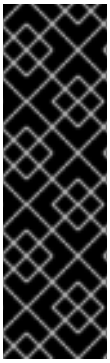
4.1.3.3.1. About volume and access modes for virtual machine disks

If you use the storage API with known storage providers, the volume and access modes are selected automatically. However, if you use a storage class that does not have a storage profile, you must configure the volume and access mode.

For a list of known storage providers for OpenShift Virtualization, see the Red Hat Ecosystem Catalog .

For best results, use the **ReadWriteMany** (RWX) access mode and the **Block** volume mode. This is important for the following reasons:

- **ReadWriteMany** (RWX) access mode is required for live migration.
- The **Block** volume mode performs significantly better than the **Filesystem** volume mode. This is because the **Filesystem** volume mode uses more storage layers, including a file system layer and a disk image file. These layers are not necessary for VM disk storage.
  For example, if you use Red Hat OpenShift Data Foundation, Ceph RBD volumes are preferable to CephFS volumes.



IMPORTANT

You cannot live migrate virtual machines with the following configurations:

- Storage volume with **ReadWriteOnce** (RWO) access mode
- Passthrough features such as GPUs

Set the **evictionStrategy** field to **None** for these virtual machines. The **None** strategy powers down VMs during node reboots.

4.1.4. Live migration requirements

- Shared storage with **ReadWriteMany** (RWX) access mode.
- Sufficient RAM and network bandwidth.

NOTE

You must ensure that there is enough memory request capacity in the cluster to support node drains that result in live migrations. You can determine the approximate required spare memory by using the following calculation:

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

The default number of migrations that can run in parallel in the cluster is 5.

- If the virtual machine uses a host model CPU, the nodes must support the virtual machine's host model CPU.

NOTE

A dedicated Multus network for live migration is highly recommended. A dedicated network minimizes the effects of network saturation on tenant workloads during migration.

4.1.5. Physical resource overhead requirements

OpenShift Virtualization is an add-on to OpenShift Container Platform and imposes additional overhead

that you must account for when planning a cluster. Each cluster machine must accommodate the following overhead requirements in addition to the OpenShift Container Platform requirements. Oversubscribing the physical resources in a cluster can affect performance.

IMPORTANT

The numbers noted in this documentation are based on Red Hat's test methodology and setup. These numbers can vary based on your own individual setup and environments.

Memory overhead
Calculate the memory overhead values for OpenShift Virtualization by using the equations below.

Cluster memory overhead

```
Memory overhead per infrastructure node ≈ 150 MiB
```

```
Memory overhead per worker node ≈ 360 MiB
```

Additionally, OpenShift Virtualization environment resources require a total of 2179 MiB of RAM that is spread across all infrastructure nodes.

Virtual machine memory overhead

```
Memory overhead per virtual machine ≈ (0.002 × requested memory) \
            + 218 MiB \ 1
            + 8 MiB × (number of vCPUs) \ 2
            + 16 MiB × (number of graphics devices) \ 3
            + (additional memory overhead) 4
```

1. Required for the processes that run in the **virt-launcher** pod.
2. Number of virtual CPUs requested by the virtual machine.
3. Number of virtual graphics cards requested by the virtual machine.
4. Additional memory overhead:
    - If your environment includes a Single Root I/O Virtualization (SR-IOV) network device or a Graphics Processing Unit (GPU), allocate 1 GiB additional memory overhead for each device.
    - If Secure Encrypted Virtualization (SEV) is enabled, add 256 MiB.
    - If Trusted Platform Module (TPM) is enabled, add 53 MiB.

CPU overhead
Calculate the cluster processor overhead requirements for OpenShift Virtualization by using the equation below. The CPU overhead per virtual machine depends on your individual setup.

Cluster CPU overhead

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

OpenShift Virtualization increases the overall utilization of cluster level services such as logging, routing, and monitoring. To account for this workload, ensure that nodes that host infrastructure components have capacity allocated for 4 additional cores (4000 millicores) distributed across those nodes.

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

Each worker node that hosts virtual machines must have capacity for 2 additional cores (2000 millicores) for OpenShift Virtualization management workloads in addition to the CPUs required for virtual machine workloads.

Virtual machine CPU overhead

If dedicated CPUs are requested, there is a 1:1 impact on the cluster CPU overhead requirement. Otherwise, there are no specific rules about how many CPUs a virtual machine requires.

Storage overhead

Use the guidelines below to estimate storage overhead requirements for your OpenShift Virtualization environment.

Cluster storage overhead

> Aggregated storage overhead per node ≈ 10 GiB

10 GiB is the estimated on-disk storage impact for each node in the cluster when you install OpenShift Virtualization.

Virtual machine storage overhead

Storage overhead per virtual machine depends on specific requests for resource allocation within the virtual machine. The request could be for ephemeral storage on the node or storage resources hosted elsewhere in the cluster. OpenShift Virtualization does not currently allocate any additional ephemeral storage for the running container itself.

Example

As a cluster administrator, if you plan to host 10 virtual machines in the cluster, each with 1 GiB of RAM and 2 vCPUs, the memory impact across the cluster is 11.68 GiB. The estimated on-disk storage impact for each node in the cluster is 10 GiB and the CPU impact for worker nodes that host virtual machine workloads is a minimum of 2 cores.

4.1.6. Single-node OpenShift differences

You can install OpenShift Virtualization on single-node OpenShift.

However, you should be aware that Single-node OpenShift does not support the following features:

- High availability
- Pod disruption
- Live migration
- Virtual machines or templates that have an eviction strategy configured

Additional resources

- Glossary of common terms for OpenShift Container Platform storage

4.1.7. Object maximums

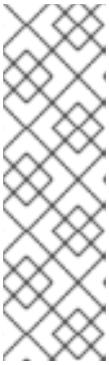
You must consider the following tested object maximums when planning your cluster:

- OpenShift Container Platform object maximums
- OpenShift Virtualization supported limits

4.1.8. Cluster high-availability options

You can configure one of the following high-availability (HA) options for your cluster:

- Automatic high availability for installer-provisioned infrastructure (IPI) is available by deploying machine health checks.



NOTE

In OpenShift Container Platform clusters installed using installer-provisioned infrastructure and with a properly configured **MachineHealthCheck** resource, if a node fails the machine health check and becomes unavailable to the cluster, it is recycled. What happens next with VMs that ran on the failed node depends on a series of conditions. See Run strategies for more detailed information about the potential outcomes and how run strategies affect those outcomes.

Currently, IPI is not supported on IBM Z®.

- Automatic high availability for both IPI and non-IPI is available by using the **Node Health Check Operator** on the OpenShift Container Platform cluster to deploy the **NodeHealthCheck** controller. The controller identifies unhealthy nodes and uses a remediation provider, such as the Self Node Remediation Operator or Fence Agents Remediation Operator, to remediate the unhealthy nodes. For more information on remediation, fencing, and maintaining nodes, see the Workload Availability for Red Hat OpenShift documentation.

NOTE

Fence Agents Remediation uses supported fencing agents to reset failed nodes faster than the Self Node Remediation Operator. This improves overall virtual machine high availability. For more information, see the OpenShift Virtualization - Fencing and VM High Availability Guide knowledgebase article.

- High availability for any platform is available by using either a monitoring system or a qualified human to monitor node availability. When a node is lost, shut it down and run **oc delete node <lost_node>**.

NOTE

Without an external monitoring system or a qualified human monitoring node health, virtual machines lose high availability.

4.2. INSTALLING OPENSHIFT VIRTUALIZATION

Install OpenShift Virtualization to add virtualization functionality to your OpenShift Container Platform cluster.

IMPORTANT

If you install OpenShift Virtualization in a restricted environment with no internet connectivity, you must configure Operator Lifecycle Manager for disconnected environments.

If you have limited internet connectivity, you can configure proxy support in OLM to access the software catalog.

4.2.1. Installing the OpenShift Virtualization Operator

Install the OpenShift Virtualization Operator by using the OpenShift Container Platform web console or the command line.

4.2.1.1. Installing the OpenShift Virtualization Operator by using the web console

You can deploy the OpenShift Virtualization Operator by using the OpenShift Container Platform web console.

Prerequisites

- Install OpenShift Container Platform 4.20 on your cluster.
- Log in to the OpenShift Container Platform web console as a user with **cluster-admin** permissions.

Procedure

1. From the **Administrator** perspective, click **Ecosystem → Software Catalog**.
2. In the **Filter by keyword** field, type **Virtualization**.
3. Select the **OpenShift Virtualization Operator** tile with the **Red Hat** source label.
4. Read the information about the Operator and click **Install**.
5. On the **Install Operator** page:
   a. Select **stable** from the list of available **Update Channel** options. This ensures that you install the version of OpenShift Virtualization that is compatible with your OpenShift Container Platform version.
   b. For **Installed Namespace**, ensure that the **Operator recommended namespace** option is selected. This installs the Operator in the mandatory **openshift-cnv** namespace, which is automatically created if it does not exist.

WARNING

Attempting to install the OpenShift Virtualization Operator in a namespace other than **openshift-cnv** causes the installation to fail.

c. For **Approval Strategy**, it is highly recommended that you select **Automatic**, which is the default value, so that OpenShift Virtualization automatically updates when a new version is available in the **stable** update channel.
Selecting the **Manual** approval strategy is not recommended, as it poses a high risk to cluster support and functionality. Only select **Manual** if you fully understand these risks and cannot use **Automatic**.

WARNING

Because OpenShift Virtualization is only supported when used with the corresponding OpenShift Container Platform version, missing OpenShift Virtualization updates can cause your cluster to become unsupported.

6. Click **Install** to make the Operator available to the **openshift-cnv** namespace.
7. When the Operator installs successfully, click **Create HyperConverged**.
8. Optional: Configure **Infra** and **Workloads** node placement options for OpenShift Virtualization components.
9. Click **Create** to launch OpenShift Virtualization.

Verification

- Navigate to the **Workloads → Pods** page and monitor the OpenShift Virtualization pods until they are all **Running**. After all the pods display the **Running** state, you can use OpenShift Virtualization.

4.2.1.2. Installing the OpenShift Virtualization Operator by using the command line

Subscribe to the OpenShift Virtualization catalog and install the OpenShift Virtualization Operator by applying manifests to your cluster.

4.2.1.2.1. Subscribing to the OpenShift Virtualization catalog by using the CLI

Before you install OpenShift Virtualization, you must subscribe to the OpenShift Virtualization catalog. Subscribing gives the **openshift-cnv** namespace access to the OpenShift Virtualization Operators.

To subscribe, configure **Namespace**, **OperatorGroup**, and **Subscription** objects by applying a single manifest to your cluster.

Prerequisites

- Install OpenShift Container Platform 4.20 on your cluster.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create a YAML file that contains the following manifest:

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-cnv
  labels:
    openshift.io/cluster-monitoring: "true"
---
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.v4.20.0
  channel: "stable" (1)
```

(1) Using the **stable** channel ensures that you install the version of OpenShift Virtualization that is compatible with your OpenShift Container Platform version.

2. Create the required **Namespace**, **OperatorGroup**, and **Subscription** objects for OpenShift Virtualization by running the following command:

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

Verification

You must verify that the subscription creation was successful before you can proceed with installing OpenShift Virtualization.

1. Check that the **ClusterServiceVersion** (CSV) object was created successfully. Run the following command and verify the output:

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

   If the CSV was created successfully, the output shows an entry that contains a **NAME** value of **kubevirt-hyperconverged-operator-***, a **DISPLAY** value of **OpenShift Virtualization**, and a **PHASE** value of **Succeeded**, as shown in the following example output:

   Example output

```
NAME                                      DISPLAY                    VERSION   REPLACES
PHASE
kubevirt-hyperconverged-operator.v4.20.0   OpenShift Virtualization   4.20.0    kubevirt-
hyperconverged-operator.v4.19.0   Succeeded
```

2. Check that the **HyperConverged** custom resource (CR) has the correct version. Run the following command and verify the output:

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

   Example output

```
{
"name": "operator",
"version": "4.20.0"
}
```

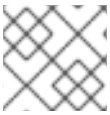
3. Verify the **HyperConverged** CR conditions. Run the following command and check the output:

```
$ oc get hco kubevirt-hyperconverged -n openshift-cnv -o json | jq -r '.status.conditions[] |
{type,status}'
```

   Example output

```
{
  "type": "ReconcileComplete",
  "status": "True"
}
{
  "type": "Available",
  "status": "True"
}
{
  "type": "Progressing",
  "status": "False"
}
{
  "type": "Degraded",
  "status": "False"
}
```

```
{
  "type": "Upgradeable",
  "status": "True"
}
```



NOTE

You can configure certificate rotation parameters in the YAML file.

4.2.1.2.2. Deploying the OpenShift Virtualization Operator by using the CLI

You can deploy the OpenShift Virtualization Operator by using the **oc** CLI.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Subscribe to the OpenShift Virtualization catalog in the **openshift-cnv** namespace.
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create a YAML file that contains the following manifest:

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

2. Deploy the OpenShift Virtualization Operator by running the following command:

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

Verification

- Ensure that OpenShift Virtualization deployed successfully by watching the **PHASE** of the cluster service version (CSV) in the **openshift-cnv** namespace. Run the following command:

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

The following output displays if deployment was successful:

Example output

```
NAME                                       DISPLAY                    VERSION   REPLACES   PHASE
kubevirt-hyperconverged-operator.v4.20.0   OpenShift Virtualization   4.20.0
Succeeded
```

4.2.2. Next steps

- The hostpath provisioner is a local storage provisioner designed for OpenShift Virtualization. If you want to configure local storage for virtual machines, you must enable the hostpath provisioner first.

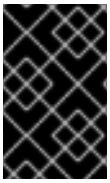
4.3. UNINSTALLING OPENSHIFT VIRTUALIZATION

You uninstall OpenShift Virtualization by using the web console or the command-line interface (CLI) to delete the OpenShift Virtualization workloads, the Operator, and its resources.

4.3.1. Uninstalling OpenShift Virtualization by using the web console

You uninstall OpenShift Virtualization by using the web console to perform the following tasks:

1. Delete the **HyperConverged** CR.
2. Delete the OpenShift Virtualization Operator.
3. Delete the **openshift-cnv** namespace.
4. Delete the OpenShift Virtualization custom resource definitions (CRDs) .



IMPORTANT

You must first delete all virtual machines, and virtual machine instances.

You cannot uninstall OpenShift Virtualization while its workloads remain on the cluster.

4.3.1.1. Deleting the HyperConverged custom resource

To uninstall OpenShift Virtualization, you first delete the **HyperConverged** custom resource (CR).

Prerequisites

- You have access to an OpenShift Container Platform cluster using an account with **cluster-admin** permissions.

Procedure

1. Navigate to the **Ecosystem → Installed Operators** page.
2. Select the OpenShift Virtualization Operator.
3. Click the **OpenShift Virtualization Deployment** tab.
4. Click the Options menu  beside **kubevirt-hyperconverged** and select **Delete HyperConverged**.
5. Click **Delete** in the confirmation window.

4.3.1.2. Deleting Operators from a cluster using the web console

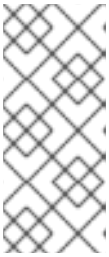
Cluster administrators can delete installed Operators from a selected namespace by using the web console.

Prerequisites

- You have access to the OpenShift Container Platform cluster web console using an account with **cluster-admin** permissions.

Procedure

1. Navigate to the **Ecosystem → Installed Operators** page.
2. Scroll or enter a keyword into the **Filter by name** field to find the Operator that you want to remove. Then, click on it.
3. On the right side of the **Operator Details** page, select **Uninstall Operator** from the **Actions** list. An **Uninstall Operator?** dialog box is displayed.
4. Select **Uninstall** to remove the Operator, Operator deployments, and pods. Following this action, the Operator stops running and no longer receives updates.



NOTE

This action does not remove resources managed by the Operator, including custom resource definitions (CRDs) and custom resources (CRs). Dashboards and navigation items enabled by the web console and off-cluster resources that continue to run might need manual clean up. To remove these after uninstalling the Operator, you might need to manually delete the Operator CRDs.

4.3.1.3. Deleting a namespace using the web console

You can delete a namespace by using the OpenShift Container Platform web console.

Prerequisites

- You have access to the OpenShift Container Platform cluster using an account with **cluster-admin** permissions.

Procedure

1. Navigate to **Administration → Namespaces**.
2. Locate the namespace that you want to delete in the list of namespaces.
3. On the far right side of the namespace listing, select **Delete Namespace** from the Options menu  .
4. When the **Delete Namespace** pane opens, enter the name of the namespace that you want to delete in the field.
5. Click **Delete**.

4.3.1.4. Deleting OpenShift Virtualization custom resource definitions

You can delete the OpenShift Virtualization custom resource definitions (CRDs) by using the web console.

Prerequisites

- You have access to the OpenShift Container Platform cluster using an account with **cluster-admin** permissions.

Procedure

1. Navigate to **Administration → CustomResourceDefinitions**.
2. Select the **Label** filter and enter **operators.coreos.com/kubevirt-hyperconverged.openshift-cnv** in the **Search** field to display the OpenShift Virtualization CRDs.
3. Click the Options menu  beside each CRD and select **Delete CustomResourceDefinition**.

4.3.2. Uninstalling OpenShift Virtualization by using the CLI

You can uninstall OpenShift Virtualization by using the OpenShift CLI (**oc**).

Prerequisites

- You have access to the OpenShift Container Platform cluster using an account with **cluster-admin** permissions.
- You have installed the OpenShift CLI (**oc**).
- You have deleted all virtual machines and virtual machine instances. You cannot uninstall OpenShift Virtualization while its workloads remain on the cluster.

Procedure

1. Delete the **HyperConverged** custom resource:

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

2. Delete the OpenShift Virtualization Operator subscription:

```
$ oc delete subscription hco-operatorhub -n openshift-cnv
```

3. Delete the OpenShift Virtualization **ClusterServiceVersion** resource:

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

4. Delete the OpenShift Virtualization namespace:

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

5. List the OpenShift Virtualization custom resource definitions (CRDs) by running the **oc delete crd** command with the **dry-run** option:

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

Example output

```
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. Delete the CRDs by running the **oc delete crd** command without the **dry-run** option:

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

Additional resources

- Deleting virtual machines
- Deleting virtual machine instances

CHAPTER 5. POSTINSTALLATION CONFIGURATION

5.1. POSTINSTALLATION CONFIGURATION

The following procedures are typically performed after OpenShift Virtualization is installed. You can configure the components that are relevant for your environment:

- Node placement rules for OpenShift Virtualization Operators, workloads, and controllers
- Network configuration:
  - Installing the Kubernetes NMState and SR-IOV Operators
  - Configuring a Linux bridge network for external access to virtual machines (VMs)
  - Configuring a dedicated secondary network for live migration
  - Configuring an SR-IOV network
  - Enabling the creation of load balancer services by using the OpenShift Container Platform web console
- Storage configuration:
  - Defining a default storage class for the Container Storage Interface (CSI)
  - Configuring local storage by using the Hostpath Provisioner (HPP)

5.2. SPECIFYING NODES FOR OPENSHIFT VIRTUALIZATION COMPONENTS

The default scheduling for virtual machines (VMs) on bare metal nodes is appropriate. Optionally, you can specify the nodes where you want to deploy OpenShift Virtualization Operators, workloads, and controllers by configuring node placement rules.

NOTE

You can configure node placement rules for some components after installing OpenShift Virtualization, but virtual machines cannot be present if you want to configure node placement rules for workloads.

5.2.1. About node placement rules for OpenShift Virtualization components

You can use node placement rules for the following tasks:

- Deploy virtual machines only on nodes intended for virtualization workloads.
- Deploy Operators only on infrastructure nodes.
- Maintain separation between workloads.

Depending on the object, you can use one or more of the following rule types:

nodeSelector

Allows pods to be scheduled on nodes that are labeled with the key-value pair or pairs that you specify in this field. The node must have labels that exactly match all listed pairs.

affinity

Enables you to use more expressive syntax to set rules that match nodes with pods. Affinity also allows for more nuance in how the rules are applied. For example, you can specify that a rule is a preference, not a requirement. If a rule is a preference, pods are still scheduled when the rule is not satisfied.

tolerations

Allows pods to be scheduled on nodes that have matching taints. If a taint is applied to a node, that node only accepts pods that tolerate the taint.

5.2.2. Applying node placement rules

You can apply node placement rules by editing a **Subscription**, **HyperConverged**, or **HostPathProvisioner** object using the command line.

Prerequisites

- The **oc** CLI tool is installed.
- You are logged in with cluster administrator permissions.

Procedure

1. Edit the object in your default editor by running the following command:

```
$ oc edit <resource_type> <resource_name> -n openshift-cnv
```

2. Save the file to apply the changes.

5.2.3. Node placement rule examples

You can specify node placement rules for a OpenShift Virtualization component by editing a **Subscription**, **HyperConverged**, or **HostPathProvisioner** object.

5.2.3.1. Subscription object node placement rule examples

To specify the nodes where OLM deploys the OpenShift Virtualization Operators, edit the **Subscription** object during OpenShift Virtualization installation.

Currently, you cannot configure node placement rules for the **Subscription** object by using the web console.

The **Subscription** object does not support the **affinity** node pplacement rule.

Example Subscription object with nodeSelector rule

```
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.v4.20.0
  channel: "stable"
  config:
    nodeSelector:
      example.io/example-infra-key: example-infra-value (1)
```

(1) OLM deploys the OpenShift Virtualization Operators on nodes labeled **example.io/example-infra-key = example-infra-value**.

Example Subscription object with tolerations rule

```
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.v4.20.0
  channel: "stable"
  config:
    tolerations:
    - key: "key"
      operator: "Equal"
      value: "virtualization" (1)
      effect: "NoSchedule"
```

(1) OLM deploys OpenShift Virtualization Operators on nodes labeled **key = virtualization:NoSchedule** taint. Only pods with the matching tolerations are scheduled on these nodes.

5.2.3.2. HyperConverged object node placement rule example

To specify the nodes where OpenShift Virtualization deploys its components, you can edit the **nodePlacement** object in the HyperConverged custom resource (CR) file that you create during OpenShift Virtualization installation.

Example HyperConverged object with nodeSelector rule

```
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) Infrastructure resources are placed on nodes labeled **example.io/example-infra-key = example-infra-value**.

(2) workloads are placed on nodes labeled **example.io/example-workloads-key = example-workloads-value**.

Example HyperConverged object with affinity rule

```
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. Infrastructure resources are placed on nodes labeled **example.io/example-infra-key = example-value**.
2. workloads are placed on nodes labeled **example.io/example-workloads-key = example-workloads-value**.
3. Nodes that have more than eight CPUs are preferred for workloads, but if they are not available, pods are still scheduled.

Example HyperConverged object with tolerations rule

```
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. Nodes reserved for OpenShift Virtualization components are labeled with the **key = virtualization:NoSchedule** taint. Only pods with matching tolerations are scheduled on reserved nodes.

5.2.3.3. HostPathProvisioner object node placement rule example

You can edit the **HostPathProvisioner** object directly or by using the web console.

WARNING

You must schedule the hostpath provisioner and the OpenShift Virtualization components on the same nodes. Otherwise, virtualization pods that use the hostpath provisioner cannot run. You cannot run virtual machines.

After you deploy a virtual machine (VM) with the hostpath provisioner (HPP) storage class, you can remove the hostpath provisioner pod from the same node by using the node selector. However, you must first revert that change, at least for that specific node, and wait for the pod to run before trying to delete the VM.

You can configure node placement rules by specifying **nodeSelector**, **affinity**, or **tolerations** for the **spec.workload** field of the **HostPathProvisioner** object that you create when you install the hostpath provisioner.

Example HostPathProvisioner object with nodeSelector rule

```
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) Workloads are placed on nodes labeled **example.io/example-workloads-key = example-workloads-value**.

5.2.4. Additional resources

- Specifying nodes for virtual machines
- Placing pods on specific nodes using node selectors
- Controlling pod placement on nodes using node affinity rules
- Controlling pod placement using node taints

5.3. POSTINSTALLATION NETWORK CONFIGURATION

By default, OpenShift Virtualization is installed with a single, internal pod network.

After you install OpenShift Virtualization, you can install networking Operators and configure additional networks.

5.3.1. Installing networking Operators

You must install the Kubernetes NMState Operator to configure a Linux bridge network for live migration or external access to virtual machines (VMs). For installation instructions, see Installing the Kubernetes NMState Operator by using the web console.

You can install the SR-IOV Operator to manage SR-IOV network devices and network attachments. For installation instructions, see Installing the SR-IOV Network Operator .

You can add the About MetalLB and the MetalLB Operator to manage the lifecycle for an instance of MetalLB on your cluster. For installation instructions, see Installing the MetalLB Operator from the software catalog using the web console.

5.3.2. Configuring a Linux bridge network

After you install the Kubernetes NMState Operator, you can configure a Linux bridge network for live migration or external access to virtual machines (VMs).

5.3.2.1. Creating a Linux bridge NNCP

You can create a **NodeNetworkConfigurationPolicy** (NNCP) manifest for a Linux bridge network.

Prerequisites

- You have installed the Kubernetes NMState Operator.

Procedure

- Create the **NodeNetworkConfigurationPolicy** manifest. This example includes sample values that you must replace with your own information.

```
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) Name of the policy.

(2) Name of the interface.

(3) Optional: Human-readable description of the interface.

(4) The type of interface. This example creates a bridge.

(5) The requested state for the interface after creation.

(6) Disables IPv4 in this example.

(7) Disables STP in this example.

(8) The node NIC to which the bridge is attached.

NOTE

To create the NNCP manifest for a Linux bridge using OSA with IBM Z®, you must disable VLAN filtering by the setting the **rx-vlan-filter** to **false** in the **NodeNetworkConfigurationPolicy** manifest.

Alternatively, if you have SSH access to the node, you can disable VLAN filtering by running the following command:

```
$ sudo ethtool -K <osa-interface-name> rx-vlan-filter off
```

5.3.2.2. Creating a Linux bridge NAD by using the web console

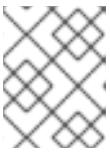
You can create a network attachment definition (NAD) to provide layer-2 networking to pods and virtual machines by using the OpenShift Container Platform web console.

WARNING

Configuring IP address management (IPAM) in a network attachment definition for virtual machines is not supported.

Procedure

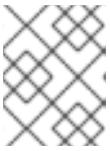
1. In the web console, click **Networking** → **NetworkAttachmentDefinitions**.
2. Click **Create Network Attachment Definition**



NOTE

The network attachment definition must be in the same namespace as the pod or virtual machine.

3. Enter a unique **Name** and optional **Description**.
4. Select **CNV Linux bridge** from the **Network Type** list.
5. Enter the name of the bridge in the **Bridge Name** field.
6. Optional: If the resource has VLAN IDs configured, enter the ID numbers in the **VLAN Tag Number** field.



NOTE

OSA interfaces on IBM Z® do not support VLAN filtering and VLAN-tagged traffic is dropped. Avoid using VLAN-tagged NADs with OSA interfaces.

7. Optional: Select **MAC Spoof Check** to enable MAC spoof filtering. This feature provides security against a MAC spoofing attack by allowing only a single MAC address to exit the pod.

8. Click **Create**.

Next steps

- Attaching a virtual machine (VM) to a Linux bridge network

5.3.3. Configuring a network for live migration

After you have configured a Linux bridge network, you can configure a dedicated network for live migration. A dedicated network minimizes the effects of network saturation on tenant workloads during live migration.

5.3.3.1. Configuring a dedicated secondary network for live migration

To configure a dedicated secondary network for live migration, you must first create a bridge network attachment definition (NAD) by using the CLI. Then, you add the name of the **NetworkAttachmentDefinition** object to the **HyperConverged** custom resource (CR).

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You logged in to the cluster as a user with the **cluster-admin** role.
- Each node has at least two Network Interface Cards (NICs).
- The NICs for live migration are connected to the same VLAN.

Procedure

1. Create a **NetworkAttachmentDefinition** manifest according to the following example:

Example configuration file

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: my-secondary-network 1
  namespace: openshift-cnv
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "migration-bridge",
    "type": "macvlan",
    "master": "eth1", 2
    "mode": "bridge",
    "ipam": {
      "type": "whereabouts", 3
      "range": "10.200.5.0/24" 4
    }
  }'
```

1 Specify the name of the **NetworkAttachmentDefinition** object.

2 Specify the name of the NIC to be used for live migration.

3 Specify the name of the CNI plugin that provides the network for the NAD.

4 Specify an IP address range for the secondary network. This range must not overlap the IP addresses of the main network.

2. Open the **HyperConverged** CR in your default editor by running the following command:

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

3. Add the name of the **NetworkAttachmentDefinition** object to the **spec.liveMigrationConfig** stanza of the **HyperConverged** CR:

Example HyperConverged manifest

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

1 Specify the name of the Multus **NetworkAttachmentDefinition** object to be used for live migrations.

4. Save your changes and exit the editor. The **virt-handler** pods restart and connect to the secondary network.

Verification

- When the node that the virtual machine runs on is placed into maintenance mode, the VM automatically migrates to another node in the cluster. You can verify that the migration occurred over the secondary network and not the default pod network by checking the target IP address in the virtual machine instance (VMI) metadata.

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

5.3.3.2. Selecting a dedicated network by using the web console

You can select a dedicated network for live migration by using the OpenShift Container Platform web console.

Prerequisites

- You configured a Multus network for live migration.
- You created a network attachment definition for the network.

Procedure

1. Navigate to **Virtualization > Overview** in the OpenShift Container Platform web console.
2. Click the **Settings** tab and then click **Live migration**.
3. Select the network from the **Live migration network** list.

5.3.4. Configuring an SR-IOV network

After you install the SR-IOV Operator, you can configure an SR-IOV network.

5.3.4.1. Configuring SR-IOV network devices

The SR-IOV Network Operator adds the **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CustomResourceDefinition to OpenShift Container Platform. You can configure an SR-IOV network device by creating a SriovNetworkNodePolicy custom resource (CR).

NOTE

When applying the configuration specified in a **SriovNetworkNodePolicy** object, the SR-IOV Operator might drain the nodes, and in some cases, reboot nodes. Reboot only happens in the following cases:

- With Mellanox NICs (**mlx5** driver) a node reboot happens every time the number of virtual functions (VFs) increase on a physical function (PF).
- With Intel NICs, a reboot only happens if the kernel parameters do not include **intel_iommu=on** and **iommu=pt**.

It might take several minutes for a configuration change to apply.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the SR-IOV Network Operator.
- You have enough available nodes in your cluster to handle the evicted workload from drained nodes.
- You have not selected any control plane nodes for SR-IOV network device configuration.

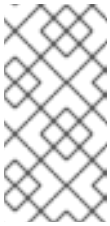
Procedure

1. Create an **SriovNetworkNodePolicy** object, and then save the YAML in the **<name>-sriov-node-network.yaml** file. Replace **<name>** with the name for this configuration.

```
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. Specify a name for the CR object.
2. Specify the namespace where the SR-IOV Operator is installed.
3. Specify the resource name of the SR-IOV device plugin. You can create multiple **SriovNetworkNodePolicy** objects for a resource name.
4. Specify the node selector to select which nodes are configured. Only SR-IOV network devices on selected nodes are configured. The SR-IOV Container Network Interface (CNI) plugin and device plugin are deployed only on selected nodes.
5. Optional: Specify an integer value between **0** and **99**. A smaller number gets higher priority, so a priority of **10** is higher than a priority of **99**. The default value is **99**.
6. Optional: Specify a value for the maximum transmission unit (MTU) of the virtual function. The maximum MTU value can vary for different NIC models.
7. Specify the number of the virtual functions (VF) to create for the SR-IOV physical network device. For an Intel network interface controller (NIC), the number of VFs cannot be larger than the total VFs supported by the device. For a Mellanox NIC, the number of VFs cannot be larger than **127**.
8. The **nicSelector** mapping selects the Ethernet device for the Operator to configure. You do not need to specify values for all the parameters.



NOTE

It is recommended to identify the Ethernet adapter with enough precision to minimize the possibility of selecting an Ethernet device unintentionally. If you specify **rootDevices**, you must also specify a value for **vendor**, **deviceID**, or **pfNames**.

If you specify both **pfNames** and **rootDevices** at the same time, ensure that they point to an identical device.

9 Optional: Specify the vendor hex code of the SR-IOV network device. The only allowed values are either **8086** or **15b3**.

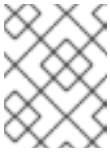
10 Optional: Specify the device hex code of SR-IOV network device. The only allowed values are **158b**, **1015**, **1017**.

11 Optional: The parameter accepts an array of one or more physical function (PF) names for the Ethernet device.

12 The parameter accepts an array of one or more PCI bus addresses for the physical function of the Ethernet device. Provide the address in the following format: **0000:02:00.1**.

13 The **vfio-pci** driver type is required for virtual functions in OpenShift Virtualization.

14 Optional: Specify whether to enable remote direct memory access (RDMA) mode. For a Mellanox card, set **isRdma** to **false**. The default value is **false**.



NOTE

If **isRDMA** flag is set to **true**, you can continue to use the RDMA enabled VF as a normal network device. A device can be used in either mode.

2. Optional: Label the SR-IOV capable cluster nodes with **SriovNetworkNodePolicy.Spec.NodeSelector** if they are not already labeled. For more information about labeling nodes, see "Understanding how to update labels on nodes".

3. Create the **SriovNetworkNodePolicy** object:

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

where **<name>** specifies the name for this configuration.

After applying the configuration update, all the pods in **sriov-network-operator** namespace transition to the **Running** status.

4. To verify that the SR-IOV network device is configured, enter the following command. Replace **<node_name>** with the name of a node with the SR-IOV network device that you just configured.

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

Next steps

- Attaching a virtual machine (VM) to an SR-IOV network

5.3.5. Enabling load balancer service creation by using the web console

You can enable the creation of load balancer services for a virtual machine (VM) by using the OpenShift Container Platform web console.

Prerequisites

- You have configured a load balancer for the cluster.
- You are logged in as a user with the **cluster-admin** role.
- You created a network attachment definition for the network.

Procedure

1. Navigate to **Virtualization → Overview**.
2. On the **Settings** tab, click **Cluster**.
3. Expand **General settings** and **SSH configuration**.
4. Set **SSH over LoadBalancer service** to on.

5.4. POSTINSTALLATION STORAGE CONFIGURATION

The following storage configuration tasks are mandatory:

- You must configure a default storage class for your cluster. Otherwise, the cluster cannot receive automated boot source updates.
- You must configure storage profiles if your storage provider is not recognized by CDI. A storage profile provides recommended storage settings based on the associated storage class.

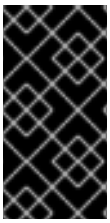
Optional: You can configure local storage by using the hostpath provisioner (HPP).

See the storage configuration overview for more options, including configuring the Containerized Data Importer (CDI), data volumes, and automatic boot source updates.

5.4.1. Configuring local storage by using the HPP

When you install the OpenShift Virtualization Operator, the Hostpath Provisioner (HPP) Operator is automatically installed. The HPP Operator creates the HPP provisioner.

The HPP is a local storage provisioner designed for OpenShift Virtualization. To use the HPP, you must create an HPP custom resource (CR).



IMPORTANT

HPP storage pools must not be in the same partition as the operating system. Otherwise, the storage pools might fill the operating system partition. If the operating system partition is full, performance can be effected or the node can become unstable or unusable.

5.4.1.1. Creating a storage class for the CSI driver with the storagePools stanza

To use the hostpath provisioner (HPP) you must create an associated storage class for the Container Storage Interface (CSI) driver.

When you create a storage class, you set parameters that affect the dynamic provisioning of persistent volumes (PVs) that belong to that storage class. You cannot update a **StorageClass** object's parameters after you create it.

NOTE

Virtual machines use data volumes that are based on local PVs. Local PVs are bound to specific nodes. While a disk image is prepared for consumption by the virtual machine, it is possible that the virtual machine cannot be scheduled to the node where the local storage PV was previously pinned.

To solve this problem, use the Kubernetes pod scheduler to bind the persistent volume claim (PVC) to a PV on the correct node. By using the **StorageClass** value with **volumeBindingMode** parameter set to **WaitForFirstConsumer**, the binding and provisioning of the PV is delayed until a pod is created using the PVC.

Procedure

1. Create a **storageclass_csi.yaml** file to define the storage class:

```
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) The two possible **reclaimPolicy** values are **Delete** and **Retain**. If you do not specify a value, the default value is **Delete**.

(2) The **volumeBindingMode** parameter determines when dynamic provisioning and volume binding occur. Specify **WaitForFirstConsumer** to delay the binding and provisioning of a persistent volume (PV) until after a pod that uses the persistent volume claim (PVC) is created. This ensures that the PV meets the pod's scheduling requirements.

(3) Specify the name of the storage pool defined in the HPP CR.

2. Save the file and exit.
3. Create the **StorageClass** object by running the following command:

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

5.5. CONFIGURING HIGHER VM WORKLOAD DENSITY

You can increase the number of virtual machines (VMs) on nodes by overcommitting memory (RAM). Increasing VM workload density can be useful in the following situations:

- You have many similar workloads.
- You have underused workloads.

NOTE

Memory overcommitment can lower workload performance on a highly utilized system.

5.5.1. Using wasp-agent to increase VM workload density

The **wasp-agent** component facilitates memory overcommitment by assigning swap resources to worker nodes. It also manages pod evictions when nodes are at risk due to high swap I/O traffic or high utilization.

IMPORTANT

Swap resources can be only assigned to virtual machine workloads (VM pods) of the **Burstable** Quality of Service (QoS) class. VM pods of the **Guaranteed** QoS class and pods of any QoS class that do not belong to VMs cannot swap resources.

For descriptions of QoS classes, see Configure Quality of Service for Pods (Kubernetes documentation).

Using **spec.domain.resources.requests.memory** in the VM manifest disables the memory overcommit configuration. Use **spec.domain.memory.guest** instead.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You are logged into the cluster with the **cluster-admin** role.
- A memory overcommit ratio is defined.
- The node belongs to a worker pool.

NOTE

The **wasp-agent** component deploys an Open Container Initiative (OCI) hook to enable swap usage for containers on the node level. The low-level nature requires the **DaemonSet** object to be privileged.

Procedure

1. Configure the **kubelet** service to permit swap usage:
   a. Create or edit a **KubeletConfig** file with the parameters shown in the following example:

Example of a KubeletConfig file

```
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
```

b. Wait for the worker nodes to sync with the new configuration by running the following command:

```
$ oc wait mcp worker --for condition=Updated=True --timeout=-1s
```

2. Provision swap by creating a **MachineConfig** object. For example:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 90-worker-swap
spec:
  config:
    ignition:
      version: 3.5.0
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Provision and enable swap
            ConditionFirstBoot=no
            ConditionPathExists=!/var/tmp/swapfile

            [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"

            [Install]
            RequiredBy=kubelet-dependencies.target
          enabled: true
          name: swap-provision.service
        - contents: |
            [Unit]
            Description=Restrict swap for system slice
            ConditionFirstBoot=no

            [Service]
            Type=oneshot
            ExecStart=/bin/sh -c "sudo systemctl set-property --runtime system.slice
MemorySwapMax=0 IODeviceLatencyTargetSec=\"/ 50ms\""

            [Install]
```

```
      RequiredBy=kubelet-dependencies.target
    enabled: true
    name: cgroup-system-slice-config.service
```

To have enough swap space for the worst-case scenario, make sure to have at least as much swap space provisioned as overcommitted RAM. Calculate the amount of swap space to be provisioned on a node by using the following formula:

```
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
```

3. Create a privileged service account by running the following commands:

```
$ 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
```

4. Wait for the worker nodes to sync with the new configuration by running the following command:

```
$ oc wait mcp worker --for condition=Updated=True --timeout=-1s
```

5. Determine the pull URL for the wasp agent image by running the following command:

```
$ oc get csv -n openshift-cnv -l=operators.coreos.com/kubevirt-hyperconverged.openshift-cnv -ojson | jq '.items[0].spec.relatedImages[] | select(.name|test(".*wasp-agent.*")) | .image'
```

6. Deploy **wasp-agent** by creating a **DaemonSet** object as shown in the following example:

```
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:
    containers:
      - env:
          - name: SWAP_UTILIZATION_THRESHOLD_FACTOR
            value: "0.8"
          - name: MAX_AVERAGE_SWAP_IN_PAGES_PER_SECOND
            value: "1000000000"
          - name: MAX_AVERAGE_SWAP_OUT_PAGES_PER_SECOND
            value: "1000000000"
          - name: AVERAGE_WINDOW_SIZE_SECONDS
            value: "30"
          - name: VERBOSITY
            value: "1"
          - name: FSROOT
            value: /host
          - name: NODE_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
        image: >-
          quay.io/openshift-virtualization/wasp-agent:v4.20 (1)
        imagePullPolicy: Always
        name: wasp-agent
        resources:
          requests:
            cpu: 100m
            memory: 50M
        securityContext:
          privileged: true
        volumeMounts:
          - mountPath: /host
            name: host
          - mountPath: /rootfs
            name: rootfs
    hostPID: true
    hostUsers: true
    priorityClassName: system-node-critical
    serviceAccountName: wasp
    terminationGracePeriodSeconds: 5
    volumes:
      - hostPath:
          path: /
        name: host
      - hostPath:
          path: /
        name: rootfs
updateStrategy:
  type: RollingUpdate
```

```
    rollingUpdate:
      maxUnavailable: 10%
      maxSurge: 0
```

1 Replace the **image** value with the image URL from the previous step.

7. Deploy alerting rules by creating a **PrometheusRule** object. For example:

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  labels:
    tier: node
    wasp.io: ""
  name: wasp-rules
  namespace: wasp
spec:
  groups:
    - name: alerts.rules
      rules:
        - alert: NodeHighSwapActivity
          annotations:
            description: High swap activity detected at {{ $labels.instance }}. The rate
              of swap out and swap in exceeds 200 in both operations in the last minute.
              This could indicate memory pressure and may affect system performance.
            runbook_url: https://github.com/openshift-virtualization/wasp-
agent/tree/main/docs/runbooks/NodeHighSwapActivity.md
            summary: High swap activity detected at {{ $labels.instance }}.
          expr: rate(node_vmstat_pswpout[1m]) > 200 and rate(node_vmstat_pswpin[1m]) >
            200
          for: 1m
          labels:
            kubernetes_operator_component: kubevirt
            kubernetes_operator_part_of: kubevirt
            operator_health_impact: warning
            severity: warning
```

8. Add the **cluster-monitoring** label to the **wasp** namespace by running the following command:

```
$ oc label namespace wasp openshift.io/cluster-monitoring="true"
```

9. Enable memory overcommitment in OpenShift Virtualization by using the web console or the CLI.

- Web console

  1. In the OpenShift Container Platform web console, go to **Virtualization → Overview → Settings → General settings → Memory density**.
  2. Set **Enable memory density** to on.

- CLI

  - Configure your OpenShift Virtualization to enable higher memory density and set the overcommit rate:

```
$ oc -n openshift-cnv patch HyperConverged/kubevirt-hyperconverged --type='json' -
p='[ \
  { \
  "op": "replace", \
  "path": "/spec/higherWorkloadDensity/memoryOvercommitPercentage", \
  "value": 150 \
  } \
]'
```

Successful output

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

Verification

1. To verify the deployment of **wasp-agent**, run the following command:

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

   If the deployment is successful, the following message is displayed:

   Example output

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

2. To verify that swap is correctly provisioned, complete the following steps:

   a. View a list of worker nodes by running the following command:

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

   b. Select a node from the list and display its memory usage by running the following command:

      ```
      $ oc debug node/<selected_node> -- free -m (1)
      ```

      (1) Replace **<selected_node>** with the node name.

      If swap is provisioned, an amount greater than zero is displayed in the **Swap:** row.

      Table 5.1. Example output

	total	used	free	shared	buff/cache	available
Mem:	31846	23155	1044	6014	14483	8690
Swap:	8191	2337	5854			

3. Verify the OpenShift Virtualization memory overcommitment configuration by running the following command:

   -

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

Example output

```
{"memoryOvercommitPercentage":150}
```

The returned value must match the value you had previously configured.

5.5.2. Removing the wasp-agent component

If you no longer need memory overcommitment, you can remove the **wasp-agent** component and associated resources from your cluster.

Prerequisites

- You are logged in to the cluster with the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Remove the **wasp-agent** DaemonSet:

```
$ oc delete daemonset wasp-agent -n wasp
```

2. If deployed, remove the alerting rules:

```
$ oc delete prometheusrule wasp-rules -n wasp
```

3. Optionally, delete the **wasp** namespace if no other resources depend on it:

```
$ oc delete namespace wasp
```

4. Revert the memory overcommitment configuration:

```
$ oc -n openshift-cnv patch HyperConverged/kubevirt-hyperconverged \
  --type='json' \
  -p='[{"op": "remove", "path": "/spec/higherWorkloadDensity"}]'
```

5. Delete the **MachineConfig** that provisions swap memory:

```
$ oc delete machineconfig 90-worker-swap
```

6. Delete the associated **KubeletConfig**:

```
$ oc delete kubeletconfig custom-config
```

7. Wait for the worker nodes to reconcile:

```
$ oc wait mcp worker --for condition=Updated=True --timeout=-1s
```

Verification

- Confirm that the **wasp-agent** DaemonSet is removed:

  ```
  $ oc get daemonset -n wasp
  ```

  No **wasp-agent** should be listed.

- Confirm that swap is no longer enabled on a node:

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

  Ensure that the **Swap:** row shows **0** or that no swap space shows as provisioned.

5.5.3. Pod eviction conditions used by wasp-agent

The wasp agent manages pod eviction when the system is heavily loaded and nodes are at risk. Eviction is triggered if one of the following conditions is met:

High swap I/O traffic

This condition is met when swap-related I/O traffic is excessively high.

Condition

```
averageSwapInPerSecond > maxAverageSwapInPagesPerSecond
&&
averageSwapOutPerSecond > maxAverageSwapOutPagesPerSecond
```

By default, **maxAverageSwapInPagesPerSecond** and **maxAverageSwapOutPagesPerSecond** are set to 1000 pages. The default time interval for calculating the average is 30 seconds.

High swap utilization

This condition is met when swap utilization is excessively high, causing the current virtual memory usage to exceed the factored threshold. The **NODE_SWAP_SPACE** setting in your **MachineConfig** object can impact this condition.

Condition

```
nodeWorkingSet + nodeSwapUsage < totalNodeMemory + totalSwapMemory × thresholdFactor
```

5.5.3.1. Environment variables

You can use the following environment variables to adjust the values used to calculate eviction conditions:

Environment variable	Function
MAX_AVERAGE_SWAP_IN_PAGES_PER_SECOND	Sets the value of **maxAverageSwapInPagesPerSecond**.

MAX_AVERAGE_SWAP_OUT_PAGES_PER_SECOND	Sets the value of **maxAverageSwapOutPagesPerSecond**.
SWAP_UTILIZATION_THRESHOLD_FACTOR	Sets the **thresholdFactor** value used to calculate high swap utilization.
AVERAGE_WINDOW_SIZE_SECONDS	Sets the time interval for calculating the average swap usage.

5.6. CONFIGURING CERTIFICATE ROTATION

Configure certificate rotation parameters to replace existing certificates.

5.6.1. Configuring certificate rotation

You can do this during OpenShift Virtualization installation in the web console or after installation in the **HyperConverged** custom resource (CR).

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Open the **HyperConverged** CR by running the following command:

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

2. Edit the **spec.certConfig** fields as shown in the following example. To avoid overloading the system, ensure that all values are greater than or equal to 10 minutes. Express all values as strings that comply with the golang **ParseDuration** format.

```
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) The value of **ca.renewBefore** must be less than or equal to the value of **ca.duration**.

(2) The value of **server.duration** must be less than or equal to the value of **ca.duration**.

3 The value of **server.renewBefore** must be less than or equal to the value of **server.duration**.

3. Apply the YAML file to your cluster.

5.6.2. Troubleshooting certificate rotation parameters

Deleting one or more **certConfig** values causes them to revert to the default values, unless the default values conflict with one of the following conditions:

- The value of **ca.renewBefore** must be less than or equal to the value of **ca.duration**.
- The value of **server.duration** must be less than or equal to the value of **ca.duration**.
- The value of **server.renewBefore** must be less than or equal to the value of **server.duration**.

If the default values conflict with these conditions, you will receive an error.

If you remove the **server.duration** value in the following example, the default value of **24h0m0s** is greater than the value of **ca.duration**, conflicting with the specified conditions.

Example

```
certConfig:
  ca:
    duration: 4h0m0s
    renewBefore: 1h0m0s
  server:
    duration: 4h0m0s
    renewBefore: 4h0m0s
```

This results in the following error message:

```
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
```

The error message only mentions the first conflict. Review all certConfig values before you proceed.

CHAPTER 6. VIRTUALIZATION WITH IBM FUSION ACCESS FOR SAN

6.1. IBM FUSION ACCESS FOR SAN OVERVIEW

6.1.1. About IBM Fusion Access for SAN

IBM Fusion Access for SAN is a solution that provides a scalable clustered file system for enterprise storage, primarily designed to offer access to consolidated, block-level data storage. It presents storage devices, such as disk arrays, to the operating system as if they were direct-attached storage.

This solution is particularly geared towards enterprise storage for Red Hat OpenShift Virtualization and leverages existing Storage Area Network (SAN) infrastructure. A SAN is a dedicated network of storage devices that is typically not accessible through the local area network (LAN).

6.1.1.1. Why use Fusion Access for SAN?

Easy user experience

Fusion Access for SAN features a wizard-driven user interface (UI) for installing and configuring storage clusters, file systems, and storage classes, to simplify the setup process.

Leverage existing infrastructure

Organizations can leverage their existing SAN investments, including Fibre Channel (FC) and iSCSI technologies, as they transition to or expand with OpenShift Virtualization.

Scalability

The storage cluster is designed to scale with OpenShift Container Platform clusters and virtual machine (VM) workloads. It can support up to approximately 3000 VMs on 6 bare-metal hosts, with possibilities for further scaling by adding more file systems or using specific storage class parameters.

Consolidated and shared storage

SANs enable multiple servers to access a large, shared data storage capacity. This architecture facilitates automatic data backup and continuous monitoring of the storage and backup processes.

High-speed data transfer

By using a dedicated high-speed network for storage, Fusion Access for SAN overcomes the data transfer bottlenecks that can occur over a traditional LAN, especially for large volumes of data.

File-level access

Although a SAN primarily operates at the block level, file systems built on top of SAN storage can provide file-level access through shared-disk file systems.

Centralized management

The underlying SAN software manages servers, storage devices, and the network to ensure that data moves directly between storage devices with minimal server intervention. It also supports centralized management and configuration of SAN components like Logical Unit Numbers (LUNs).

6.1.2. Prerequisites and Limitations for Fusion Access for SAN

6.1.2.1. Prerequisites

Installing and configuring Fusion Access for SAN require the following prerequisites:

- Bare-metal worker nodes with attached SAN storage.
- A working container registry enabled.
- All worker nodes must connect to the same LUNs.
  A shared LUN is a shared disk that is accessed by all worker nodes simultaneously.
- A Kubernetes pull secret.

6.1.2.2. Limitations

- Limitations for Fusion Access for SAN rely on the IBM Storage Scale container native limitations and can be found in the documentation for IBM Storage Scale container native .
- Hosted control planes (HCP) clusters are not supported.

6.2. INSTALLING AND CONFIGURING IBM FUSION ACCESS FOR SAN

To use Red Hat OpenShift Virtualization with IBM Fusion Access for SAN, you must first install the Fusion Access for SAN Operator.

Then you must create a Kubernetes pull secret and create the FusionAccess custom resource (CR).

Finally, follow the Red Hat OpenShift Container Platform web console wizard to configure the storage cluster, local disk, and file systems.

6.2.1. Installing the Fusion Access for SAN Operator

Install the Fusion Access for SAN Operator from the software catalog in the OpenShift Container Platform web console.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have a working container registry enabled.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Ecosystem → Software Catalog**.
2. In the **Filter by keyword** field, type **Fusion Access for SAN**.
3. Select the **Fusion Access for SAN**tile and click **Install**.
4. On the **Install Operator** page, keep the default selections for **Update Channel**, **Version**, and **Installation mode**.
5. Verify that **Operator recommended Namespace** is selected for **Installed Namespace**.
   This installs the Operator in the **ibm-fusion-access** namespace. If this namespace does not yet exist, it is automatically created.

WARNING

You must install the Fusion Access for SAN Operator in the **ibm-fusion-access** namespace. Installation in any other namespace is not supported.

6. Verify that the **Automatic** default is selected for **Update Approval**.
   This enables automatic updates when a new z-stream release is available.
7. Click **Install**.
   This installs the Operator.

Verification

1. Navigate to **Ecosystem → Installed Operators**.
2. Verify that the Fusion Access for SAN Operator is displayed.

6.2.2. Creating a Kubernetes pull secret

After installing the Fusion Access for SAN Operator, you must create a Kubernetes secret object to hold the IBM entitlement key for pulling the required container images from the IBM container registry.

Prerequisites

- You installed the **oc** CLI.
- You have access to the cluster as a user with the **cluster-admin** role.
- You installed the Fusion Access for SAN Operator and created the **ibm-fusion-access** namespace in the process.

Procedure

1. Log in to the IBM Container software library with your Fusion Access for SAN **IBMid** and **password**.
2. In the **IBM Container software library**, get the entitlement key:
   a. If you do not have an entitlement key yet, click **Get entitlement key** or **Add new key**, and then click **Copy**.
   b. If you already have an entitlement key, click **Copy**.
3. Save the entitlement key in a safe place.
4. Create the secret object by running the **oc create** command:

```
$ oc create secret -n ibm-fusion-access generic fusion-pullsecret \
--from-literal=ibm-entitlement-key=<ibm-entitlement-key> (1)
```

1 This is the entitlement key you copied in step 2 from the **IBM Container software library**.

Verification

1. In the OpenShift Container Platform web console, navigate to **Workloads → Secrets**.
2. Find the **fusion-pullsecret** in the list.

6.2.3. Creating the FusionAccess CR

After installing the Fusion Access for SAN Operator and creating a Kubernetes pull secret, you must create the **FusionAccess** custom resource (CR).

Creating the **FusionAccess** CR triggers the installation of the correct version of IBM Storage Scale and detects worker nodes with shared LUNs.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You installed the Fusion Access for SAN Operator.
- You created a Kubernetes pull secret.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Ecosystem → Installed Operators**.
2. Click on the Fusion Access for SAN Operator you installed.
3. In the **Fusion Access for SAN** page, select the **Fusion Access** tab.
4. Click **Create FusionAccess**.
5. On the **Create FusionAccess** page, enter the object **Name**.
6. Optional: You can choose to add **Labels** if they are relevant.
7. Select the **IBM Storage Scale Version** from the drop-down list.
8. Click **Create**.

Verification

- In the **Fusion Access for SAN** Operator page, in the **Fusion Access** tab, verify that the created **FusionAccess** CR appears with the status **Ready**.

6.2.4. Creating a storage cluster with Fusion Access for SAN

Once you have installed the Fusion Access for SAN Operator, you can create a storage cluster with shared storage nodes.

The wizard for creating the storage cluster in the OpenShift Container Platform web console provides easy-to-follow steps and lists the relevant worker nodes with shared disks.

Prerequisites

- You have bare-metal worker nodes with visible and attached shared LUNs.
  A shared LUN is a shared disk that is accessed by all workers simultaneously.
- You installed the Fusion Access for SAN Operator.
- You created the **FusionAccess** custom resource (CR) in the **ibm-fusion-access** namespace.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Storage → Fusion Access for SAN**.
2. Click **Create storage cluster**.
3. Select the worker nodes that have shared LUNs.

NOTE

You can only select worker nodes with a minimum of 20 GB of RAM from the list.

4. Click **Create storage cluster**.
   The page reloads, opening the Fusion Access for SAN page for the new storage cluster.

6.2.5. Creating a file system with Fusion Access for SAN

You need to create a file system to represent your required storage.

The file system is based on the storage available in the worker nodes you selected when creating the storage cluster.

Prerequisites

- You created a Fusion Access for SAN storage cluster.

Procedure

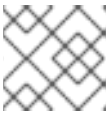
1. In the OpenShift Container Platform web console, navigate to **Storage → Fusion Access for SAN**.
2. In the **File systems** tab, click **Create file system**.
3. Enter a **Name** for the new file system.
4. Select the LUNs that you want to use as the storage volumes for your file system.
5. Click **Create file system**.
   The **Fusion Access for SAN** page reloads, and the new file system appears in the **File systems** tab.

Next steps

Repeat this procedure for each file system that you want to create.

Verification

1. Watch the **Status** of the file system in the **File systems** tab until it is marked as **Healthy**.



NOTE

This may take several minutes.

2. Click on the **StorageClass** for the file system.
3. In the **YAML** tab, verify the following:
   a. The value in the **name** field is the name of the file system you created.
   b. The value in the **provisioner** field is **spectrumscale.csi.ibm.com**.
   c. The value in the **volBackendFs** field matches the name of the file system you created.

```
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: filesystem1
  uid: eb410309-a043-a89b-9bb05483872a
  resourceVersion: '87746'
  creationTimestamp: '2025-05-14T12:30:08Z'
  managedFields:
provisioner: spectrumscale.csi.ibm.com
parameters:
  volBackendFs: filesystem1
reclaimPolicy: Delete
allowVolumeExpansion: true
volumeBindingMode: Immediate
```

6.2.6. Next steps

Once you create a storage cluster with file systems, you can create a virtual machine (VM) on the storage cluster.

Create a VM from an instance type or template and select a storage class that corresponds to one of the file systems you created as the storage type.

- Creating virtual machines from instance types.
- Creating virtual machines from templates.

6.2.7. IBM Fusion Access for SAN release updates

Release updates for IBM Fusion Access for SAN, including new features, bug fixes, and known issues.

6.2.7.1. New and changed features

Image registry requirements for kernel module management

IBM Fusion Access for SAN uses the OpenShift Container Platform image registry to manage the kernel module. Do not configure the registry to use **emptyDir** storage because it provides only temporary storage and is not suitable for production use. Configure IBM Fusion Access for SAN to use a different image registry by creating a config map and secret after installing the Operator and before creating the **FusionAccess** CR. (OCPNAS-213)

6.2.7.2. Bug fixes

Filesystem creation button stays disabled until daemons are ready

The IBM Fusion Access for SAN Operator was updated to check the readiness of filesystem daemons before allowing a filesystem to be created. The **Create file system** button in the web console now stays disabled with a tooltip explaining the condition until the environment is ready. This change prevents filesystems from appearing stuck during creation. (OCPNAS-184)

Filesystems cannot be deleted from the user interface

The OpenShift Container Platform web console does not support deleting filesystems. To delete a filesystem, use the OpenShift CLI (**oc**). (OCPNAS-217)

6.2.7.3. Known issues

Filesystem creation might fail during core pod deletion

Filesystem creation might fail if core pods are deleted at the same time. The filesystem might be partially created on the LUN, which results in the following persistent error:

> Disk <ID> may still belong to an active file system

No workaround is available. Contact IBM Support for assistance. (OCPNAS-233)

CHAPTER 7. UPDATING

7.1. UPDATING OPENSHIFT VIRTUALIZATION

Learn how to keep OpenShift Virtualization updated and compatible with OpenShift Container Platform.

7.1.1. About updating OpenShift Virtualization

When you install OpenShift Virtualization, you select an update channel and an approval strategy. The update channel determines the versions that OpenShift Virtualization will be updated to. The approval strategy setting determines whether updates occur automatically or require manual approval. Both settings can impact supportability.

7.1.1.1. Recommended settings

To maintain a supportable environment, use the following settings:

- Update channel: **stable**
- Approval strategy: **Automatic**

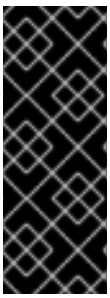
With these settings, the update process automatically starts when a new version of the Operator is available in the **stable** channel. This ensures that your OpenShift Virtualization and OpenShift Container Platform versions remain compatible, and that your version of OpenShift Virtualization is suitable for production environments.

NOTE

Each minor version of OpenShift Virtualization is supported only if you run the corresponding OpenShift Container Platform version. For example, you must run OpenShift Virtualization 4.20 on OpenShift Container Platform 4.20.

7.1.1.2. What to expect

- The amount of time an update takes to complete depends on your network connection. Most automatic updates complete within fifteen minutes.
- Updating OpenShift Virtualization does not interrupt network connections.
- Data volumes and their associated persistent volume claims are preserved during an update.



IMPORTANT

If you have virtual machines running that use hostpath provisioner storage, they cannot be live migrated and might block an OpenShift Container Platform cluster update.

As a workaround, you can reconfigure the virtual machines so that they can be powered off automatically during a cluster update. Set the **evictionStrategy** field to **None** and the **runStrategy** field to **Always**.

7.1.1.3. How updates work

- Operator Lifecycle Manager (OLM) manages the lifecycle of the OpenShift Virtualization Operator. The Marketplace Operator, which is deployed during OpenShift Container Platform installation, makes external Operators available to your cluster.
- OLM provides z-stream and minor version updates for OpenShift Virtualization. Minor version updates become available when you update OpenShift Container Platform to the next minor version. You cannot update OpenShift Virtualization to the next minor version without first updating OpenShift Container Platform.

7.1.1.4. RHEL 9 compatibility

OpenShift Virtualization 4.20 is based on Red Hat Enterprise Linux (RHEL) 9. You can update to OpenShift Virtualization 4.20 from a version that was based on RHEL 8 by following the standard OpenShift Virtualization update procedure. No additional steps are required.

As in previous versions, you can perform the update without disrupting running workloads. OpenShift Virtualization 4.20 supports live migration from RHEL 8 nodes to RHEL 9 nodes.

7.1.1.4.1. RHEL 9 machine type

All VM templates that are included with OpenShift Virtualization now use the RHEL 9 machine type by default: **machineType: pc-q35-rhel9.<y>.0**, where **<y>** is a single digit corresponding to the latest minor version of RHEL 9. For example, the value **pc-q35-rhel9.2.0** is used for RHEL 9.2.

Updating OpenShift Virtualization does not change the **machineType** value of any existing VMs. These VMs continue to function as they did before the update. You can optionally change a VM's machine type so that it can benefit from RHEL 9 improvements.

IMPORTANT

Before you change a VM's **machineType** value, you must shut down the VM.

7.1.2. Monitoring update status

To monitor the status of a OpenShift Virtualization Operator update, watch the cluster service version (CSV) **PHASE**. You can also monitor the CSV conditions in the web console or by running the command provided here.

NOTE

The **PHASE** and conditions values are approximations that are based on available information.

Prerequisites

- Log in to the cluster as a user with the **cluster-admin** role.
- Install the OpenShift CLI (**oc**).

Procedure

1. Run the following command:

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

2. Review the output, checking the **PHASE** field. For example:

Example output

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

3. Optional: Monitor the aggregated status of all OpenShift Virtualization component conditions by running the following command:

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

A successful upgrade results in the following output:

Example output

```
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
```

7.1.3. VM workload updates

When you update OpenShift Virtualization, virtual machine workloads, including **libvirt**, **virt-launcher**, and **qemu**, update automatically if they support live migration.

NOTE

Each virtual machine has a **virt-launcher** pod that runs the virtual machine instance (VMI). The **virt-launcher** pod runs an instance of **libvirt**, which is used to manage the virtual machine (VM) process.

You can configure how workloads are updated by editing the **spec.workloadUpdateStrategy** stanza of the **HyperConverged** custom resource (CR). There are two available workload update methods: **LiveMigrate** and **Evict**.

Because the **Evict** method shuts down VMI pods, only the **LiveMigrate** update strategy is enabled by default.

When **LiveMigrate** is the only update strategy enabled:

- VMIs that support live migration are migrated during the update process. The VM guest moves into a new pod with the updated components enabled.
- VMIs that do not support live migration are not disrupted or updated.
    - If a VMI has the **LiveMigrate** eviction strategy but does not support live migration, it is not updated.

If you enable both **LiveMigrate** and **Evict**:

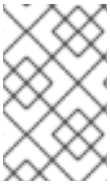
- VMIs that support live migration use the **LiveMigrate** update strategy.
- VMIs that do not support live migration use the **Evict** update strategy. If a VMI is controlled by a **VirtualMachine** object that has **runStrategy: Always** set, a new VMI is created in a new pod with updated components.

Migration attempts and timeouts
When updating workloads, live migration fails if a pod is in the **Pending** state for the following periods:

5 minutes
If the pod is pending because it is **Unschedulable**.

15 minutes
If the pod is stuck in the pending state for any reason.

When a VMI fails to migrate, the **virt-controller** tries to migrate it again. It repeats this process until all migratable VMIs are running on new **virt-launcher** pods. If a VMI is improperly configured, however, these attempts can repeat indefinitely.



NOTE

Each attempt corresponds to a migration object. Only the five most recent attempts are held in a buffer. This prevents migration objects from accumulating on the system while retaining information for debugging.

7.1.3.1. Configuring workload update methods

You can configure workload update methods by editing the **HyperConverged** custom resource (CR).

Prerequisites

- To use live migration as an update method, you must first enable live migration in the cluster.

NOTE

If a **VirtualMachineInstance** CR contains **evictionStrategy: LiveMigrate** and the virtual machine instance (VMI) does not support live migration, the VMI will not update.

- You have installed the OpenShift CLI (**oc**).

Procedure

1. To open the **HyperConverged** CR in your default editor, run the following command:

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

2. Edit the **workloadUpdateStrategy** stanza of the **HyperConverged** CR. For example:

```
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. The methods that can be used to perform automated workload updates. The available values are **LiveMigrate** and **Evict**. If you enable both options as shown in this example, updates use **LiveMigrate** for VMIs that support live migration and **Evict** for any VMIs that do not support live migration. To disable automatic workload updates, you can either remove the **workloadUpdateStrategy** stanza or set **workloadUpdateMethods: []** to leave the array empty.

2. The least disruptive update method. VMIs that support live migration are updated by migrating the virtual machine (VM) guest into a new pod with the updated components enabled. If **LiveMigrate** is the only workload update method listed, VMIs that do not support live migration are not disrupted or updated.

3. A disruptive method that shuts down VMI pods during upgrade. **Evict** is the only update method available if live migration is not enabled in the cluster. If a VMI is controlled by a **VirtualMachine** object that has **runStrategy: Always** configured, a new VMI is created in a new pod with updated components.

4. The number of VMIs that can be forced to be updated at a time by using the **Evict** method. This does not apply to the **LiveMigrate** method.

5. The interval to wait before evicting the next batch of workloads. This does not apply to the **LiveMigrate** method.

NOTE

You can configure live migration limits and timeouts by editing the **spec.liveMigrationConfig** stanza of the **HyperConverged** CR.

3. To apply your changes, save and exit the editor.

7.1.3.2. Viewing outdated VM workloads

You can view a list of outdated virtual machine (VM) workloads by using the CLI.

NOTE

If there are outdated virtualization pods in your cluster, the **OutdatedVirtualMachineInstanceWorkloads** alert fires.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- To view a list of outdated virtual machine instances (VMIs), run the following command:

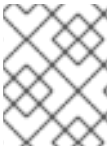
```
$ oc get vmi -l kubevirt.io/outdatedLauncherImage --all-namespaces
```

NOTE

To ensure that VMIs update automatically, configure workload updates.

7.1.4. Control Plane Only updates

Every even-numbered minor version of OpenShift Container Platform is an Extended Update Support (EUS) version. However, Kubernetes design mandates serial minor version updates, so you cannot directly update from one EUS version to the next. An EUS-to-EUS update starts with updating OpenShift Virtualization to the latest z-stream of the next odd-numbered minor version. Next, update OpenShift Container Platform to the target EUS version. When the OpenShift Container Platform update succeeds, the corresponding update for OpenShift Virtualization becomes available. You can now update OpenShift Virtualization to the target EUS version.



NOTE

You can directly update OpenShift Virtualization to the latest z-stream release of your current minor version without applying each intermediate z-stream update.

For more information about EUS versions, see the Red Hat OpenShift Container Platform Life Cycle Policy.

7.1.4.1. Prerequisites

Before beginning a Control Plane Only update, you must:

- Pause worker nodes' machine config pools before you start a Control Plane Only update so that the workers are not rebooted twice.
- Disable automatic workload updates before you begin the update process. This is to prevent OpenShift Virtualization from migrating or evicting your virtual machines (VMs) until you update to your target EUS version.

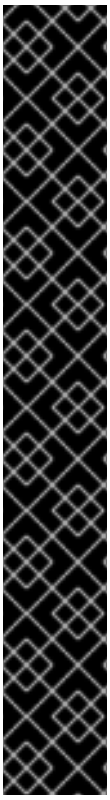
NOTE

By default, OpenShift Virtualization automatically updates workloads, such as the **virt-launcher** pod, when you update the OpenShift Virtualization Operator. You can configure this behavior in the **spec.workloadUpdateStrategy** stanza of the **HyperConverged** custom resource.

Learn more about Performing a Control Plane Only update .

7.1.4.2. Preventing workload updates during a Control Plane Only update

When you update from one Extended Update Support (EUS) version to the next, you must manually disable automatic workload updates to prevent OpenShift Virtualization from migrating or evicting workloads during the update process.



IMPORTANT

In OpenShift Container Platform 4.16, the underlying Red Hat Enterprise Linux CoreOS (RHCOS) upgraded to version 9.4 of Red Hat Enterprise Linux (RHEL). To operate correctly, all **virt-launcher** pods in the cluster need to use the same version of RHEL.

After upgrading to OpenShift Container Platform 4.16 from an earlier version, re-enable workload updates in OpenShift Virtualization to allow **virt-launcher** pods to update. Before upgrading to the next OpenShift Container Platform version, verify that all VMIs use up-to-date workloads:

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

If the previous command returns a value larger than **0**, list all VMIs with outdated **virt-launcher** pods and start live migration to update them to a new version:

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

For the list of supported OpenShift Container Platform releases and the RHEL versions they use, see RHEL Versions Utilized by RHCOS and OpenShift Container Platform .

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You are running an EUS version of OpenShift Container Platform and want to update to the next EUS version. You have not yet updated to the odd-numbered version in between.
- You read "Preparing to perform a Control Plane Only update" and learned the caveats and requirements that pertain to your OpenShift Container Platform cluster.
- You paused the worker nodes' machine config pools as directed by the OpenShift Container Platform documentation.
- It is recommended that you use the default **Automatic** approval strategy. If you use the **Manual** approval strategy, you must approve all pending updates in the web console. For more details, refer to the "Manually approving a pending Operator update" section.

Procedure

1. Run the following command and record the **workloadUpdateMethods** configuration:

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

2. Turn off all workload update methods by running the following command:

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

Example output

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

3. Ensure that the **HyperConverged** Operator is **Upgradeable** before you continue. Enter the following command and monitor the output:

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

Example 7.1. Example output


```
[
  {
    "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 The OpenShift Virtualization Operator has the **Upgradeable** status.

4. Manually update your cluster from the source EUS version to the next minor version of OpenShift Container Platform:

```
$ oc adm upgrade
```

Verification

- Check the current version by running the following command:

```
$ oc get clusterversion
```

NOTE

Updating OpenShift Container Platform to the next version is a prerequisite for updating OpenShift Virtualization. For more details, refer to the "Updating clusters" section of the OpenShift Container Platform documentation.

5. Update OpenShift Virtualization.

- With the default **Automatic** approval strategy, OpenShift Virtualization automatically updates to the corresponding version after you update OpenShift Container Platform.
- If you use the **Manual** approval strategy, approve the pending updates by using the web console.

6. Monitor the OpenShift Virtualization update by running the following command:

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

7. Confirm that OpenShift Virtualization successfully updated to the latest z-stream release of the non-EUS version by running the following command:

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

Example output

```
[
  {
    "name": "operator",
    "version": "4.20.0"
  }
]
```

8. Wait until the **HyperConverged** Operator has the **Upgradeable** status before you perform the next update. Enter the following command and monitor the output:

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

9. Update OpenShift Container Platform to the target EUS version.
10. Confirm that the update succeeded by checking the cluster version:

```
$ oc get clusterversion
```

11. Update OpenShift Virtualization to the target EUS version.
    - With the default **Automatic** approval strategy, OpenShift Virtualization automatically updates to the corresponding version after you update OpenShift Container Platform.
    - If you use the **Manual** approval strategy, approve the pending updates by using the web console.
12. Monitor the OpenShift Virtualization update by running the following command:

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

    The update completes when the **VERSION** field matches the target EUS version and the **PHASE** field reads **Succeeded**.

13. Restore the **workloadUpdateMethods** configuration that you recorded from step 1 with the following command:

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

Example output

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

Verification

- Check the status of VM migration by running the following command:

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

Next steps

- Unpause the machine config pools for each compute node.

7.1.5. Advanced options

The **stable** release channel and the **Automatic** approval strategy are recommended for most OpenShift Virtualization installations. Use other settings only if you understand the risks.

7.1.5.1. Changing update settings

You can change the update channel and approval strategy for your OpenShift Virtualization Operator subscription by using the web console.

Prerequisites

- You have installed the OpenShift Virtualization Operator.
- You have administrator permissions.

Procedure

1. Click **Ecosystem → Installed Operators**.
2. Select **OpenShift Virtualization** from the list.
3. Click the **Subscription** tab.
4. In the **Subscription details** section, click the setting that you want to change. For example, to change the approval strategy from **Manual** to **Automatic**, click **Manual**.
5. In the window that opens, select the new update channel or approval strategy.
6. Click **Save**.

7.1.5.2. Manual approval strategy

If you use the **Manual** approval strategy, you must manually approve every pending update. If OpenShift Container Platform and OpenShift Virtualization updates are out of sync, your cluster becomes unsupported. To avoid risking the supportability and functionality of your cluster, use the **Automatic** approval strategy.

If you must use the **Manual** approval strategy, maintain a supportable cluster by approving pending Operator updates as soon as they become available.

7.1.5.3. Manually approving a pending Operator update

If an installed Operator has the approval strategy in its subscription set to **Manual**, when new updates are released in its current update channel, the update must be manually approved before installation can begin.

Prerequisites

- An Operator previously installed using Operator Lifecycle Manager (OLM).

Procedure

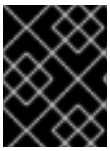
1. In the OpenShift Container Platform web console, navigate to **Ecosystem → Installed Operators**.
2. Operators that have a pending update display a status with **Upgrade available**. Click the name of the Operator you want to update.
3. Click the **Subscription** tab. Any updates requiring approval are displayed next to **Upgrade status**. For example, it might display **1 requires approval**.
4. Click **1 requires approval**, then click **Preview Install Plan**.
5. Review the resources that are listed as available for update. When satisfied, click **Approve**.
6. Navigate back to the **Ecosystem → Installed Operators** page to monitor the progress of the update. When complete, the status changes to **Succeeded** and **Up to date**.

7.1.6. Early access releases

You can gain access to builds in development by subscribing to the **candidate** update channel for your version of OpenShift Virtualization. These releases have not been fully tested by Red Hat and are not supported, but you can use them on non-production clusters to test capabilities and bug fixes being developed for that version.

The **stable** channel, which matches the underlying OpenShift Container Platform version and is fully tested, is suitable for production systems. You can switch between the **stable** and **candidate** channels in Operator Hub. However, updating from a **candidate** channel release to a **stable** channel release is not tested by Red Hat.

Some candidate releases are promoted to the **stable** channel. However, releases present only in **candidate** channels might not contain all features that will be made generally available (GA), and some features in candidate builds might be removed before GA. Additionally, candidate releases might not offer update paths to later GA releases.



IMPORTANT

The candidate channel is only suitable for testing purposes where destroying and recreating a cluster is acceptable.

7.1.7. Additional resources

- Performing a Control Plane Only update
- What are Operators?
- Operator Lifecycle Manager concepts and resources
- Cluster service versions (CSVs)
- About live migration
- Configuring eviction strategies
- Configuring live migration limits and timeouts

CHAPTER 8. CREATING A VIRTUAL MACHINE

8.1. CREATING VIRTUAL MACHINES FROM INSTANCE TYPES

You can simplify virtual machine (VM) creation by using instance types, whether you use the OpenShift Container Platform web console or the CLI to create VMs.

8.1.1. About instance types

An instance type is a reusable object where you can define resources and characteristics to apply to new VMs. You can define custom instance types or use the variety that are included when you install OpenShift Virtualization.

To create a new instance type, you must first create a manifest, either manually or by using the **virtctl** CLI tool. You then create the instance type object by applying the manifest to your cluster.

OpenShift Virtualization provides two CRDs for configuring instance types:

- A namespaced object: **VirtualMachineInstancetype**
- A cluster-wide object: **VirtualMachineClusterInstancetype**

These objects use the same **VirtualMachineInstancetypeSpec**.

8.1.1.1. Required attributes

When you configure an instance type, you must define the **cpu** and **memory** attributes. Other attributes are optional.

NOTE

When you create a VM from an instance type, you cannot override any parameters defined in the instance type.

Because instance types require defined CPU and memory attributes, OpenShift Virtualization always rejects additional requests for these resources when creating a VM from an instance type.

You can manually create an instance type manifest. For example:

Example YAML file with required fields

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

(1) Required. Specifies the number of vCPUs to allocate to the guest.

2 Required. Specifies an amount of memory to allocate to the guest.

You can create an instance type manifest by using the **virtctl** CLI utility. For example:

Example virtctl command with required fields

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

where:

--cpu <value>
Specifies the number of vCPUs to allocate to the guest. Required.

--memory <value>
Specifies an amount of memory to allocate to the guest. Required.

TIP

You can immediately create the object from the new manifest by running the following command:

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

8.1.1.2. Optional attributes

In addition to the required **cpu** and **memory** attributes, you can include the following optional attributes in the **VirtualMachineInstancetypeSpec**:

annotations
List annotations to apply to the VM.

gpus
List vGPUs for passthrough.

hostDevices
List host devices for passthrough.

ioThreadsPolicy
Define an IO threads policy for managing dedicated disk access.

launchSecurity
Configure Secure Encrypted Virtualization (SEV).

nodeSelector
Specify node selectors to control the nodes where this VM is scheduled.

schedulerName
Define a custom scheduler to use for this VM instead of the default scheduler.

8.1.1.3. Controller revisions

When you create a VM by using an instance type, a **ControllerRevision** object retains an immutable snapshot of the instance type object. This snapshot locks in resource-related characteristics defined in the instance type object, such as the required guest CPU and memory. The VM status also contains a reference to the **ControllerRevision** object.

This snapshot is essential for versioning, and ensures that the VM instance created when starting a VM does not change if the underlying instance type object is updated while the VM is running.

8.1.2. Pre-defined instance types

OpenShift Virtualization includes a set of pre-defined instance types called **common-instancetypes**. Some are specialized for specific workloads and others are workload-agnostic.

These instance type resources are named according to their series, version, and size. The size value follows the **.** delimiter and ranges from **nano** to **8xlarge**.

Table 8.1. common-instancetypes series comparison

Use case	Series	Characteristics	vCPU to memory ratio	Example resource
Network	N	• Hugepages • Dedicated CPU • Isolated emulator threads • Requires nodes capable of running DPDK workloads	1:2	**n1.medium** • 4 vCPUs • 4GiB Memory
Overcommitted	O	• Overcommitted memory • Burstable CPU performance	1:4	**o1.small** • 1 vCPU • 2GiB Memory
Compute Exclusive	CX	• Hugepages • Dedicated CPU • Isolated emulator threads • vNUMA	1:2	**cx1.2xlarge** • 8 vCPUs • 16GiB Memory

Use case	Series	Characteristics	vCPU to memory ratio	Example resource
General Purpose	U	• Burstable CPU performance	1:4	**u1.medium** • 1 vCPU • 4GiB Memory
Memory Intensive	M	• Hugepages • Burstable CPU performance	1:8	**m1.large** • 2 vCPUs • 16GiB Memory

8.1.3. Specifying an instance type or preference

You can specify an instance type, a preference, or both to define a set of workload sizing and runtime characteristics for reuse across multiple VMs.

8.1.3.1. Using flags to specify instance types and preferences

Specify instance types and preferences by using flags.

Prerequisites

- You must have an instance type, preference, or both on the cluster.

Procedure

1. To specify an instance type when creating a VM, use the **--instancetype** flag. To specify a preference, use the **--preference** flag. The following example includes both flags:

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

2. Optional: To specify a namespaced instance type or preference, include the **kind** in the value passed to the **--instancetype** or **--preference** flag command. The namespaced instance type or preference must be in the same namespace you are creating the VM in. The following example includes flags for a namespaced instance type and a namespaced preference:

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

8.1.3.2. Inferring an instance type or preference

Inferring instance types, preferences, or both is enabled by default, and the **inferFromVolumeFailure** policy of the **inferFromVolume** attribute is set to **Ignore**. When inferring from the boot volume, errors are ignored, and the VM is created with the instance type and preference left unset.

However, when flags are applied, the **inferFromVolumeFailure** policy defaults to **Reject**. When inferring from the boot volume, errors result in the rejection of the creation of that VM.

You can use the **--infer-instancetype** and **--infer-preference** flags to infer which instance type, preference, or both to use to define the workload sizing and runtime characteristics of a VM.

Prerequisites

- You have installed the **virtctl** tool.

Procedure

- To explicitly infer instance types from the volume used to boot the VM, use the **--infer-instancetype** flag. To explicitly infer preferences, use the **--infer-preference** flag. The following command includes both flags:

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

- To infer an instance type or preference from a volume other than the volume used to boot the VM, use the **--infer-instancetype-from** and **--infer-preference-from** flags to specify any of the virtual machine's volumes. In the example below, the virtual machine boots from **volume-a** but infers the instancetype and preference from **volume-b**.

```
$ virtctl create vm \
  --volume-import=type:pvc,src:my-ns/my-pvc-a,name:volume-a \
  --volume-import=type:pvc,src:my-ns/my-pvc-b,name:volume-b \
  --infer-instancetype-from volume-b \
  --infer-preference-from volume-b
```

8.1.3.3. Setting the inferFromVolume labels

Use the following labels on your PVC, data source, or data volume to instruct the inference mechanism which instance type, preference, or both to use when trying to boot from a volume.

- A cluster-wide instance type: **instancetype.kubevirt.io/default-instancetype** label.
- A namespaced instance type: **instancetype.kubevirt.io/default-instancetype-kind** label. Defaults to the **VirtualMachineClusterInstancetype** label if left empty.
- A cluster-wide preference: **instancetype.kubevirt.io/default-preference** label.
- A namespaced preference: **instancetype.kubevirt.io/default-preference-kind** label. Defaults to **VirtualMachineClusterPreference** label, if left empty.

Prerequisites

- You must have an instance type, preference, or both on the cluster.
- You have installed the OpenShift CLI (**oc**).

Procedure

- To apply a label to a data source, use **oc label**. The following command applies a label that points to a cluster-wide instance type:

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

8.1.4. Creating a VM from an instance type by using the web console

You can create a virtual machine (VM) from an instance type by using the OpenShift Container Platform web console. You can also use the web console to create a VM by copying an existing snapshot or to clone a VM.

You can create a VM from a list of available bootable volumes. You can add Linux- or Windows-based volumes to the list.

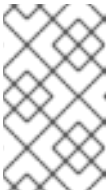
Procedure

1. In the web console, navigate to **Virtualization → Catalog**.
   The **InstanceTypes** tab opens by default.

NOTE

When configuring a downward-metrics device on an IBM Z® system that uses a VM preference, set the **spec.preference.name** value to **rhel.9.s390x** or another available preference with the format ***.s390x**.

2. Heterogeneous clusters only: To filter the bootable volumes using the options provided, click **Architecture**.
3. Select either of the following options:
   - Select a suitable bootable volume from the list. If the list is truncated, click the **Show all** button to display the entire list.



NOTE

The bootable volume table lists only those volumes in the **openshift-virtualization-os-images** namespace that have the **instancetype.kubevirt.io/default-preference** label.

     - Optional: Click the star icon to designate a bootable volume as a favorite. Starred bootable volumes appear first in the volume list.
   - Click **Add volume** to upload a new volume or to use an existing persistent volume claim (PVC), a volume snapshot, or a **containerDisk** volume. Click **Save**.
     Logos of operating systems that are not available in the cluster are shown at the bottom of the list. You can add a volume for the required operating system by clicking the **Add volume** link.

     In addition, there is a link to the **Create a Windows bootable volume** quick start. The same link appears in a popover if you hover the pointer over the question mark icon next to the *Select volume to boot from* line.

Immediately after you install the environment or when the environment is disconnected, the list of volumes to boot from is empty. In that case, three operating system logos are displayed: Windows, RHEL, and Linux. You can add a new volume that meets your requirements by clicking the **Add volume** button.

4. Click an instance type tile and select the resource size appropriate for your workload. You can select huge pages for Red Hat-provided instance types of the **M** and **CX** series. Huge page options are identified by names that end with **1gi**.

5. Optional: Choose the virtual machine details, including the VM's name, that apply to the volume you are booting from:

   - For a Linux-based volume, follow these steps to configure SSH:

     a. If you have not already added a public SSH key to your project, click the edit icon beside **Authorized SSH key** in the **VirtualMachine details** section.

     b. Select one of the following options:

        - **Use existing**: Select a secret from the secrets list.
        - **Add new**: Follow these steps:

          i. Browse to the public SSH key file or paste the file in the key field.

          ii. Enter the secret name.

          iii. Optional: Select **Automatically apply this key to any new VirtualMachine you create in this project**.

     c. Click **Save**.

   - For a Windows volume, follow either of these set of steps to configure sysprep options:

     - If you have not already added sysprep options for the Windows volume, follow these steps:

       i. Click the edit icon beside **Sysprep** in the **VirtualMachine details** section.

       ii. Add the **Autoattend.xml** answer file.

       iii. Add the **Unattend.xml** answer file.

       iv. Click **Save**.

     - If you want to use existing sysprep options for the Windows volume, follow these steps:

       i. Click **Attach existing sysprep**.

       ii. Enter the name of the existing sysprep **Unattend.xml** answer file.

       iii. Click **Save**.

6. Optional: If you are creating a Windows VM, you can mount a Windows driver disk:

   a. Click the **Customize VirtualMachine** button.

   b. On the **VirtualMachine details** page, click **Storage**.

c. Select the **Mount Windows drivers disk** checkbox.

7. Optional: Click **View YAML & CLI** to view the YAML file. Click **CLI** to view the CLI commands. You can also download or copy either the YAML file contents or the CLI commands.

8. Click **Create VirtualMachine**.

After the VM is created, you can monitor the status on the **VirtualMachine details** page.

Additional resources

- Configuring a downward metrics device

8.1.5. Changing the instance type for a VM

As a cluster administrator or VM owner, you might want to change the instance type for an existing VM for the following reasons:

- If a VM's workload has increased, you might change the instance type to one with more CPU, more memory, or specific hardware resources, to prevent performance bottlenecks.
- If you are using specialized workloads, you might switch to a different instance type to improve performance, as some instance types are optimized for specific use cases.

You can use the OpenShift Container Platform web console or the OpenShift CLI (**oc**) to change the instance type for an existing VM.

8.1.5.1. Changing the instance type of a VM by using the web console

You can change the instance type associated with a running virtual machine (VM) by using the web console. The change takes effect immediately.

Prerequisites

- You created the VM by using an instance type.

Procedure

1. In the OpenShift Container Platform web console, click **Virtualization → VirtualMachines**.
2. Select a VM to open the **VirtualMachine details** page.
3. Click the **Configuration** tab.
4. On the **Details** tab, click the instance type text to open the **Edit Instancetype** dialog. For example, click **1 CPU | 2 GiB Memory**
5. Edit the instance type by using the **Series** and **Size** lists.
   a. Select an item from the **Series** list to show the relevant sizes for that series. For example, select **General Purpose**.
   b. Select the VM's new instance type from the **Size** list. For example, select **medium: 1 CPUs, 4Gi Memory**, which is available in the **General Purpose** series.
6. Click **Save**.

Verification

1. Click the **YAML** tab.
2. Click **Reload**.
3. Review the VM YAML to confirm that the instance type changed.

8.1.5.2. Changing the instance type of a VM by using the CLI

To change the instance type of a VM, change the **name** field in the VM spec. This triggers the update logic, which ensures that a new, immutable controller revision snapshot is taken of the new resource configuration.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You created the VM by using an instance type, or have administrator privileges for the VM that you want to modify.

Procedure

1. Stop the VM.
2. Run the following command, and replace **<vm_name>** with the name of your VM, and **<new_instancetype>** with the name of the instance type you want to change to:

```
$ oc patch vm/<vm_name> --type merge -p '{"spec":{"instancetype":{"name": "
<new_instancetype>"}}}'
```

Verification

- Check the controller revision reference in the updated VM **status** field. Run the following command and verify that the revision name is updated in the output:

```
$ oc get vms/<vm_name> -o json | jq .status.instancetypeRef
```

Example output

```
{
  "controllerRevisionRef": {
    "name": "vm-cirros-csmall-csmall-3e86e367-9cd7-4426-9507-b14c27a08671-2"
  },
  "kind": "VirtualMachineInstancetype",
  "name": "csmall"
}
```

- Optional: Check that the VM instance is running the new configuration defined in the latest controller revision. For example, if you updated the instance type to use 2 vCPUs instead of 1, run the following command and check the output:

```
$ oc get vmi/<vm_name> -o json | jq .spec.domain.cpu
```

Example output that verifies that the revision uses 2 vCPUs

```
{
  "cores": 1,
  "model": "host-model",
  "sockets": 2,
  "threads": 1
}
```

8.2. CREATING VIRTUAL MACHINES FROM TEMPLATES

You can create virtual machines (VMs) from Red Hat templates by using the OpenShift Container Platform web console.

8.2.1. About VM templates

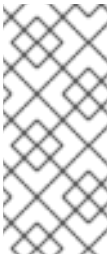
You can use VM templates to help you easily create VMs.

Expedite creation with boot sources

You can expedite VM creation by using templates that have an available boot source. Templates with a boot source are labeled **Available boot source** if they do not have a custom label.
Templates without a boot source are labeled **Boot source required**. See Managing automatic boot source updates for details.

Customize before starting the VM

You can customize the disk source and VM parameters before you start the VM.



NOTE

If you copy a VM template with all its labels and annotations, your version of the template is marked as deprecated when a new version of the Scheduling, Scale, and Performance (SSP) Operator is deployed. You can remove this designation. See Removing a deprecated designation from a customized VM template by using the web console.

Single-node OpenShift

Due to differences in storage behavior, some templates are incompatible with single-node OpenShift. To ensure compatibility, do not set the **evictionStrategy** field for templates or VMs that use data volumes or storage profiles.

8.2.2. Creating a VM from a template

You can create a virtual machine (VM) from a template with an available boot source by using the OpenShift Container Platform web console. You can customize template or VM parameters, such as data sources, Cloud-init, or SSH keys, before you start the VM.

You can choose between two views in the web console to create the VM:

- A virtualization-focused view, which provides a concise list of virtualization-related options at the top of the view

- A general view, which provides access to the various web console options, including **Virtualization**

Procedure

1. From the OpenShift Container Platform web console, choose your view:
   - For a virtualization-focused view, select **Administrator → Virtualization → Catalog**.
   - For a general view, navigate to **Virtualization → Catalog**.
2. Click the **Template catalog** tab.
3. Click the **Boot source available** checkbox to filter templates with boot sources. The catalog displays the default templates.
4. Heterogeneous clusters only: To filter the search results to show templates associated with a particular architecture, click **Architecture Type** .
5. Click **All templates** to view the available templates for your filters.
   - To focus on particular templates, enter the keyword in the **Filter by keyword** field.
   - Choose a template project from the **All projects** dropdown menu, or view all projects.
6. Click a template tile to view its details.
   - Optional: If you are using a Windows template, you can mount a Windows driver disk by selecting the **Mount Windows drivers disk** checkbox.
   - If you do not need to customize the template or VM parameters, click **Quick create VirtualMachine** to create a VM from the template.
   - If you need to customize the template or VM parameters, do the following:
     a. Click **Customize VirtualMachine**. The **Customize and create VirtualMachine** page displays the **Overview**, **YAML**, **Scheduling**, **Environment**, **Network interfaces**, **Disks**, **Scripts**, and **Metadata** tabs.
     b. Click the **Scripts** tab to edit the parameters that must be set before the VM boots, such as **Cloud-init**, **SSH key**, or **Sysprep** (Windows VM only).
     c. Optional: Click the **Start this virtualmachine after creation (Always)** checkbox.
     d. Click **Create VirtualMachine**.
     The **VirtualMachine details** page displays the provisioning status.

8.2.2.1. Removing a deprecated designation from a customized VM template by using the web console

You can customize an existing virtual machine (VM) template by modifying the VM or template parameters, such as data sources, cloud-init, or SSH keys, before you start the VM. If you customize a template by copying it and including all of its labels and annotations, the customized template is marked as deprecated when a new version of the Scheduling, Scale, and Performance (SSP) Operator is deployed.

You can remove the deprecated designation from the customized template.

Procedure

1. Navigate to **Virtualization → Templates** in the web console.
2. From the list of VM templates, click the template marked as deprecated.
3. Click **Edit** next to the pencil icon beside **Labels**.
4. Remove the following two labels:
   - **template.kubevirt.io/type: "base"**
   - **template.kubevirt.io/version: "version"**
5. Click **Save**.
6. Click the pencil icon beside the number of existing **Annotations**.
7. Remove the following annotation:
   - **template.kubevirt.io/deprecated**
8. Click **Save**.

8.2.2.2. Creating a custom VM template in the web console

You create a virtual machine template by editing a YAML file example in the OpenShift Container Platform web console.

Procedure

1. In the web console, click **Virtualization → Templates** in the side menu.
2. Optional: Use the **Project** drop-down menu to change the project associated with the new template. All templates are saved to the **openshift** project by default.
3. Click **Create Template**.
4. Specify the template parameters by editing the YAML file.
5. Click **Create**.
   The template is displayed on the **Templates** page.
6. Optional: Click **Download** to download and save the YAML file.

8.3. CONFIGURING IBM SECURE EXECUTION VIRTUAL MACHINES ON IBM Z AND IBM LINUXONE

You can configure IBM® Secure Execution virtual machines (VMs) on IBM Z® and IBM® LinuxONE.

IBM® Secure Execution for Linux is a s390x security technology that is introduced with IBM® z15 and IBM® LinuxONE III. It protects data of workloads that run in a KVM guest from being inspected or modified by the server environment.

In particular, no hardware administrator, no KVM code, and no KVM administrator can access the data in a guest that was started as an IBM Secure Execution guest.

Additional resources

- What is IBM Secure Execution?

8.3.1. Enabling VMs to run IBM(R) Secure Execution on IBM Z(R) and IBM(R) LinuxONE

To enable IBM® Secure Execution virtual machines (VMs) on IBM Z® and IBM® LinuxONE on the compute nodes of your cluster, you must ensure that you meet the prerequisites and complete the following steps.

Prerequisites

- Your cluster has logical partition (LPAR) nodes running on IBM® z15 or later, or IBM® LinuxONE III or later.
- You have IBM® Secure Execution workloads available to run on the cluster.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. To run IBM® Secure Execution VMs, you must add the **prot_virt=1** kernel parameter for each compute node. To enable all compute nodes, create a file named **secure-execution.yaml** that contains the following machine config manifest:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: secure-execution
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  kernelArguments:
    - prot_virt=1
```

   where:

   prot_virt=1

   Specifies that the ultravisor can store memory security information.

2. Apply the changes by running the following command:

```
$ oc apply -f secure-execution.yaml
```

   The Machine Config Operator (MCO) applies the changes and reboots the nodes in a controlled rollout.

3. Edit the **HyperConverged** custom resource (CR) by running the following command:

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

4. Enable the feature gate for IBM® Secure Execution by applying the following annotations:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  annotations:
    kubevirt.kubevirt.io/jsonpatch: |-
     [
      {
       "op":"add",
       "path":"/spec/configuration/developerConfiguration/featureGates/-",
       "value":"SecureExecution"
      }
     ]
```

8.3.2. Launching an IBM Secure Execution VM on IBM Z and IBM LinuxONE

Before launching an IBM® Secure Execution VM on IBM Z® and IBM® LinuxONE, you must add the **launchSecurity** parameter to the VM manifest. Otherwise, the VM does not boot correctly because it does not have access to the devices.

Procedure

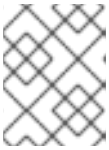
- Apply the following **VirtualMachine** manifest to the cluster:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: f41-se
  name: f41-se
spec:
  runStrategy: Always
  template:
    metadata:
      labels:
        kubevirt.io/vm: f41-se
    spec:
      domain:
        launchSecurity: {}
        devices:
          disks:
          - disk:
              bus: virtio
            name: rootfs
        machine:
          type: ""
        resources:
          requests:
            memory: 4Gi
      terminationGracePeriodSeconds: 0
      volumes:
        - name: rootfs
          dataVolume:
            name: f41-se
```

To launch IBM® Secure Execution VMs, you must include the following YAML in the manifest:

```
spec:
  domain:
    launchSecurity: {}
```

The rest of the VM manifest is variable depending on your setup.



NOTE

Because the memory of the VM is protected, IBM® Secure Execution VMs are not live migratable. The VMs can only be migrated offline.

CHAPTER 9. ADVANCED VM CREATION

9.1. CREATING VMS FROM RED HAT IMAGES

9.1.1. Creating virtual machines from Red Hat images

Red Hat images are golden images. They are published as container disks in a secure registry. The Containerized Data Importer (CDI) polls and imports the container disks into your cluster and stores them in the **openshift-virtualization-os-images** project as snapshots or persistent volume claims (PVCs). You can optionally use a custom namespace for golden images. For more information about using a custom namespace, see:

- Configuring a custom namespace for golden images by using the web console
- Configuring a custom namespace for golden images by using the CLI

Red Hat images are automatically updated. You can disable and re-enable automatic updates for these images. See Managing Red Hat boot source updates .

Cluster administrators can enable automatic subscription for Red Hat Enterprise Linux (RHEL) virtual machines in the OpenShift Virtualization web console.

You can create virtual machines (VMs) from operating system images provided by Red Hat by using one of the following methods:

- Creating a VM from a template by using the web console
- Creating a VM from an instance type by using the web console
- Creating a VM from a **VirtualMachine** manifest by using the command line

IMPORTANT

Do not create VMs in the default **openshift-*** namespaces. Instead, create a new namespace or use an existing namespace without the **openshift** prefix.

9.1.1.1. About golden images

A golden image is a preconfigured snapshot of a virtual machine (VM) that you can use as a resource to deploy new VMs. For example, you can use golden images to provision the same system environment consistently and deploy systems more quickly and efficiently.

9.1.1.1.1. How do golden images work?

Golden images are created by installing and configuring an operating system and software applications on a reference machine or virtual machine. This includes setting up the system, installing required drivers, applying patches and updates, and configuring specific options and preferences.

After the golden image is created, it is saved as a template or image file that can be replicated and deployed across multiple clusters. The golden image can be updated by its maintainer periodically to incorporate necessary software updates and patches, ensuring that the image remains up to date and secure, and newly created VMs are based on this updated image.

9.1.1.1.2. Red Hat implementation of golden images

Red Hat publishes golden images as container disks in the registry for versions of Red Hat Enterprise Linux (RHEL). Container disks are virtual machine images that are stored as a container image in a container image registry. Any published image will automatically be made available in connected clusters after the installation of OpenShift Virtualization. After the images are available in a cluster, they are ready to use to create VMs.

9.1.1.2. About VM boot sources

Virtual machines (VMs) consist of a VM definition and one or more disks that are backed by data volumes. VM templates enable you to create VMs using predefined specifications.

Every template requires a boot source, which is a fully configured disk image including configured drivers. Each template contains a VM definition with a pointer to the boot source. Each boot source has a predefined name and namespace. For some operating systems, a boot source is automatically provided. If it is not provided, then an administrator must prepare a custom boot source.

Provided boot sources are updated automatically to the latest version of the operating system. For auto-updated boot sources, persistent volume claims (PVCs) and volume snapshots are created with the cluster's default storage class. If you select a different default storage class after configuration, you must delete the existing boot sources in the cluster namespace that are configured with the previous default storage class.

9.1.1.3. Configuring a custom namespace for golden images by using the web console

You can configure a custom namespace for golden images in your cluster by using the OpenShift Container Platform web console.

Procedure

1. In the web console, select **Virtualization → Overview**.
2. Select the **Settings** tab.
3. On the **Cluster** tab, select **General settings → Bootable volumes project**.
4. Select a namespace to use for golden images.
    a. If you already created a namespace, select it from the **Project** list.
    b. If you did not create a namespace, scroll to the bottom of the list and click **Create project**.
        i. Enter a name for your new namespace in the **Name** field of the **Create project** dialog.
        ii. Click **Create**.

9.1.1.4. Configuring a custom namespace for golden images by using the CLI

You can configure a custom namespace for golden images in your cluster by setting the **spec.commonBootImageNamespace** field in the **HyperConverged** custom resource (CR).

Prerequisites

- You installed the OpenShift CLI (**oc**).

- You created a namespace to use for golden images.

Procedure

1. Open the **HyperConverged** CR in your default editor by running the following command:

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

2. Configure the custom namespace by updating the value of the **spec.commonBootImageNamespace** field:

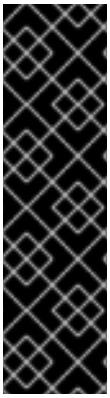
Example configuration file

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  commonBootImageNamespace: <custom_namespace> 1
# ...
```

1 The namespace to use for golden images.

3. Save your changes and exit the editor.

9.1.2. Heterogeneous cluster support



IMPORTANT

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

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

A heterogeneous cluster is a cluster where nodes have differing architectures. Heterogeneous clusters promote optimal compute resource usage by mixing different types of hardware in one cluster. This allows workloads to be better matched to hardware intended for the workload task instead of general purpose compute platforms. For example, in a heterogeneous cluster, GPU and general purpose compute resources could be combined and workloads assigned to the appropriate hardware.

IMPORTANT

If golden image support is disabled in a heterogeneous cluster, you can encounter inconsistencies between node and image architectures. This happens when images are used for virtual machine creation that do not match the node architecture. This can lead to the failure of virtual machine boot up or virtual machines that do not run as expected. The warning level alert **HCOMultiArchGoldenImagesDisabled** is produced when this feature is not enabled in a heterogeneous cluster.

If you have a heterogeneous cluster but do not want to enable multiple architecture support, see Modifying workloads node placement in a hetergeneous cluster for the procedure to limit node placement to a specific architecture.

Golden image support for heterogeneous clusters extends golden image support in the following areas:

- Enables VM creators to deploy persistent virtual machines with specific architectures.
- Enables VM creators to define custom golden images that support heterogenous clusters.

The same golden image can be used with nodes of different architectures if the boot image supports the required architectures. For example, a golden image that supports both ARM and AMD architectures can be used with both types of nodes.

Golden image support for heterogeneous clusters is not enabled by default. For the procedure to enable this feature, see Enabling hetergenous cluster support

9.1.2.1. Enabling heterogeneous cluster support

You can enable golden image support for heterogeneous clusters by setting the **enableMultiArchBootImageImport** feature gate to **true** in the **HyperConverged** custom resource (CR).

IMPORTANT

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

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

Prerequisites

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

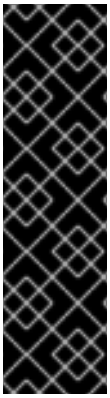
Procedure

- Enable the **enableMultiArchBootImageImport** feature gate by running the following command:

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

9.1.2.2. Modifying a common golden image source in a heterogeneous cluster

You can modify the image source of a common golden image in a heterogeneous cluster by specifying the supported architectures in the **ssp.kubevirt.io/dict.architectures** annotation in the **HyperConverged** custom resource (CR).



IMPORTANT

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

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

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Open the **HyperConverged** CR in your default editor by running the following command:

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

2. Edit the **HyperConverged** CR, adding the appropriate values for **ssp.kubevirt.io/dict.architectures** annotation in the **dataImportCronTemplates** section. For example:

```
#...
spec:
  dataImportCronTemplates:
  - metadata:
      name: kubevirt-hyperconverged
      annotations:
        ssp.kubevirt.io/dict.architectures: "<architecture_list>" 1
    spec:
      schedule: "0 */12 * * *"
      template:
        spec:
          source:
            registry:
                url: docker://my-private-registry/my-own-version-of-centos:8
      managedDataSource: centos-stream8
#...
```

1. The comma-separated list of supported architectures for this image. For example, if the image supports **amd64** and **arm64** architectures, the value would be **"amd64,arm64"**.

3. Save and exit the editor to update the **HyperConverged** CR.

9.1.2.3. Adding a custom golden image in a heterogeneous cluster

IMPORTANT

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

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

Add a custom golden image in a heterogeneous cluster by setting the **ssp.kubevirt.io/dict.architectures** annotation in the **spec.dataImportCronTemplates.metadata.annotations** stanza of the **HyperConverged** custom resource (CR). This annotation lists the architectures supported by the image.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Open the **HyperConverged** CR in your default editor by running the following command:

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

2. Edit the **HyperConverged** CR, to add the custom golden image. You must add the appropriate values for **ssp.kubevirt.io/dict.architectures** annotation in the **dataImportCronTemplates** section. For example:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      name: custom-image1
      annotations:
        ssp.kubevirt.io/dict.architectures: "<architecture_list>" 1
    spec:
      schedule: "0 */12 * * *"
      template:
        spec:
```

```
      source:
        registry:
          url: docker://myprivateregistry/custom1
    managedDataSource: custom1
    retentionPolicy: "All"
#...
```

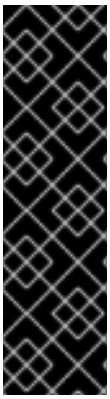
1 The comma-separated list of supported architectures for this image. For example, if the image supports **amd64** and **arm64** architectures, the value would be **"amd64,arm64"**.

NOTE

An image may support more architectures than you want to use in your cluster. You do not have to list all of the architectures an image supports, only those for which you want to create a boot source.

3. Save and exit the editor to update the **HyperConverged** CR.

9.1.2.4. Modifying workloads node placement in a heterogeneous cluster



IMPORTANT

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

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

If you have a heterogeneous cluster but not want to enable multiple archiecture support, you can modify the workloads node placement in the **HyperConverged** custom resource (CR) to only include nodes with a specific architecture.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Open the **HyperConverged** CR in your default editor by running the following command:

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

2. Edit the **HyperConverged** CR, to modify the workloads node placement to include only nodes with a specific architecture. For example:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
```

```
  name: kubevirt-hyperconverged
spec:
#...
  workloads:
    nodePlacement:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: kubernetes.io/arch
                    operator: In
                    values:
                      - <node_architecture> 1
```

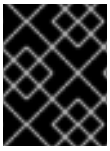
1 Replace **<node_architecture>** with the target architecture. For example, to limit placement to AMD nodes, use **amd64**.

3. Save and exit the editor to update the **HyperConverged** CR.

9.2. CREATING VMS IN THE WEB CONSOLE

9.2.1. Creating VMs by importing images from web pages

You can create virtual machines (VMs) by importing operating system images from web pages.



IMPORTANT

You must install the QEMU guest agent on VMs created from operating system images that are not provided by Red Hat.

9.2.1.1. Creating a VM from an image on a web page by using the web console

You can create a virtual machine (VM) by importing an image from a web page by using the OpenShift Container Platform web console.

Prerequisites

- You must have access to the web page that contains the image.

Procedure

1. Navigate to **Virtualization → Catalog** in the web console.
2. Click a template tile without an available boot source.
3. Click **Customize VirtualMachine**.
4. On the **Customize template parameters** page, expand **Storage** and select **URL (creates PVC)** from the **Disk source** list.
5. Enter the image URL. Example: **https://access.redhat.com/downloads/content/69/ver=/rhel---7/7.9/x86_64/product-software**

6. Set the disk size.
7. Click **Next**.
8. Click **Create VirtualMachine**.

9.2.1.2. Creating a VM from an image on a web page by using the CLI

You can create a virtual machine (VM) from an image on a web page by using the command line.

When the VM is created, the data volume with the image is imported into persistent storage.

Prerequisites

- You must have access credentials for the web page that contains the image.
- You have installed the **virtctl** CLI.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **VirtualMachine** manifest for your VM and save it as a YAML file. For example, to create a minimal Red Hat Enterprise Linux (RHEL) VM from an image on a web page, run the following command:

```
$ virtctl create vm --name vm-rhel-9 --instancetype u1.small --preference rhel.9 --volume-import type:http,url:https://example.com/rhel9.qcow2,size:10Gi
```

2. Review the **VirtualMachine** manifest for your VM:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-rhel-9 1
spec:
  dataVolumeTemplates:
  - metadata:
      name: imported-volume-6dcpf 2
    spec:
      source:
        http:
          url: https://example.com/rhel9.qcow2 3
      storage:
        resources:
          requests:
            storage: 10Gi 4
  instancetype:
    name: u1.small 5
  preference:
    name: rhel.9 6
  runStrategy: Always
  template:
    spec:
```

```
    domain:
      devices: {}
      resources: {}
    terminationGracePeriodSeconds: 180
    volumes:
    - dataVolume:
        name: imported-volume-6dcpf
      name: imported-volume-6dcpf
```

1. The VM name.
2. The data volume name.
3. The URL of the image.
4. The size of the storage requested for the data volume.
5. The instance type to use to control resource sizing of the VM.
6. The preference to use.

3. Create the VM by running the following command:

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

The **oc create** command creates the data volume and the VM. The CDI controller creates an underlying PVC with the correct annotation and the import process begins. When the import is complete, the data volume status changes to **Succeeded**. You can start the VM.

Data volume provisioning happens in the background, so there is no need to monitor the process.

Verification

1. The importer pod downloads the image from the specified URL and stores it on the provisioned persistent volume. View the status of the importer pod:

```
$ oc get pods
```

2. Monitor the status of the data volume:

```
$ oc get dv <data_volume_name>
```

If the provisioning is successful, the data volume phase is **Succeeded**:

Example output

```
NAME                    PHASE       PROGRESS   RESTARTS   AGE
imported-volume-6dcpf   Succeeded   100.0%                18s
```

3. Verify that provisioning is complete and that the VM has started by accessing its serial console:

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

If the VM is running and the serial console is accessible, the output looks as follows:

Example output

```
Successfully connected to vm-rhel-9 console. The escape sequence is ^]
```

9.2.2. Creating VMs by uploading images

You can create virtual machines (VMs) by uploading operating system images from your local machine.

You can create a Windows VM by uploading a Windows image to a PVC. Then you clone the PVC when you create the VM.

IMPORTANT

You must install the QEMU guest agent on VMs created from operating system images that are not provided by Red Hat.

You must also install VirtIO drivers on Windows VMs.

9.2.2.1. Creating a VM from an uploaded image by using the web console

You can create a virtual machine (VM) from an uploaded operating system image by using the OpenShift Container Platform web console.

Prerequisites

- You must have an **IMG**, **ISO**, or **QCOW2** image file.

Procedure

1. Navigate to **Virtualization → Catalog** in the web console.
2. Click a template tile without an available boot source.
3. Click **Customize VirtualMachine**.
4. On the **Customize template parameters** page, expand **Storage** and select **Upload (Upload a new file to a PVC)** from the **Disk source** list.
5. Browse to the image on your local machine and set the disk size.
6. Click **Customize VirtualMachine**.
7. Click **Create VirtualMachine**.

9.2.2.1.1. Generalizing a VM image

You can generalize a Red Hat Enterprise Linux (RHEL) image to remove all system-specific configuration data before you use the image to create a golden image, a preconfigured snapshot of a virtual machine (VM). You can use a golden image to deploy new VMs.

You can generalize a RHEL VM by using the **virtctl**, **guestfs**, and **virt-sysprep** tools.

Prerequisites

- You have a RHEL virtual machine (VM) to use as a base VM.
- You have installed the OpenShift CLI (**oc**).
- You have installed the **virtctl** tool.

Procedure

1. Stop the RHEL VM if it is running, by entering the following command:

```
$ virtctl stop <my_vm_name>
```

2. Optional: Clone the virtual machine to avoid losing the data from your original VM. You can then generalize the cloned VM.

3. Retrieve the **dataVolume** that stores the root filesystem for the VM by running the following command:

```
$ oc get vm <my_vm_name> -o jsonpath="{.spec.template.spec.volumes}{'\n'}"
```

Example output

```
[{"dataVolume":{"name":"<my_vm_volume>"},"name":"rootdisk"},{"cloudInitNoCloud":{...}]
```

4. Retrieve the persistent volume claim (PVC) that matches the listed **dataVolume** by running the followimg command:

```
$ oc get pvc
```

Example output

```
NAME          STATUS   VOLUME  CAPACITY   ACCESS MODES  STORAGECLASS
AGE
<my_vm_volume> Bound  …
```

NOTE

If your cluster configuration does not enable you to clone a VM, to avoid losing the data from your original VM, you can clone the VM PVC to a data volume instead. You can then use the cloned PVC to create a golden image.

If you are creating a golden image by cloning a PVC, continue with the next steps, using the cloned PVC.

5. Deploy a new interactive container with **libguestfs-tools** and attach the PVC to it by running the following command:

```
$ virtctl guestfs <my-vm-volume> --uid 107
```

This command opens a shell for you to run the next command.

6. Remove all configurations specific to your system by running the following command:

```
$ virt-sysprep -a disk.img
```

7. In the OpenShift Container Platform console, click **Virtualization → Catalog**.
8. Click **Add volume**.
9. In the **Add volume** window:
    a. From the **Source type** list, select **Use existing Volume**.
    b. From the **Volume project** list, select your project.
    c. From the **Volume name** list, select the correct PVC.
    d. In the **Volume name** field, enter a name for the new golden image.
    e. From the **Preference** list, select the RHEL version you are using.
    f. From the **Default Instance Type** list, select the instance type with the correct CPU and memory requirements for the version of RHEL you selected previously.
    g. Heterogeneous clusters only: From the **Architecture** list, select the architecture that corresponds with the selected volume.
    h. Click **Save**.

The new volume appears in the **Select volume to boot from** list. This is your new golden image. You can use this volume to create new VMs.

Additional resources for generalizing VMs

- Cloning VMs
- Cloning a PVC to a data volume

9.2.2.2. Creating a Windows VM

You can create a Windows virtual machine (VM) by uploading a Windows image to a persistent volume claim (PVC) and then cloning the PVC when you create a VM by using the OpenShift Container Platform web console.

Prerequisites

- You created a Windows installation DVD or USB with the Windows Media Creation Tool. See Create Windows 10 installation media in the Microsoft documentation.
- You created an **autounattend.xml** answer file. See Answer files (unattend.xml) in the Microsoft documentation.

Procedure

1. Upload the Windows image as a new PVC:
    a. Navigate to **Storage → PersistentVolumeClaims** in the web console.

b. Click **Create PersistentVolumeClaim → With Data upload form**.

c. Browse to the Windows image and select it.

d. Enter the PVC name, select the storage class and size and then click **Upload**.
The Windows image is uploaded to a PVC.

2. Configure a new VM by cloning the uploaded PVC:

a. Navigate to **Virtualization → Catalog**.

b. Select a Windows template tile and click **Customize VirtualMachine**.

c. Select **Clone (clone PVC)** from the **Disk source** list.

d. Select the PVC project, the Windows image PVC, and the disk size.

3. Apply the answer file to the VM:

a. Click **Customize VirtualMachine parameters**.

b. On the **Sysprep** section of the **Scripts** tab, click **Edit**.

c. Browse to the **autounattend.xml** answer file and click **Save**.

4. Set the run strategy of the VM:

a. Clear **Start this VirtualMachine after creation** so that the VM does not start immediately.

b. Click **Create VirtualMachine**.

c. On the **YAML** tab, replace **running:false** with **runStrategy: RerunOnFailure** and click **Save**.

5. Click the Options menu  and select **Start**.
The VM boots from the **sysprep** disk containing the **autounattend.xml** answer file.

9.2.2.2.1. Generalizing a Windows VM image

You can generalize a Windows operating system image to remove all system-specific configuration data before you use the image to create a new virtual machine (VM).

Before generalizing the VM, you must ensure the **sysprep** tool cannot detect an answer file after the unattended Windows installation.

Prerequisites

- A running Windows VM with the QEMU guest agent installed.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization → VirtualMachines**.

2. Select a Windows VM to open the **VirtualMachine details** page.

3. Click **Configuration → Disks**.
4. Click the Options menu  beside the **sysprep** disk and select **Detach**.
5. Click **Detach**.
6. Rename **C:\Windows\Panther\unattend.xml** to avoid detection by the **sysprep** tool.
7. Start the **sysprep** program by running the following command:

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

8. After the **sysprep** tool completes, the Windows VM shuts down. The disk image of the VM is now available to use as an installation image for Windows VMs.

You can now specialize the VM.

9.2.2.2.2. Specializing a Windows VM image

Specializing a Windows virtual machine (VM) configures the computer-specific information from a generalized Windows image onto the VM.

Prerequisites

- You must have a generalized Windows disk image.
- You must create an **unattend.xml** answer file. See the Microsoft documentation for details.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization → Catalog**.
2. Select a Windows template and click **Customize VirtualMachine**.
3. Select **PVC (clone PVC)** from the **Disk source** list.
4. Select the PVC project and PVC name of the generalized Windows image.
5. Click **Customize VirtualMachine parameters**.
6. Click the **Scripts** tab.
7. In the **Sysprep** section, click **Edit**, browse to the **unattend.xml** answer file, and click **Save**.
8. Click **Create VirtualMachine**.

During the initial boot, Windows uses the **unattend.xml** answer file to specialize the VM. The VM is now ready to use.

Additional resources for creating Windows VMs

- Microsoft, Sysprep (Generalize) a Windows installation
- Microsoft, generalize

- Microsoft, specialize

9.2.2.3. Creating a VM from an uploaded image by using the CLI

You can upload an operating system image by using the **virtctl** command-line tool. You can use an existing data volume or create a new data volume for the image.

Prerequisites

- You must have an **ISO**, **IMG**, or **QCOW2** operating system image file.
- For best performance, compress the image file by using the virt-sparsify tool or the **xz** or **gzip** utilities.
- The client machine must be configured to trust the OpenShift Container Platform router's certificate.
- You have installed the **virtctl** CLI.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Upload the image by running the **virtctl image-upload** command:

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

1. The name of the data volume.
2. The size of the data volume. For example: **--size=500Mi**, **--size=1G**
3. The file path of the image.

NOTE

- If you do not want to create a new data volume, omit the **--size** parameter and include the **--no-create** flag.
- When uploading a disk image to a PVC, the PVC size must be larger than the size of the uncompressed virtual disk.
- To allow insecure server connections when using HTTPS, use the **--insecure** parameter. When you use the **--insecure** flag, the authenticity of the upload endpoint is **not** verified.

2. Optional. To verify that a data volume was created, view all data volumes by running the following command:

```
$ oc get dvs
```

9.2.3. Cloning VMs

You can clone virtual machines (VMs) or create new VMs from snapshots.

IMPORTANT

Cloning a VM with a vTPM device attached to it or creating a new VM from its snapshot is not supported.

9.2.3.1. Cloning a VM by using the web console

You can clone an existing VM by using the web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Select a VM to open the **VirtualMachine details** page.
3. Click **Actions**.
   Alternatively, access the same menu in the tree view by right-clicking the VM.
4. Select **Clone**.
5. On the **Clone VirtualMachine** page, enter the name of the new VM.
6. (Optional) Select the **Start cloned VM**checkbox to start the cloned VM.
7. Click **Clone**.

9.2.3.2. Creating a VM from an existing snapshot by using the web console

You can create a new VM by copying an existing snapshot.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Select a VM to open the **VirtualMachine details** page.
3. Click the **Snapshots** tab.
4. Click the Options menu  for the snapshot you want to copy.
5. Select **Create VirtualMachine**.
6. Enter the name of the virtual machine.
7. (Optional) Select the **Start this VirtualMachine after creation**checkbox to start the new virtual machine.
8. Click **Create**.

9.2.3.3. Additional resources

- Creating VMs by cloning PVCs

9.3. CREATING VMS USING THE CLI

9.3.1. Creating virtual machines from the CLI

You can create virtual machines (VMs) from the command line by editing or creating a **VirtualMachine** manifest. You can simplify VM configuration by using an instance type in your VM manifest.

NOTE

You can also create VMs from instance types by using the web console .

9.3.1.1. Creating a VM from a VirtualMachine manifest

You can create a virtual machine (VM) from a **VirtualMachine** manifest. To simplify the creation of these manifests, you can use the **virtctl** command-line tool.

Prerequisites

- You have installed the **virtctl** CLI.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **VirtualMachine** manifest for your VM and save it as a YAML file. For example, to create a minimal Red Hat Enterprise Linux (RHEL) VM, run the following command:

```
$ virtctl create vm --name rhel-9-minimal --volume-import type:ds,src:openshift-virtualization-os-images/rhel9
```

2. Review the **VirtualMachine** manifest for your VM:

NOTE

This example manifest does not configure VM authentication.

Example manifest for a RHEL VM

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: rhel-9-minimal (1)
spec:
  dataVolumeTemplates:
  - metadata:
      name: imported-volume-mk4lj
    spec:
      sourceRef:
```

```
      kind: DataSource
      name: rhel9 (2)
      namespace: openshift-virtualization-os-images (3)
    storage:
      resources: {}
instancetype:
  inferFromVolume: imported-volume-mk4lj (4)
  inferFromVolumeFailurePolicy: Ignore
preference:
  inferFromVolume: imported-volume-mk4lj (5)
  inferFromVolumeFailurePolicy: Ignore
runStrategy: Always
template:
  spec:
    domain:
      devices: {}
      memory:
        guest: 512Mi
      resources: {}
    terminationGracePeriodSeconds: 180
    volumes:
    - dataVolume:
        name: imported-volume-mk4lj
      name: imported-volume-mk4lj
```

(1) The VM name.

(2) The boot source for the guest operating system.

(3) The namespace for the boot source. Golden images are stored in the **openshift-virtualization-os-images** namespace.

(4) The instance type is inferred from the selected **DataSource** object.

(5) The preference is inferred from the selected **DataSource** object.

3. Create a virtual machine by using the manifest file:

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

4. Optional: Start the virtual machine:

```
$ virtctl start <vm_name>
```

Next steps

- Configuring SSH access to virtual machines

9.3.2. Creating VMs by using container disks

You can create virtual machines (VMs) by using container disks built from operating system images.

You can enable auto updates for your container disks. See Managing automatic boot source updates for details.

IMPORTANT

If the container disks are large, the I/O traffic might increase and cause worker nodes to be unavailable. You can perform the following tasks to resolve this issue:

- Pruning **DeploymentConfig** objects.
- Configuring garbage collection.

You create a VM from a container disk by performing the following steps:

1. Build an operating system image into a container disk and upload it to your container registry .
2. If your container registry does not have TLS, configure your environment to disable TLS for your registry.
3. Create a VM with the container disk as the disk source by using the web console or the command line.

IMPORTANT

You must install the QEMU guest agent on VMs created from operating system images that are not provided by Red Hat.

9.3.2.1. Building and uploading a container disk

You can build a virtual machine (VM) image into a container disk and upload it to a registry.

The size of a container disk is limited by the maximum layer size of the registry where the container disk is hosted.

NOTE

For Red Hat Quay, you can change the maximum layer size by editing the YAML configuration file that is created when Red Hat Quay is first deployed.

Prerequisites

- You must have **podman** installed.
- You must have a QCOW2 or RAW image file.

Procedure

1. Create a Dockerfile to build the VM image into a container image. The VM image must be owned by QEMU, which has a UID of **107**, and placed in the **/disk/** directory inside the container. Permissions for the **/disk/** directory must then be set to **0440**.
   The following example uses the Red Hat Universal Base Image (UBI) to handle these configuration changes in the first stage, and uses the minimal **scratch** image in the second stage to store the result:

```
$ 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 Where **<vm_image>** is the image in either QCOW2 or RAW format. If you use a remote image, replace **<vm_image>.qcow2** with the complete URL.

2. Build and tag the container:

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

3. Push the container image to the registry:

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

9.3.2.2. Disabling TLS for a container registry

You can disable TLS (transport layer security) for one or more container registries by editing the **insecureRegistries** field of the **HyperConverged** custom resource.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Open the **HyperConverged** CR in your default editor by running the following command:

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

2. Add a list of insecure registries to the **spec.storageImport.insecureRegistries** field.

Example HyperConverged custom resource

```
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 Replace the examples in this list with valid registry hostnames.

9.3.2.3. Creating a VM from a container disk by using the web console

You can create a virtual machine (VM) by importing a container disk from a container registry by using the OpenShift Container Platform web console.

Procedure

1. Navigate to **Virtualization → Catalog** in the web console.
2. Click a template tile without an available boot source.
3. Click **Customize VirtualMachine**.
4. On the **Customize template parameters** page, expand **Storage** and select **Registry (creates PVC)** from the **Disk source** list.
5. Enter the container image URL. Example: **https://mirror.arizona.edu/fedora/linux/releases/38/Cloud/x86_64/images/Fedora-Cloud-Base-38-1.6.x86_64.qcow2**
6. Set the disk size.
7. Click **Next**.
8. Click **Create VirtualMachine**.

9.3.2.4. Creating a VM from a container disk by using the CLI

You can create a virtual machine (VM) from a container disk by using the command line.

Prerequisites

- You must have access credentials for the container registry that contains the container disk.
- You have installed the **virtctl** CLI.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **VirtualMachine** manifest for your VM and save it as a YAML file. For example, to create a minimal Red Hat Enterprise Linux (RHEL) VM from a container disk, run the following command:

```
$ virtctl create vm --name vm-rhel-9 --instancetype u1.small --preference rhel.9 --volume-containerdisk src:registry.redhat.io/rhel9/rhel-guest-image:9.5
```

2. Review the **VirtualMachine** manifest for your VM:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-rhel-9 (1)
spec:
  instancetype:
```

```
    name: u1.small (2)
  preference:
    name: rhel.9 (3)
  runStrategy: Always
  template:
    metadata:
      creationTimestamp: null
    spec:
      domain:
        devices: {}
        resources: {}
      terminationGracePeriodSeconds: 180
      volumes:
      - containerDisk:
          image: registry.redhat.io/rhel9/rhel-guest-image:9.5 (4)
        name: vm-rhel-9-containerdisk-0
```

(1) The VM name.

(2) The instance type to use to control resource sizing of the VM.

(3) The preference to use.

(4) The URL of the container disk.

3. Create the VM by running the following command:

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

Verification

1. Monitor the status of the VM:

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

If the provisioning is successful, the VM status is **Running**:

Example output

```
NAME       AGE   STATUS    READY
vm-rhel-9   18s   Running   True
```

2. Verify that provisioning is complete and that the VM has started by accessing its serial console:

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

If the VM is running and the serial console is accessible, the output looks as follows:

Example output

```
Successfully connected to vm-rhel-9 console. The escape sequence is ^]
```

9.3.3. Creating VMs by cloning PVCs

You can create virtual machines (VMs) by cloning existing persistent volume claims (PVCs) with custom images.

You must install the QEMU guest agent on VMs created from operating system images that are not provided by Red Hat.

You clone a PVC by creating a data volume that references a source PVC.

9.3.3.1. About cloning

When cloning a data volume, the Containerized Data Importer (CDI) chooses one of the following Container Storage Interface (CSI) clone methods:

- CSI volume cloning
- Smart cloning

Both CSI volume cloning and smart cloning methods are efficient, but they have certain requirements for use. If the requirements are not met, the CDI uses host-assisted cloning. Host-assisted cloning is the slowest and least efficient method of cloning, but it has fewer requirements than either of the other two cloning methods.

9.3.3.1.1. CSI volume cloning

Container Storage Interface (CSI) cloning uses CSI driver features to more efficiently clone a source data volume.

CSI volume cloning has the following requirements:

- The CSI driver that backs the storage class of the persistent volume claim (PVC) must support volume cloning.
- For provisioners not recognized by the CDI, the corresponding storage profile must have the **cloneStrategy** set to CSI Volume Cloning.
- The source and target PVCs must have the same storage class and volume mode.
- If you create the data volume, you must have permission to create the **datavolumes/source** resource in the source namespace.
- The source volume must not be in use.

9.3.3.1.2. Smart cloning

When a Container Storage Interface (CSI) plugin with snapshot capabilities is available, the Containerized Data Importer (CDI) creates a persistent volume claim (PVC) from a snapshot, which then allows efficient cloning of additional PVCs.

Smart cloning has the following requirements:

- A snapshot class associated with the storage class must exist.
- The source and target PVCs must have the same storage class and volume mode.

- If you create the data volume, you must have permission to create the **datavolumes/source** resource in the source namespace.
- The source volume must not be in use.

9.3.3.1.3. Host-assisted cloning

When the requirements for neither Container Storage Interface (CSI) volume cloning nor smart cloning have been met, host-assisted cloning is used as a fallback method. Host-assisted cloning is less efficient than either of the two other cloning methods.

Host-assisted cloning uses a source pod and a target pod to copy data from the source volume to the target volume. The target persistent volume claim (PVC) is annotated with the fallback reason that explains why host-assisted cloning has been used, and an event is created.

Example PVC target annotation

```
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
```

Example event

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

9.3.3.2. Creating a VM from a PVC by using the web console

You can create a virtual machine (VM) by cloning a persistent volume claim (PVC) by using the OpenShift Container Platform web console.

Prerequisites

- You must have access to the namespace that contains the source PVC.

Procedure

1. Navigate to **Virtualization → Catalog** in the web console.
2. Click a template tile without an available boot source.
3. Click **Customize VirtualMachine**.
4. On the **Customize template parameters** page, expand **Storage** and select **PVC (clone PVC)** from the **Disk source** list.
5. Select the PVC project and the PVC name.
6. Set the disk size.

7. Click **Next**.

8. Click **Create VirtualMachine**.

9.3.3.3. Creating a VM from a PVC by using the CLI

You can create a virtual machine (VM) by cloning the persistent volume claim (PVC) of an existing VM by using the command line.

You can clone a PVC by using one of the following options:

- Cloning a PVC to a new data volume.
  This method creates a data volume whose lifecycle is independent of the original VM. Deleting the original VM does not affect the new data volume or its associated PVC.

- Cloning a PVC by creating a **VirtualMachine** manifest with a **dataVolumeTemplates** stanza.
  This method creates a data volume whose lifecycle is dependent on the original VM. Deleting the original VM deletes the cloned data volume and its associated PVC.

9.3.3.3.1. Optimizing clone Performance at scale in OpenShift Data Foundation

When you use OpenShift Data Foundation, the storage profile configures the default cloning strategy as **csi-clone**. However, this method has limitations, as shown in the following link. After a certain number of clones are created from a persistent volume claim (PVC), a background flattening process begins, which can significantly reduce clone creation performance at scale.

To improve performance when creating hundreds of clones from a single source PVC, use the **VolumeSnapshot** cloning method instead of the default **csi-clone** strategy.

Procedure

Create a **VolumeSnapshot** custom resource (CR) of the source image by using the following content:

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: golden-volumesnapshot
  namespace: golden-ns
spec:
  volumeSnapshotClassName: ocs-storagecluster-rbdplugin-snapclass
  source:
    persistentVolumeClaimName: golden-snap-source
```

1. Add the **spec.source.snapshot** stanza to reference the **VolumeSnapshot** as the source for the **DataVolume clone**:

```
spec:
  source:
    snapshot:
      namespace: golden-ns
      name: golden-volumesnapshot
```

Additional resources

- Setting a default cloning strategy using a storage profile

- Volume cloning
- CSI volume snapshots

9.3.3.3.2. Cloning a PVC to a data volume

You can clone the persistent volume claim (PVC) of an existing virtual machine (VM) disk to a data volume by using the command line.

You create a data volume that references the original source PVC. The lifecycle of the new data volume is independent of the original VM. Deleting the original VM does not affect the new data volume or its associated PVC.

Cloning between different volume modes is supported for host-assisted cloning, such as cloning from a block persistent volume (PV) to a file system PV, as long as the source and target PVs belong to the **kubevirt** content type.

NOTE

Smart-cloning is faster and more efficient than host-assisted cloning because it uses snapshots to clone PVCs. Smart-cloning is supported by storage providers that support snapshots, such as Red Hat OpenShift Data Foundation.

Cloning between different volume modes is not supported for smart-cloning.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- The VM with the source PVC must be powered down.
- If you clone a PVC to a different namespace, you must have permissions to create resources in the target namespace.
- Additional prerequisites for smart-cloning:
  - Your storage provider must support snapshots.
  - The source and target PVCs must have the same storage provider and volume mode.
  - The value of the **driver** key of the **VolumeSnapshotClass** object must match the value of the **provisioner** key of the **StorageClass** object as shown in the following example:

    Example VolumeSnapshotClass object

    ```
    kind: VolumeSnapshotClass
    apiVersion: snapshot.storage.k8s.io/v1
    driver: openshift-storage.rbd.csi.ceph.com
    # ...
    ```

    Example StorageClass object

    ```
    kind: StorageClass
    apiVersion: storage.k8s.io/v1
    # ...
    ```

```
provisioner: openshift-storage.rbd.csi.ceph.com
```

Procedure

1. Create a **DataVolume** manifest as shown in the following example:

```
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. Specify the name of the new data volume.
2. Specify the namespace of the source PVC.
3. Specify the name of the source PVC.

2. Create the data volume by running the following command:

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

NOTE

Data volumes prevent a VM from starting before the PVC is prepared. You can create a VM that references the new data volume while the PVC is being cloned.

9.3.3.3.3. Creating a VM from a cloned PVC by using a data volume template

You can create a virtual machine (VM) that clones the persistent volume claim (PVC) of an existing VM by using a data volume template. This method creates a data volume whose lifecycle is independent on the original VM.

Prerequisites

- The VM with the source PVC must be powered down.
- You have installed the **virtctl** CLI.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **VirtualMachine** manifest for your VM and save it as a YAML file, for example:

```
$ virtctl create vm --name rhel-9-clone --volume-import type:pvc,src:my-project/imported-volume-q5pr9
```

2. Review the **VirtualMachine** manifest for your VM:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: rhel-9-clone 1
spec:
  dataVolumeTemplates:
  - metadata:
      name: imported-volume-h4qn8
    spec:
      source:
        pvc:
          name: imported-volume-q5pr9 2
          namespace: my-project 3
      storage:
        resources: {}
  instancetype:
    inferFromVolume: imported-volume-h4qn8 4
    inferFromVolumeFailurePolicy: Ignore
  preference:
    inferFromVolume: imported-volume-h4qn8 5
    inferFromVolumeFailurePolicy: Ignore
  runStrategy: Always
  template:
    spec:
      domain:
        devices: {}
        memory:
          guest: 512Mi
        resources: {}
      terminationGracePeriodSeconds: 180
      volumes:
      - dataVolume:
          name: imported-volume-h4qn8
        name: imported-volume-h4qn8
```

1. The VM name.
2. The name of the source PVC.
3. The namespace of the source PVC.
4. If the PVC source has appropriate labels, the instance type is inferred from the selected **DataSource** object.
5. If the PVC source has appropriate labels, the preference is inferred from the selected **DataSource** object.

3. Create the virtual machine with the PVC-cloned data volume:

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

CHAPTER 10. MANAGING VMS

10.1. LISTING VIRTUAL MACHINES

You can list available virtual machines (VMs) by using the web console or the OpenShift CLI (**oc**).

10.1.1. Listing virtual machines by using the CLI

You can either list all of the virtual machines (VMs) in your cluster or limit the list to VMs in a specified namespace by using the OpenShift CLI (**oc**).

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- List all of the VMs in your cluster by running the following command:

```
$ oc get vms -A
```

- List all of the VMs in a specific namespace by running the following command:

```
$ oc get vms -n <namespace>
```

10.1.2. Listing virtual machines by using the web console

You can list all of the virtual machines (VMs) in your cluster by using the web console.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu to access the tree view with all of the projects and VMs in your cluster.
2. Optional: Enable the **Show only projects with VirtualMachines** option above the tree view to limit the displayed projects.
3. Optional: Click the **Advanced search** button next to the search bar to further filter VMs by one of the following: their name, the project they belong to, their labels, or the allocated vCPU and memory resources.

10.1.3. Organizing virtual machines by using the web console

In addition to creating virtual machines (VMs) in different projects, you can use the tree view to further organize them in folders.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu to access the tree view with all projects and VMs in your cluster.
2. Perform one of the following actions depending on your use case:

- To move the VM to a new folder in the same project:
  a. Right-click the name of the VM in the tree view.
  b. Select **Move to folder** from the menu.
  c. Type the name of the folder to create in the "Search folder" bar.
  d. Click **Create folder** in the drop-down list.
  e. Click **Save**.
- To move the VM to an existing folder in the same project:
  - Click the name of the VM in the tree view and drag it to a folder in the same project. If the operation is permitted, the folder is highlighted in green when you drag the VM over it.
- To move the VM from a folder to the project:
  - Click the name of the VM in the tree view and drag it on the project name. If the operation is permitted, the project name is highlighted in green when you drag the VM over it.

10.2. INSTALLING THE QEMU GUEST AGENT AND VIRTIO DRIVERS

The QEMU guest agent is a daemon that runs on the virtual machine (VM) and passes information to the host about the VM, users, file systems, and secondary networks.

You must install the QEMU guest agent on VMs created from operating system images that are not provided by Red Hat.

10.2.1. Installing the QEMU guest agent

10.2.1.1. Installing the QEMU guest agent on a Linux VM

The **qemu-guest-agent** is available by default in Red Hat Enterprise Linux (RHEL) virtual machines (VMs)

To create snapshots of a VM in the **Running** state with the highest integrity, install the QEMU guest agent.

The QEMU guest agent takes a consistent snapshot by attempting to quiesce the VM file system. This ensures that in-flight I/O is written to the disk before the snapshot is taken. If the guest agent is not present, quiescing is not possible and a best-effort snapshot is taken.

The conditions under which a snapshot is taken are reflected in the snapshot indications that are displayed in the web console or CLI. If these conditions do not meet your requirements, try creating the snapshot again, or use an offline snapshot

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Log in to the VM by using a console or SSH.
2. Install the QEMU guest agent by running the following command:

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

3. Ensure the service is persistent and start it:

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

Verification

- Run the following command to verify that **AgentConnected** is listed in the VM spec:

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

10.2.1.2. Installing the QEMU guest agent on a Windows VM

For Windows virtual machines (VMs), the QEMU guest agent is included in the VirtIO drivers. You can install the drivers during a Windows installation or on an existing Windows VM.

To create snapshots of a VM in the **Running** state with the highest integrity, install the QEMU guest agent.

The QEMU guest agent takes a consistent snapshot by attempting to quiesce the VM file system. This ensures that in-flight I/O is written to the disk before the snapshot is taken. If the guest agent is not present, quiescing is not possible and a best-effort snapshot is taken.

Note that in a Windows guest operating system, quiescing also requires the Volume Shadow Copy Service (VSS). Therefore, before you create a snapshot, ensure that VSS is enabled on the VM as well.

The conditions under which a snapshot is taken are reflected in the snapshot indications that are displayed in the web console or CLI. If these conditions do not meet your requirements, try creating the snapshot again or use an offline snapshot.

Procedure

1. In the Windows guest operating system, use the **File Explorer** to navigate to the **guest-agent** directory in the **virtio-win** CD drive.
2. Run the **qemu-ga-x86_64.msi** installer.

Verification

1. Obtain a list of network services by running the following command:

```
$ net start
```

2. Verify that the output contains the **QEMU Guest Agent**.

10.2.2. Installing VirtIO drivers on Windows VMs

VirtIO drivers are paravirtualized device drivers required for Microsoft Windows virtual machines (VMs) to run in OpenShift Virtualization. The drivers are shipped with the rest of the images and do not require a separate download.

The **container-native-virtualization/virtio-win** container disk must be attached to the VM as a SATA CD drive to enable driver installation. You can install VirtIO drivers during Windows installation or added to an existing Windows installation.

After the drivers are installed, the **container-native-virtualization/virtio-win** container disk can be removed from the VM.

Table 10.1. Supported drivers

Driver name	Hardware ID	Description
viostor	VEN_1AF4&DEV_1001 VEN_1AF4&DEV_1042	The block driver. Sometimes labeled as an **SCSI Controller** in the **Other devices** group.
viorng	VEN_1AF4&DEV_1005 VEN_1AF4&DEV_1044	The entropy source driver. Sometimes labeled as a **PCI Device** in the **Other devices** group.
NetKVM	VEN_1AF4&DEV_1000 VEN_1AF4&DEV_1041	The network driver. Sometimes labeled as an **Ethernet Controller** in the **Other devices** group. Available only if a VirtIO NIC is configured.

10.2.2.1. Attaching VirtIO container disk to Windows VMs during installation

You must attach the VirtIO container disk to the Windows VM to install the necessary Windows drivers. This can be done during creation of the VM.

Procedure

1. When creating a Windows VM from a template, click **Customize VirtualMachine**.
2. Select **Mount Windows drivers disk**.
3. Click the **Customize VirtualMachine parameters**.
4. Click **Create VirtualMachine**.

After the VM is created, the **virtio-win** SATA CD disk will be attached to the VM.

10.2.2.2. Attaching VirtIO container disk to an existing Windows VM

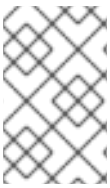
You must attach the VirtIO container disk to the Windows VM to install the necessary Windows drivers. This can be done to an existing VM.

Procedure

1. Navigate to the existing Windows VM, and click **Actions → Stop**.
2. Go to **VM Details → Configuration → Storage**.
3. Select the **Mount Windows drivers disk** checkbox.
4. Click **Save**.
5. Start the VM, and connect to a graphical console.

10.2.2.3. Installing VirtIO drivers during Windows installation

You can install the VirtIO drivers while installing Windows on a virtual machine (VM).



NOTE

This procedure uses a generic approach to the Windows installation and the installation method might differ between versions of Windows. See the documentation for the version of Windows that you are installing.

Prerequisites

- A storage device containing the **virtio** drivers must be attached to the VM.

Procedure

1. In the Windows operating system, use the **File Explorer** to navigate to the **virtio-win** CD drive.
2. Double-click the drive to run the appropriate installer for your VM.
   For a 64-bit vCPU, select the **virtio-win-gt-x64** installer. 32-bit vCPUs are no longer supported.
3. Optional: During the **Custom Setup** step of the installer, select the device drivers you want to install. The recommended driver set is selected by default.
4. After the installation is complete, select **Finish**.
5. Reboot the VM.

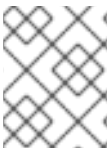
Verification

1. Open the system disk on the PC. This is typically **C:**.
2. Navigate to **Program Files → Virtio-Win**.

If the **Virtio-Win** directory is present and contains a sub-directory for each driver, the installation was successful.

10.2.2.4. Installing VirtIO drivers from a SATA CD drive on an existing Windows VM

You can install the VirtIO drivers from a SATA CD drive on an existing Windows virtual machine (VM).



NOTE

This procedure uses a generic approach to adding drivers to Windows. See the installation documentation for your version of Windows for specific installation steps.

Prerequisites

- A storage device containing the virtio drivers must be attached to the VM as a SATA CD drive.

Procedure

1. Start the VM and connect to a graphical console.
2. Log in to a Windows user session.
3. Open **Device Manager** and expand **Other devices** to list any **Unknown device**.
   a. Open the **Device Properties** to identify the unknown device.
   b. Right-click the device and select **Properties**.
   c. Click the **Details** tab and select **Hardware Ids** in the **Property** list.
   d. Compare the **Value** for the **Hardware Ids** with the supported VirtIO drivers.
4. Right-click the device and select **Update Driver Software**.
5. Click **Browse my computer for driver software** and browse to the attached SATA CD drive, where the VirtIO drivers are located. The drivers are arranged hierarchically according to their driver type, operating system, and CPU architecture.
6. Click **Next** to install the driver.
7. Repeat this process for all the necessary VirtIO drivers.
8. After the driver installs, click **Close** to close the window.
9. Reboot the VM to complete the driver installation.

10.2.2.5. Installing VirtIO drivers from a container disk added as a SATA CD drive

You can install VirtIO drivers from a container disk that you add to a Windows virtual machine (VM) as a SATA CD drive.

TIP

Downloading the **container-native-virtualization/virtio-win** container disk from the Red Hat Ecosystem Catalog is not mandatory, because the container disk is downloaded from the Red Hat registry if it not already present in the cluster. However, downloading reduces the installation time.

Prerequisites

- You must have access to the Red Hat registry or to the downloaded **container-native-virtualization/virtio-win** container disk in a restricted environment.
- You have installed the **virtctl** CLI.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Add the **container-native-virtualization/virtio-win** container disk as a CD drive by editing the **VirtualMachine** manifest:

```
# ...
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 boots the VM disks in the order defined in the **VirtualMachine** manifest. You can either define other VM disks that boot before the **container-native-virtualization/virtio-win** container disk or use the optional **bootOrder** parameter to ensure the VM boots from the correct disk. If you configure the boot order for a disk, you must configure the boot order for the other disks.

2. Apply the changes:
   - If the VM is not running, run the following command:

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

   - If the VM is running, reboot the VM or run the following command:

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

3. After the VM has started, install the VirtIO drivers from the SATA CD drive.

10.2.3. Updating VirtIO drivers

10.2.3.1. Updating VirtIO drivers on a Windows VM

Update the **virtio** drivers on a Windows virtual machine (VM) by using the Windows Update service.

Prerequisites

- The cluster must be connected to the internet. Disconnected clusters cannot reach the Windows Update service.

Procedure

1. In the Windows Guest operating system, click the **Windows** key and select **Settings**.
2. Navigate to **Windows Update → Advanced Options → Optional Updates**.
3. Install all updates from **Red Hat, Inc.**

4. Reboot the VM.

Verification

1. On the Windows VM, navigate to the **Device Manager**.
2. Select a device.
3. Select the **Driver** tab.
4. Click **Driver Details** and confirm that the **virtio** driver details displays the correct version.

10.3. CONNECTING TO VIRTUAL MACHINE CONSOLES

You can connect to the following consoles to access running virtual machines (VMs):

- VNC console
- Serial console
- Desktop viewer for Windows VMs

10.3.1. Connecting to the VNC console

You can connect to the VNC console of a virtual machine by using the OpenShift Container Platform web console or the **virtctl** command-line tool.

10.3.1.1. Connecting to the VNC console by using the web console

You can connect to the VNC console of a virtual machine (VM) by using the OpenShift Container Platform web console.

NOTE

If you connect to a Windows VM with a vGPU assigned as a mediated device, you can switch between the default display and the vGPU display.

Procedure

1. On the **Virtualization → VirtualMachines** page, click a VM to open the **VirtualMachine details** page.
2. Click the **Console** tab. The VNC console session starts automatically.
3. Optional: To switch to the vGPU display of a Windows VM, select **Ctl + Alt + 2**from the **Send key** list.
   - Select **Ctl + Alt + 1**from the **Send key** list to restore the default display.
4. To end the console session, click outside the console pane and then click **Disconnect**.

10.3.1.2. Connecting to the VNC console by using virtctl

You can use the **virtctl** command-line tool to connect to the VNC console of a running virtual machine.

NOTE

If you run the **virtctl vnc** command on a remote machine over an SSH connection, you must forward the X session to your local machine by running the **ssh** command with the **-X** or **-Y** flags.

Prerequisites

- You must install the **virt-viewer** package.

Procedure

1. Run the following command to start the console session:

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

2. If the connection fails, run the following command to collect troubleshooting information:

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

10.3.1.3. Generating a temporary token for the VNC console

To access the VNC of a virtual machine (VM), generate a temporary authentication bearer token for the Kubernetes API.

NOTE

Kubernetes also supports authentication using client certificates, instead of a bearer token, by modifying the curl command.

Prerequisites

- A running VM with OpenShift Virtualization 4.14 or later and **ssp-operator** 4.14 or later.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Set the **deployVmConsoleProxy** field value in the HyperConverged (**HCO**) custom resource (CR) to **true**:

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

2. Generate a token by entering the following command:

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

The **duration**

The **<duration>** parameter can be set in hours and minutes, with a minimum duration of 10 minutes. For example: **5h30m**. If this parameter is not set, the token is valid for 10 minutes by default.

Sample output:

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

3. Optional: Use the token provided in the output to create a variable:

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

You can now use the token to access the VNC console of a VM.

Verification

1. Log in to the cluster by entering the following command:

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

2. Test access to the VNC console of the VM by using the **virtctl** command:

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

WARNING

It is currently not possible to revoke a specific token.

To revoke a token, you must delete the service account that was used to create it. However, this also revokes all other tokens that were created by using the service account. Use the following command with caution:

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

Additional resources

- About the Scheduling, Scale, and Performance (SSP) Operator

10.3.1.3.1. Granting token generation permission for the VNC console by using the cluster role

As a cluster administrator, you can install a cluster role and bind it to a user or service account to allow access to the endpoint that generates tokens for the VNC console.

Procedure

- Choose to bind the cluster role to either a user or service account.

- Run the following command to bind the cluster role to a user:

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

- Run the following command to bind the cluster role to a service account:

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

10.3.2. Connecting to the serial console

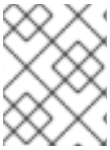
You can connect to the serial console of a virtual machine by using the OpenShift Container Platform web console or the **virtctl** command-line tool.

NOTE

Running concurrent VNC connections to a single virtual machine is not currently supported.

10.3.2.1. Connecting to the serial console by using the web console

You can connect to the serial console of a virtual machine (VM) by using the OpenShift Container Platform web console.



NOTE

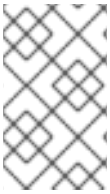
If you connect to a Windows VM with a vGPU assigned as a mediated device, you can switch between the default display and the vGPU display.

Procedure

1. On the **Virtualization → VirtualMachines** page, click a VM to open the **VirtualMachine details** page.
2. Click the **Console** tab. The VNC console session starts automatically.
3. Click **Disconnect** to end the VNC console session. Otherwise, the VNC console session continues to run in the background.
4. Select **Serial console** from the console list.
5. Optional: To switch to the vGPU display of a Windows VM, select **Ctl + Alt + 2**from the **Send key** list.
   - Select **Ctl + Alt + 1**from the **Send key** list to restore the default display.
6. To end the console session, click outside the console pane and then click **Disconnect**.

10.3.2.2. Connecting to the serial console by using virtctl

You can use the **virtctl** command-line tool to connect to the serial console of a running virtual machine.



NOTE

If you run the **virtctl vnc** command on a remote machine over an SSH connection, you must forward the X session to your local machine by running the **ssh** command with the **-X** or **-Y** flags.

Prerequisites

- You must install the **virt-viewer** package.

Procedure

1. Run the following command to start the console session:

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

2. Press **Ctrl+]** to end the console session.

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

3. If the connection fails, run the following command to collect troubleshooting information:

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

10.3.3. Connecting to the desktop viewer

You can connect to a Windows virtual machine (VM) by using the desktop viewer and the Remote Desktop Protocol (RDP).

10.3.3.1. Connecting to the desktop viewer by using the web console

You can connect to the desktop viewer of a virtual machine (VM) by using the OpenShift Container Platform web console. You can connect to the desktop viewer of a Windows virtual machine (VM) by using the OpenShift Container Platform web console.

NOTE

If you connect to a Windows VM with a vGPU assigned as a mediated device, you can switch between the default display and the vGPU display.

Prerequisites

- You installed the QEMU guest agent on the Windows VM.
- You have an RDP client installed.

Procedure

1. On the **Virtualization → VirtualMachines** page, click a VM to open the **VirtualMachine details** page.
2. Click the **Console** tab. The VNC console session starts automatically.

3. Click **Disconnect** to end the VNC console session. Otherwise, the VNC console session continues to run in the background.
4. Select **Desktop viewer** from the console list.
5. Click **Create RDP Service** to open the **RDP Service** dialog.
6. Select **Expose RDP Service** and click **Save** to create a node port service.
7. Click **Launch Remote Desktop** to download an **.rdp** file and launch the desktop viewer.
8. Optional: To switch to the vGPU display of a Windows VM, select **Ctl + Alt + 2** from the **Send key** list.
    - Select **Ctl + Alt + 1** from the **Send key** list to restore the default display.
9. To end the console session, click outside the console pane and then click **Disconnect**.

10.4. CONFIGURING SSH ACCESS TO VIRTUAL MACHINES

You can configure SSH access to virtual machines (VMs) by using the following methods:

- **virtctl ssh** command
  You create an SSH key pair, add the public key to a VM, and connect to the VM by running the **virtctl ssh** command with the private key.

  You can add public SSH keys to Red Hat Enterprise Linux (RHEL) 9 VMs at runtime or at first boot to VMs with guest operating systems that can be configured by using a cloud-init data source.
- **virtctl port-forward** command
  You add the **virtctl port-foward** command to your **.ssh/config** file and connect to the VM by using OpenSSH.
- Service
  You create a service, associate the service with the VM, and connect to the IP address and port exposed by the service.
- Secondary network
  You configure a secondary network, attach a virtual machine (VM) to the secondary network interface, and connect to the DHCP-allocated IP address.

10.4.1. Access configuration considerations

Each method for configuring access to a virtual machine (VM) has advantages and limitations, depending on the traffic load and client requirements.

NOTE

Services provide excellent performance and are recommended for applications that are accessed from outside the cluster.

If the internal cluster network cannot handle the traffic load, you can configure a secondary network.

virtctl ssh and virtctl port-forwarding commands

- Simple to configure.
- Recommended for troubleshooting VMs.
- **virtctl port-forwarding** recommended for automated configuration of VMs with Ansible.
- Dynamic public SSH keys can be used to provision VMs with Ansible.
- Not recommended for high-traffic applications like Rsync or Remote Desktop Protocol because of the burden on the API server.
- The API server must be able to handle the traffic load.
- The clients must be able to access the API server.
- The clients must have access credentials for the cluster.

Cluster IP service

- The internal cluster network must be able to handle the traffic load.
- The clients must be able to access an internal cluster IP address.

Node port service

- The internal cluster network must be able to handle the traffic load.
- The clients must be able to access at least one node.

Load balancer service

- A load balancer must be configured.
- Each node must be able to handle the traffic load of one or more load balancer services.

Secondary network

- Excellent performance because traffic does not go through the internal cluster network.
- Allows a flexible approach to network topology.
- Guest operating system must be configured with appropriate security because the VM is exposed directly to the secondary network. If a VM is compromised, an intruder could gain access to the secondary network.

10.4.2. Using virtctl ssh

You can add a public SSH key to a virtual machine (VM) and connect to the VM by running the **virtctl ssh** command.

This method is simple to configure. However, it is not recommended for high traffic loads because it places a burden on the API server.

10.4.2.1. About static and dynamic SSH key management

You can add public SSH keys to virtual machines (VMs) statically at first boot or dynamically at runtime.

NOTE

Only Red Hat Enterprise Linux (RHEL) 9 supports dynamic key injection.

Static SSH key management
You can add a statically managed SSH key to a VM with a guest operating system that supports configuration by using a cloud-init data source. The key is added to the virtual machine (VM) at first boot.

You can add the key by using one of the following methods:

- Add a key to a single VM when you create it by using the web console or the command line.
- Add a key to a project by using the web console. Afterwards, the key is automatically added to the VMs that you create in this project.

Use cases

- As a VM owner, you can provision all your newly created VMs with a single key.

Dynamic SSH key management
You can enable dynamic SSH key management for a VM with Red Hat Enterprise Linux (RHEL) 9 installed. Afterwards, you can update the key during runtime. The key is added by the QEMU guest agent, which is installed with Red Hat boot sources.

When dynamic key management is disabled, the default key management setting of a VM is determined by the image used for the VM.

Use cases

- Granting or revoking access to VMs: As a cluster administrator, you can grant or revoke remote VM access by adding or removing the keys of individual users from a **Secret** object that is applied to all VMs in a namespace.
- User access: You can add your access credentials to all VMs that you create and manage.
- Ansible provisioning:
  - As an operations team member, you can create a single secret that contains all the keys used for Ansible provisioning.
  - As a VM owner, you can create a VM and attach the keys used for Ansible provisioning.
- Key rotation:
  - As a cluster administrator, you can rotate the Ansible provisioner keys used by VMs in a namespace.
  - As a workload owner, you can rotate the key for the VMs that you manage.

10.4.2.2. Static key management

You can add a statically managed public SSH key when you create a virtual machine (VM) by using the OpenShift Container Platform web console or the command line. The key is added as a cloud-init data source when the VM boots for the first time.

You can also add a public SSH key to a project when you create a VM by using the web console. The key is saved as a secret and is added automatically to all VMs that you create.

NOTE

If you add a secret to a project and then delete the VM, the secret is retained because it is a namespace resource. You must delete the secret manually.

10.4.2.2.1. Adding a key when creating a VM from a template

You can add a statically managed public SSH key when you create a virtual machine (VM) by using the OpenShift Container Platform web console. The key is added to the VM as a cloud-init data source at first boot. This method does not affect cloud-init user data.

Optional: You can add a key to a project. Afterwards, this key is added automatically to VMs that you create in the project.

Prerequisites

- You generated an SSH key pair by running the **ssh-keygen** command.

Procedure

1. Navigate to **Virtualization → Catalog** in the web console.
2. Click a template tile.
   The guest operating system must support configuration from a cloud-init data source.
3. Click **Customize VirtualMachine**.
4. Click **Next**.
5. Click the **Scripts** tab.
6. If you have not already added a public SSH key to your project, click the edit icon beside **Authorized SSH key** and select one of the following options:
   - **Use existing**: Select a secret from the secrets list.
   - **Add new**:
     a. Browse to the SSH key file or paste the file in the key field.
     b. Enter the secret name.
     c. Optional: Select **Automatically apply this key to any new VirtualMachine you create in this project**.
7. Click **Save**.
8. Click **Create VirtualMachine**.
   The **VirtualMachine details** page displays the progress of the VM creation.

Verification

- Click the **Scripts** tab on the **Configuration** tab.
  The secret name is displayed in the **Authorized SSH key** section.

10.4.2.2.2. Creating a VM from an instance type by using the web console

You can create a virtual machine (VM) from an instance type by using the OpenShift Container Platform web console. You can also use the web console to create a VM by copying an existing snapshot or to clone a VM.

You can create a VM from a list of available bootable volumes. You can add Linux- or Windows-based volumes to the list.

You can add a statically managed SSH key when you create a virtual machine (VM) from an instance type by using the OpenShift Container Platform web console. The key is added to the VM as a cloud-init data source at first boot. This method does not affect cloud-init user data.

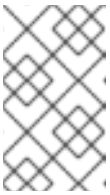
Procedure

1. In the web console, navigate to **Virtualization → Catalog**.
   The **InstanceTypes** tab opens by default.

NOTE

When configuring a downward-metrics device on an IBM Z® system that uses a VM preference, set the **spec.preference.name** value to **rhel.9.s390x** or another available preference with the format ***.s390x**.

2. Heterogeneous clusters only: To filter the bootable volumes using the options provided, click **Architecture**.
3. Select either of the following options:
   - Select a suitable bootable volume from the list. If the list is truncated, click the **Show all** button to display the entire list.



NOTE

The bootable volume table lists only those volumes in the **openshift-virtualization-os-images** namespace that have the **instancetype.kubevirt.io/default-preference** label.

     - Optional: Click the star icon to designate a bootable volume as a favorite. Starred bootable volumes appear first in the volume list.
   - Click **Add volume** to upload a new volume or to use an existing persistent volume claim (PVC), a volume snapshot, or a **containerDisk** volume. Click **Save**.
     Logos of operating systems that are not available in the cluster are shown at the bottom of the list. You can add a volume for the required operating system by clicking the **Add volume** link.

In addition, there is a link to the **Create a Windows bootable volume** quick start. The same link appears in a popover if you hover the pointer over the question mark icon next to the *Select volume to boot from* line.

Immediately after you install the environment or when the environment is disconnected, the list of volumes to boot from is empty. In that case, three operating system logos are displayed: Windows, RHEL, and Linux. You can add a new volume that meets your requirements by clicking the **Add volume** button.

4. Click an instance type tile and select the resource size appropriate for your workload. You can select huge pages for Red Hat-provided instance types of the **M** and **CX** series. Huge page options are identified by names that end with **1gi**.

5. Optional: Choose the virtual machine details, including the VM's name, that apply to the volume you are booting from:

   - For a Linux-based volume, follow these steps to configure SSH:

     a. If you have not already added a public SSH key to your project, click the edit icon beside **Authorized SSH key** in the **VirtualMachine details** section.

     b. Select one of the following options:

        - **Use existing**: Select a secret from the secrets list.
        - **Add new**: Follow these steps:

          i. Browse to the public SSH key file or paste the file in the key field.

          ii. Enter the secret name.

          iii. Optional: Select **Automatically apply this key to any new VirtualMachine you create in this project**.

     c. Click **Save**.

   - For a Windows volume, follow either of these set of steps to configure sysprep options:

     - If you have not already added sysprep options for the Windows volume, follow these steps:

       i. Click the edit icon beside **Sysprep** in the **VirtualMachine details** section.

       ii. Add the **Autoattend.xml** answer file.

       iii. Add the **Unattend.xml** answer file.

       iv. Click **Save**.

     - If you want to use existing sysprep options for the Windows volume, follow these steps:

       i. Click **Attach existing sysprep**.

       ii. Enter the name of the existing sysprep **Unattend.xml** answer file.

       iii. Click **Save**.

6. Optional: If you are creating a Windows VM, you can mount a Windows driver disk:

a. Click the **Customize VirtualMachine** button.

b. On the **VirtualMachine details** page, click **Storage**.

c. Select the **Mount Windows drivers disk** checkbox.

7. Optional: Click **View YAML & CLI** to view the YAML file. Click **CLI** to view the CLI commands. You can also download or copy either the YAML file contents or the CLI commands.

8. Click **Create VirtualMachine**.

After the VM is created, you can monitor the status on the **VirtualMachine details** page.

10.4.2.2.3. Adding a key when creating a VM by using the CLI

You can add a statically managed public SSH key when you create a virtual machine (VM) by using the command line. The key is added to the VM at first boot.

The key is added to the VM as a cloud-init data source. This method separates the access credentials from the application data in the cloud-init user data. This method does not affect cloud-init user data.

Prerequisites

- You generated an SSH key pair by running the **ssh-keygen** command.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a manifest file for a **VirtualMachine** object and a **Secret** object:

Example manifest

```
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
  runStrategy: Always
  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 Specify the **cloudInitNoCloud** data source.

2 Specify the **Secret** object name.

3 Paste the public SSH key.

2. Create the **VirtualMachine** and **Secret** objects by running the following command:

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

3. Start the VM by running the following command:

```
$ virtctl start vm example-vm -n example-namespace
```

Verification

- Get the VM configuration:

```
$ oc describe vm example-vm -n example-namespace
```

Example output

```
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
# ...
```

10.4.2.3. Dynamic key management

You can enable dynamic key injection for a virtual machine (VM) by using the OpenShift Container Platform web console or the command line. Then, you can update the key at runtime.

NOTE

Only Red Hat Enterprise Linux (RHEL) 9 supports dynamic key injection.

If you disable dynamic key injection, the VM inherits the key management method of the image from which it was created.

10.4.2.3.1. Enabling dynamic key injection when creating a VM from a template

You can enable dynamic public SSH key injection when you create a virtual machine (VM) from a template by using the OpenShift Container Platform web console. Then, you can update the key at runtime.

NOTE

Only Red Hat Enterprise Linux (RHEL) 9 supports dynamic key injection.

The key is added to the VM by the QEMU guest agent, which is installed with RHEL 9.

Prerequisites

- You generated an SSH key pair by running the **ssh-keygen** command.

Procedure

1. Navigate to **Virtualization → Catalog** in the web console.
2. Click the **Red Hat Enterprise Linux 9 VM**tile.
3. Click **Customize VirtualMachine**.
4. Click **Next**.
5. Click the **Scripts** tab.
6. If you have not already added a public SSH key to your project, click the edit icon beside **Authorized SSH key** and select one of the following options:

- **Use existing**: Select a secret from the secrets list.
- **Add new**:
  a. Browse to the SSH key file or paste the file in the key field.
  b. Enter the secret name.
  c. Optional: Select **Automatically apply this key to any new VirtualMachine you create in this project**.

7. Set **Dynamic SSH key injection** to on.
8. Click **Save**.
9. Click **Create VirtualMachine**.
   The **VirtualMachine details** page displays the progress of the VM creation.

Verification

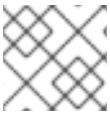
- Click the **Scripts** tab on the **Configuration** tab.
  The secret name is displayed in the **Authorized SSH key** section.

10.4.2.3.2. Creating a VM from an instance type by using the web console

You can create a virtual machine (VM) from an instance type by using the OpenShift Container Platform web console. You can also use the web console to create a VM by copying an existing snapshot or to clone a VM.

You can create a VM from a list of available bootable volumes. You can add Linux- or Windows-based volumes to the list.

You can enable dynamic SSH key injection when you create a virtual machine (VM) from an instance type by using the OpenShift Container Platform web console. Then, you can add or revoke the key at runtime.



NOTE

Only Red Hat Enterprise Linux (RHEL) 9 supports dynamic key injection.

The key is added to the VM by the QEMU guest agent, which is installed with RHEL 9.

Procedure

1. In the web console, navigate to **Virtualization → Catalog**.
   The **InstanceTypes** tab opens by default.

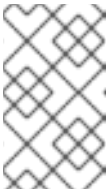
NOTE

When configuring a downward-metrics device on an IBM Z® system that uses a VM preference, set the **spec.preference.name** value to **rhel.9.s390x** or another available preference with the format ***.s390x**.

2. Heterogeneous clusters only: To filter the bootable volumes using the options provided, click **Architecture**.

3. Select either of the following options:

   - Select a suitable bootable volume from the list. If the list is truncated, click the **Show all** button to display the entire list.

     NOTE

     The bootable volume table lists only those volumes in the **openshift-virtualization-os-images** namespace that have the **instancetype.kubevirt.io/default-preference** label.

     - Optional: Click the star icon to designate a bootable volume as a favorite. Starred bootable volumes appear first in the volume list.

   - Click **Add volume** to upload a new volume or to use an existing persistent volume claim (PVC), a volume snapshot, or a **containerDisk** volume. Click **Save**.
     Logos of operating systems that are not available in the cluster are shown at the bottom of the list. You can add a volume for the required operating system by clicking the **Add volume** link.

     In addition, there is a link to the **Create a Windows bootable volume** quick start. The same link appears in a popover if you hover the pointer over the question mark icon next to the *Select volume to boot from* line.

     Immediately after you install the environment or when the environment is disconnected, the list of volumes to boot from is empty. In that case, three operating system logos are displayed: Windows, RHEL, and Linux. You can add a new volume that meets your requirements by clicking the **Add volume** button.

4. Click an instance type tile and select the resource size appropriate for your workload. You can select huge pages for Red Hat-provided instance types of the **M** and **CX** series. Huge page options are identified by names that end with **1gi**.

5. Click the **Red Hat Enterprise Linux 9 VM** tile.

6. Optional: Choose the virtual machine details, including the VM's name, that apply to the volume you are booting from:

   - For a Linux-based volume, follow these steps to configure SSH:

     a. If you have not already added a public SSH key to your project, click the edit icon beside **Authorized SSH key** in the **VirtualMachine details** section.

     b. Select one of the following options:

        - **Use existing**: Select a secret from the secrets list.
        - **Add new**: Follow these steps:

          i. Browse to the public SSH key file or paste the file in the key field.

          ii. Enter the secret name.

iii. Optional: Select **Automatically apply this key to any new VirtualMachine you create in this project**.

c. Click **Save**.

- For a Windows volume, follow either of these set of steps to configure sysprep options:
  - If you have not already added sysprep options for the Windows volume, follow these steps:
    i. Click the edit icon beside **Sysprep** in the **VirtualMachine details** section.
    ii. Add the **Autoattend.xml** answer file.
    iii. Add the **Unattend.xml** answer file.
    iv. Click **Save**.
  - If you want to use existing sysprep options for the Windows volume, follow these steps:
    i. Click **Attach existing sysprep**.
    ii. Enter the name of the existing sysprep **Unattend.xml** answer file.
    iii. Click **Save**.

7. Set **Dynamic SSH key injection** in the **VirtualMachine details** section to on.
8. Optional: If you are creating a Windows VM, you can mount a Windows driver disk:
   a. Click the **Customize VirtualMachine** button.
   b. On the **VirtualMachine details** page, click **Storage**.
   c. Select the **Mount Windows drivers disk** checkbox.
9. Optional: Click **View YAML & CLI** to view the YAML file. Click **CLI** to view the CLI commands. You can also download or copy either the YAML file contents or the CLI commands.
10. Click **Create VirtualMachine**.

After the VM is created, you can monitor the status on the **VirtualMachine details** page.

10.4.2.3.3. Enabling dynamic SSH key injection by using the web console

You can enable dynamic key injection for a virtual machine (VM) by using the OpenShift Container Platform web console. Then, you can update the public SSH key at runtime.

The key is added to the VM by the QEMU guest agent, which is installed with Red Hat Enterprise Linux (RHEL) 9.

Prerequisites

- The guest operating system is RHEL 9.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.

2. Select a VM to open the **VirtualMachine details** page.
3. On the **Configuration** tab, click **Scripts**.
4. If you have not already added a public SSH key to your project, click the edit icon beside **Authorized SSH key** and select one of the following options:
   - **Use existing**: Select a secret from the secrets list.
   - **Add new**:
     a. Browse to the SSH key file or paste the file in the key field.
     b. Enter the secret name.
     c. Optional: Select **Automatically apply this key to any new VirtualMachine you create in this project**.
5. Set **Dynamic SSH key injection** to on.
6. Click **Save**.

10.4.2.3.4. Enabling dynamic key injection by using the CLI

You can enable dynamic key injection for a virtual machine (VM) by using the command line. Then, you can update the public SSH key at runtime.

NOTE

Only Red Hat Enterprise Linux (RHEL) 9 supports dynamic key injection.

The key is added to the VM by the QEMU guest agent, which is installed automatically with RHEL 9.

Prerequisites

- You generated an SSH key pair by running the **ssh-keygen** command.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a manifest file for a **VirtualMachine** object and a **Secret** object:

Example manifest

```
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
  runStrategy: Always
  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) Specify the **cloudInitNoCloud** data source.

(2) Specify the **Secret** object name.

(3) Paste the public SSH key.

2. Create the **VirtualMachine** and **Secret** objects by running the following command:

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

3. Start the VM by running the following command:

```
$ virtctl start vm example-vm -n example-namespace
```

Verification

- Get the VM configuration:

```
$ oc describe vm example-vm -n example-namespace
```

Example output

```
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
# ...
```

10.4.2.4. Using the virtctl ssh command

You can access a running virtual machine (VM) by using the **virtcl ssh** command.

Prerequisites

- You installed the **virtctl** command-line tool.
- You added a public SSH key to the VM.
- You have an SSH client installed.
- The environment where you installed the **virtctl** tool has the cluster permissions required to access the VM. For example, you ran **oc login** or you set the **KUBECONFIG** environment variable.

Procedure

- Run the **virtctl ssh** command:

```
$ virtctl -n <namespace> ssh <username>@example-vm -i <ssh_key> 1
```

1 Specify the namespace, user name, and the SSH private key. The default SSH key location is **/home/user/.ssh**. If you save the key in a different location, you must specify the path.

Example

```
$ virtctl -n my-namespace ssh cloud-user@example-vm -i my-key
```

TIP

You can copy the **virtctl ssh** command in the web console by selecting **Copy SSH command** from the options menu beside a VM on the **VirtualMachines** page.

Alternatively, right-click the VM in the tree view and select **Copy SSH command** from the pop-up menu to copy the **virtctl ssh** command.

10.4.3. Using the virtctl port-forward command

You can use your local OpenSSH client and the **virtctl port-forward** command to connect to a running virtual machine (VM). You can use this method with Ansible to automate the configuration of VMs.

This method is recommended for low-traffic applications because port-forwarding traffic is sent over the control plane. This method is not recommended for high-traffic applications such as Rsync or Remote Desktop Protocol because it places a heavy burden on the API server.

Prerequisites

- You have installed the **virtctl** client.
- The virtual machine you want to access is running.
- The environment where you installed the **virtctl** tool has the cluster permissions required to access the VM. For example, you ran **oc login** or you set the **KUBECONFIG** environment variable.

Procedure

1. Add the following text to the **~/.ssh/config** file on your client machine:

```
Host vm/*
  ProxyCommand virtctl port-forward --stdio=true %h %p
```

2. Connect to the VM by running the following command:

```
$ ssh <user>@vm/<vm_name>.<namespace>
```

10.4.4. Using a service for SSH access

You can create a service for a virtual machine (VM) and connect to the IP address and port exposed by the service.

NOTE

Services provide excellent performance and are recommended for applications that are accessed from outside the cluster or within the cluster. Ingress traffic is protected by firewalls.

If the cluster network cannot handle the traffic load, consider using a secondary network for VM access.

10.4.4.1. About services

A Kubernetes service exposes network access for clients to an application running on a set of pods. Services offer abstraction, load balancing, and, in the case of the **NodePort** and **LoadBalancer** types, exposure to the outside world.

ClusterIP

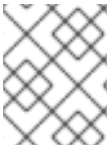
Exposes the service on an internal IP address and as a DNS name to other applications within the cluster. A single service can map to multiple virtual machines. When a client tries to connect to the service, the client's request is load balanced among available backends. **ClusterIP** is the default service type.

NodePort

Exposes the service on the same port of each selected node in the cluster. **NodePort** makes a port accessible from outside the cluster, as long as the node itself is externally accessible to the client.

LoadBalancer

Creates an external load balancer in the current cloud (if supported) and assigns a fixed, external IP address to the service.



NOTE

For on-premise clusters, you can configure a load-balancing service by deploying the MetalLB Operator.

10.4.4.2. Creating a service

You can create a service to expose a virtual machine (VM) by using the OpenShift Container Platform web console, **virtctl** command-line tool, or a YAML file.

10.4.4.2.1. Enabling load balancer service creation by using the web console

You can enable the creation of load balancer services for a virtual machine (VM) by using the OpenShift Container Platform web console.

Prerequisites

- You have configured a load balancer for the cluster.
- You are logged in as a user with the **cluster-admin** role.
- You created a network attachment definition for the network.

Procedure

1. Navigate to **Virtualization → Overview**.
2. On the **Settings** tab, click **Cluster**.
3. Expand **General settings** and **SSH configuration**.
4. Set **SSH over LoadBalancer service** to on.

10.4.4.2.2. Creating a service by using the web console

You can create a node port or load balancer service for a virtual machine (VM) by using the OpenShift Container Platform web console.

Prerequisites

- You configured the cluster network to support either a load balancer or a node port.
- To create a load balancer service, you enabled the creation of load balancer services.

Procedure

1. Navigate to **VirtualMachines** and select a virtual machine to view the **VirtualMachine details** page.
2. On the **Details** tab, select **SSH over LoadBalancer** from the **SSH service type** list.
3. Optional: Click the copy icon to copy the **SSH** command to your clipboard.

Verification

- Check the **Services** pane on the **Details** tab to view the new service.

10.4.4.2.3. Creating a service by using virtctl

You can create a service for a virtual machine (VM) by using the **virtctl** command-line tool.

Prerequisites

- You installed the **virtctl** command-line tool.
- You configured the cluster network to support the service.
- The environment where you installed **virtctl** has the cluster permissions required to access the VM. For example, you ran **oc login** or you set the **KUBECONFIG** environment variable.

Procedure

- Create a service by running the following command:

```
$ virtctl expose vm <vm_name> --name <service_name> --type <service_type> --port <port>
```
(1)

(1) Specify the **ClusterIP**, **NodePort**, or **LoadBalancer** service type.

Example

```
$ virtctl expose vm example-vm --name example-service --type NodePort --port 22
```

Verification

- Verify the service by running the following command:

```
$ oc get service
```

Next steps

After you create a service with **virtctl**, you must add **special: key** to the **spec.template.metadata.labels** stanza of the **VirtualMachine** manifest. See Creating a service by using the command line .

10.4.4.2.4. Creating a service by using the CLI

You can create a service and associate it with a virtual machine (VM) by using the command line.

Prerequisites

- You configured the cluster network to support the service.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **VirtualMachine** manifest to add the label for service creation:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  runStrategy: Halted
  template:
    metadata:
      labels:
        special: key 1
# ...
```

1 Add **special: key** to the **spec.template.metadata.labels** stanza.

NOTE

Labels on a virtual machine are passed through to the pod. The **special: key** label must match the label in the **spec.selector** attribute of the **Service** manifest.

2. Save the **VirtualMachine** manifest file to apply your changes.
3. Create a **Service** manifest to expose the VM:

```
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) Specify the label that you added to the **spec.template.metadata.labels** stanza of the **VirtualMachine** manifest.

(2) Specify **ClusterIP**, **NodePort**, or **LoadBalancer**.

(3) Specifies a collection of network ports and protocols that you want to expose from the virtual machine.

4. Save the **Service** manifest file.
5. Create the service by running the following command:

```
$ oc create -f example-service.yaml
```

6. Restart the VM to apply the changes.

Verification

- Query the **Service** object to verify that it is available:

```
$ oc get service -n example-namespace
```

10.4.4.3. Connecting to a VM exposed by a service by using SSH

You can connect to a virtual machine (VM) that is exposed by a service by using SSH.

Prerequisites

- You created a service to expose the VM.
- You have an SSH client installed.
- You are logged in to the cluster.

Procedure

- Run the following command to access the VM:

```
$ ssh <user_name>@<ip_address> -p <port> (1)
```

(1) Specify the cluster IP for a cluster IP service, the node IP for a node port service, or the external IP address for a load balancer service.

10.4.5. Using a secondary network for SSH access

You can configure a secondary network, attach a virtual machine (VM) to the secondary network interface, and connect to the DHCP-allocated IP address by using SSH.

IMPORTANT

Secondary networks provide excellent performance because the traffic is not handled by the cluster network stack. However, the VMs are exposed directly to the secondary network and are not protected by firewalls. If a VM is compromised, an intruder could gain access to the secondary network. You must configure appropriate security within the operating system of the VM if you use this method.

See the Multus and SR-IOV documentation in the OpenShift Virtualization Tuning & Scaling Guide for additional information about networking options.

Prerequisites

- You configured a secondary network such as Linux bridge or SR-IOV.
- You created a network attachment definition for a Linux bridge network or the SR-IOV Network Operator created a network attachment definition when you created an **SriovNetwork** object.

10.4.5.1. Configuring a VM network interface by using the web console

You can configure a network interface for a virtual machine (VM) by using the OpenShift Container Platform web console.

Prerequisites

- You created a network attachment definition for the network.

Procedure

1. Navigate to **Virtualization → VirtualMachines**.
2. Click a VM to view the **VirtualMachine details** page.
3. On the **Configuration** tab, click the **Network interfaces** tab.
4. Click **Add network interface**.
5. Enter the interface name and select the network attachment definition from the **Network** list.
6. Click **Save**.
7. Restart or live migrate the VM to apply the changes.

10.4.5.2. Connecting to a VM attached to a secondary network by using SSH

You can connect to a virtual machine (VM) attached to a secondary network by using SSH.

Prerequisites

- You attached a VM to a secondary network with a DHCP server.
- You have an SSH client installed.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Obtain the IP address of the VM by running the following command:

```
$ oc describe vm <vm_name> -n <namespace>
```

Example output

```
# ...
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. Connect to the VM by running the following command:

```
$ ssh <user_name>@<ip_address> -i <ssh_key>
```

Example

```
$ ssh cloud-user@10.244.0.37 -i ~/.ssh/id_rsa_cloud-user
```

NOTE

You can also access a VM attached to a secondary network interface by using the cluster FQDN.

10.5. EDITING VIRTUAL MACHINES

You can update a virtual machine (VM) configuration by using the OpenShift Container Platform web console. You can update the YAML file or the **VirtualMachine details** page.

You can also edit a VM by using the command line.

To edit a VM to configure disk sharing by using virtual disks or LUN, see Configuring shared volumes for virtual machines.

10.5.1. Changing the instance type of a VM by using the web console

You can change the instance type associated with a running virtual machine (VM) by using the web console. The change takes effect immediately.

Prerequisites

- You created the VM by using an instance type.

Procedure

1. In the OpenShift Container Platform web console, click **Virtualization → VirtualMachines**.
2. Select a VM to open the **VirtualMachine details** page.
3. Click the **Configuration** tab.
4. On the **Details** tab, click the instance type text to open the **Edit Instancetype** dialog. For example, click **1 CPU | 2 GiB Memory**.
5. Edit the instance type by using the **Series** and **Size** lists.
   a. Select an item from the **Series** list to show the relevant sizes for that series. For example, select **General Purpose**.
   b. Select the VM's new instance type from the **Size** list. For example, select **medium: 1 CPUs, 4Gi Memory**, which is available in the **General Purpose** series.
6. Click **Save**.

Verification

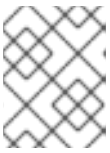
1. Click the **YAML** tab.
2. Click **Reload**.
3. Review the VM YAML to confirm that the instance type changed.

10.5.2. Hot plugging memory on a virtual machine

You can add or remove the amount of memory allocated to a virtual machine (VM) without having to restart the VM by using the OpenShift Container Platform web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines**.
2. Select the required VM to open the **VirtualMachine details** page.
3. On the **Configuration** tab, click **Edit CPU|Memory**.
4. Enter the desired amount of memory and click **Save**.



NOTE

You can hot plug up to three times the default initial amount of memory of the VM. Exceeding this limit requires a restart.

The system applies these changes immediately. If the VM is migratable, a live migration is triggered. If not, or if the changes cannot be live-updated, a **RestartRequired** condition is added to the VM.

NOTE

Memory hot plugging for virtual machines requires guest operating system support for the **virtio-mem** driver. This support depends on the driver being included and enabled within the guest operating system, not on specific upstream kernel versions.

Supported guest operating systems:

- RHEL 9.4 and later
- RHEL 8.10 and later (hot-unplug is disabled by default)
- Other Linux guests require kernel version 5.16 or later and the **virtio-mem** kernel module
- Windows guests require **virtio-mem** driver version 100.95.104.26200 or later

10.5.3. Hot plugging CPUs on a virtual machine

You can increase or decrease the number of CPU sockets allocated to a virtual machine (VM) without having to restart the VM by using the OpenShift Container Platform web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines**.
2. Select the required VM to open the **VirtualMachine details** page.
3. On the **Configuration** tab, click **Edit CPU|Memory**.
4. Select the **vCPU** radio button.
5. Enter the desired number of vCPU sockets and click **Save**.

NOTE

You can hot plug up to three times the default initial number of vCPU sockets of the VM. Exceeding this limit requires a restart.

If the VM is migratable, a live migration is triggered. If not, or if the changes cannot be live-updated, a **RestartRequired** condition is added to the VM.

10.5.4. Editing a virtual machine by using the CLI

You can edit a virtual machine (VM) by using the command line.

Prerequisites

- You installed the **oc** CLI.

Procedure

1. Obtain the virtual machine configuration by running the following command:

```
$ oc edit vm <vm_name>
```

2. Edit the YAML configuration.
3. If you edit a running virtual machine, you need to do one of the following:
   - Restart the virtual machine.
   - Run the following command for the new configuration to take effect:

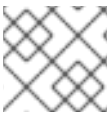
```
$ oc apply vm <vm_name> -n <namespace>
```

10.5.5. Adding a disk to a virtual machine

You can add a virtual disk to a virtual machine (VM) by using the OpenShift Container Platform web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Select a VM to open the **VirtualMachine details** page.
3. On the **Disks** tab, click **Add disk**.
4. Specify the **Source**, **Name**, **Size**, **Type**, **Interface**, and **Storage Class**.
   a. Optional: You can enable preallocation if you use a blank disk source and require maximum write performance when creating data volumes. To do so, select the **Enable preallocation** checkbox.
   b. Optional: You can clear **Apply optimized StorageProfile settings** to change the **Volume Mode** and **Access Mode** for the virtual disk. If you do not specify these parameters, the system uses the default values from the **kubevirt-storage-class-defaults** config map.
5. Click **Add**.



NOTE

If the VM is running, you must restart the VM to apply the change.

10.5.5.1. Storage fields

Field	Description
Blank (creates PVC)	Create an empty disk.
Import via URL (creates PVC)	Import content via URL (HTTP or HTTPS endpoint).

Field	Description
Use an existing PVC	Use a PVC that is already available in the cluster.
Clone existing PVC (creates PVC)	Select an existing PVC available in the cluster and clone it.
Import via Registry (creates PVC)	Import content via container registry.
Container (ephemeral)	Upload content from a container located in a registry accessible from the cluster. The container disk should be used only for read-only filesystems such as CD-ROMs or temporary virtual machines.
Name	Name of the disk. The name can contain lowercase letters (**a-z**), numbers (**0-9**), hyphens (**-**), and periods (**.**), up to a maximum of 253 characters. The first and last characters must be alphanumeric. The name must not contain uppercase letters, spaces, or special characters.
Size	Size of the disk in GiB.
Type	Type of disk. Example: Disk or CD-ROM
Interface	Type of disk device. Supported interfaces are **virtIO**, **SATA**, and **SCSI**.
Storage Class	The storage class that is used to create the disk.

Advanced storage settings
The following advanced storage settings are optional and available for **Blank**, **Import via URL**, and **Clone existing PVC** disks.

If you do not specify these parameters, the system uses the default storage profile values.

Parameter	Option	Parameter description
Volume Mode	Filesystem	Stores the virtual disk on a file system-based volume.
	Block	Stores the virtual disk directly on the block volume. Only use **Block** if the underlying storage supports it.
Access Mode	ReadWriteOnce (RWO)	Volume can be mounted as read-write by a single node.

Parameter	Option	Parameter description
	ReadWriteMany (RWX)	Volume can be mounted as read-write by many nodes at one time.  **NOTE** This mode is required for live migration.

10.5.6. Mounting a Windows driver disk on a virtual machine

You can mount a Windows driver disk on a virtual machine (VM) by using the OpenShift Container Platform web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines**.
2. Select the required VM to open the **VirtualMachine details** page.
3. On the **Configuration** tab, click **Storage**.
4. Select the **Mount Windows drivers disk** checkbox.
   The Windows driver disk is displayed in the list of mounted disks.

10.5.7. Adding a secret, config map, or service account to a virtual machine

You add a secret, config map, or service account to a virtual machine by using the OpenShift Container Platform web console.

These resources are added to the virtual machine as disks. You then mount the secret, config map, or service account as you would mount any other disk.

If the virtual machine is running, changes do not take effect until you restart the virtual machine. The newly added resources are marked as pending changes at the top of the page.

Prerequisites

- The secret, config map, or service account that you want to add must exist in the same namespace as the target virtual machine.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click **Configuration → Environment**.
4. Click **Add Config Map, Secret or Service Account**

5. Click **Select a resource** and select a resource from the list. A six character serial number is automatically generated for the selected resource.
6. Optional: Click **Reload** to revert the environment to its last saved state.
7. Click **Save**.

Verification

1. On the **VirtualMachine details** page, click **Configuration → Disks** and verify that the resource is displayed in the list of disks.
2. Restart the virtual machine by clicking **Actions → Restart**.

You can now mount the secret, config map, or service account as you would mount any other disk.

10.5.8. Updating multiple virtual machines

You can use the command line interface (CLI) to update multiple virtual machines (VMs) at the same time.

Prerequisites

- You installed the **oc** CLI.
- You have access to the OpenShift Container Platform cluster, and you have **cluster-admin** permissions.

Procedure

1. Create a privileged service account by running the following commands:

```
$ oc adm new-project kubevirt-api-lifecycle-automation
```

```
$ oc create sa kubevirt-api-lifecycle-automation -n kubevirt-api-lifecycle-automation
```

```
$ oc create clusterrolebinding kubevirt-api-lifecycle-automation --clusterrole=cluster-admin --
serviceaccount=kubevirt-api-lifecycle-automation:kubevirt-api-lifecycle-automation
```

2. Determine the pull URL for the **kubevirt-api-lifecycle** image by running the following command:

```
$ oc get csv -n openshift-cnv -l=operators.coreos.com/kubevirt-hyperconverged.openshift-
cnv -ojson | jq '.items[0].spec.relatedImages[] | select(.name|test(".*kubevirt-api-lifecycle-
automation.*")) | .image'
```

3. Deploy **Kubevirt-Api-Lifecycle-Automation** by creating a job object as shown in the following example:

```
apiVersion: batch/v1
kind: Job
metadata:
 name: kubevirt-api-lifecycle-automation
```

```
 namespace: kubevirt-api-lifecycle-automation
spec:
 template:
  spec:
   containers:
   - name: kubevirt-api-lifecycle-automation
     image: quay.io/openshift-virtualization/kubevirt-api-lifecycle-automation:v4.20 (1)
     imagePullPolicy: Always
     env:
     - name: MACHINE_TYPE_GLOB (2)
       value: smth-glob9.10.0
     - name: RESTART_REQUIRED (3)
       value: "true"
     - name: NAMESPACE (4)
       value: "default"
     - name: LABEL_SELECTOR (5)
       value: my-vm
     securityContext:
      allowPrivilegeEscalation: false
      capabilities:
       drop:
       - ALL
      privileged: false
      runAsNonRoot: true
      seccompProfile:
       type: RuntimeDefault
   restartPolicy: Never
   serviceAccountName: kubevirt-api-lifecycle-automation
```

(1) Replace the image value with your pull URL for the image.

(2) Replace the **MACHINE_TYPE_GLOB** value with your own pattern. This pattern is used to detect deprecated machine types that need to be upgraded.

(3) If the **RESTART_REQUIRED** emvironment variable is set to **true**, VMs are restarted after the machine type is updated. If you do not want VMs to be restarted, set the value to **false**.

(4) The **namespace** environment value indicates the namespace to look for VMs in. Leave the parameter empty for the job to go over all namespaces in the cluster.

(5) You can use the **LABEL_SELECTOR** environment variable to select VMs that receive the job action. If you want the job to go over all VMs in the cluster, do not assign a value to the parameter.

10.5.8.1. Performing bulk actions on virtual machines

You can perform bulk actions on multiple virtual machines (VMs) simultaneously by using the **VirtualMachines** list view in the web console. This allows you to efficiently manage a group of VMs with minimal manual effort.

Available bulk actions

- **Label VMs** - Add, edit, or remove labels that are applied across selected VMs.

- **Delete VMs** - Select multiple VMs to delete. The confirmation dialog displays the number of VMs selected for deletion.
- **Move VMs to folder** - Move selected VMs to a folder. All VMs must belong to the same namespace.

10.5.9. Configuring multiple IOThreads for fast storage access

You can improve storage performance by configuring multiple IOThreads for a virtual machine (VM) that uses fast storage, such as solid-state drive (SSD) or non-volatile memory express (NVMe). This configuration option is only available by editing YAML of the VM.

NOTE

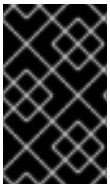
Multiple IOThreads are supported only when **blockMultiQueue** is enabled and the disk bus is set to **virtio**. You must set this configuration for the configuration to work correctly.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **YAML** tab to open the VM manifest.
4. In the YAML editor, locate the **spec.template.spec.domain** section and add or modify the following fields:

```
domain:
  ioThreadsPolicy: supplementalPool
  ioThreads:
    supplementalPoolThreadCount: 4
  devices:
    blockMultiQueue: true
    disks:
    - name: datavolume
      disk:
        bus: virtio
# ...
```

5. Click **Save**.



IMPORTANT

The **spec.template.spec.domain** setting cannot be changed while the VM is running. You must stop the VM before applying the changes, and then restart the VM for the new settings to take effect.

Additional resources for config maps, secrets, and service accounts

- Understanding config maps
- Providing sensitive data to pods

- Understanding and creating service accounts

10.6. EDITING BOOT ORDER

You can update the values for a boot order list by using the web console or the CLI.

With **Boot Order** in the **Virtual Machine Overview** page, you can:

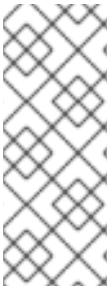
- Select a disk or network interface controller (NIC) and add it to the boot order list.
- Edit the order of the disks or NICs in the boot order list.
- Remove a disk or NIC from the boot order list, and return it back to the inventory of bootable sources.

10.6.1. Adding items to a boot order list in the web console

Add items to a boot order list by using the web console.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Details** tab.
4. Click the pencil icon that is located on the right side of **Boot Order**. If a YAML configuration does not exist, or if this is the first time that you are creating a boot order list, the following message displays: **No resource selected. VM will attempt to boot from disks by order of appearance in YAML file.**
5. Click **Add Source** and select a bootable disk or network interface controller (NIC) for the virtual machine.
6. Add any additional disks or NICs to the boot order list.
7. Click **Save**.



NOTE

If the virtual machine is running, changes to **Boot Order** will not take effect until you restart the virtual machine.

You can view pending changes by clicking **View Pending Changes** on the right side of the **Boot Order** field. The **Pending Changes** banner at the top of the page displays a list of all changes that will be applied when the virtual machine restarts.

10.6.2. Editing a boot order list in the web console

Edit the boot order list in the web console.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.

2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Details** tab.
4. Click the pencil icon that is located on the right side of **Boot Order**.
5. Choose the appropriate method to move the item in the boot order list:
   - If you do not use a screen reader, hover over the arrow icon next to the item that you want to move, drag the item up or down, and drop it in a location of your choice.
   - If you use a screen reader, press the Up Arrow key or Down Arrow key to move the item in the boot order list. Then, press the **Tab** key to drop the item in a location of your choice.
6. Click **Save**.

NOTE

If the virtual machine is running, changes to the boot order list will not take effect until you restart the virtual machine.

You can view pending changes by clicking **View Pending Changes** on the right side of the **Boot Order** field. The **Pending Changes** banner at the top of the page displays a list of all changes that will be applied when the virtual machine restarts.

10.6.3. Editing a boot order list in the YAML configuration file

Edit the boot order list in a YAML configuration file by using the CLI.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Open the YAML configuration file for the virtual machine by running the following command:

```
$ oc edit vm <vm_name> -n <namespace>
```

2. Edit the YAML file and modify the values for the boot order associated with a disk or network interface controller (NIC). For example:

```
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. The boot order value specified for the disk.
2. The boot order value specified for the network interface controller.

3. Save the YAML file.

10.6.4. Removing items from a boot order list in the web console

Remove items from a boot order list by using the web console.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Details** tab.
4. Click the pencil icon that is located on the right side of **Boot Order**.
5. Click the **Remove** icon ⛔ next to the item. The item is removed from the boot order list and saved in the list of available boot sources. If you remove all items from the boot order list, the following message displays: **No resource selected. VM will attempt to boot from disks by order of appearance in YAML file.**

NOTE

If the virtual machine is running, changes to **Boot Order** will not take effect until you restart the virtual machine.

You can view pending changes by clicking **View Pending Changes** on the right side of the **Boot Order** field. The **Pending Changes** banner at the top of the page displays a list of all changes that will be applied when the virtual machine restarts.

10.7. DELETING VIRTUAL MACHINES

You can delete a virtual machine by using the web console or the **oc** command line interface.

10.7.1. Deleting a virtual machine using the web console

Deleting a virtual machine (VM) permanently removes it from the cluster.

If the VM is delete protected, the **Delete** action is disabled in the VM's **Actions** menu.

Prerequisites

- You have disabled the VM's delete protection setting.
- You have stopped the VM.

Procedure

1. From the OpenShift Container Platform web console, choose your view:
   - For a virtualization-focused view, select **Administrator → Virtualization → VirtualMachines**.
   - For a general view, navigate to **Virtualization → VirtualMachines**.
2. Click the **Options** menu  beside a VM and select **Delete**.
   Alternatively, click the VM's name to open the **VirtualMachine details** page and click **Actions → Delete**.

   You can also right-click the VM in the tree view and select **Delete** from the pop-up menu.
3. Optional: Select **With grace period** or clear **Delete disks**.
4. Click **Delete** to permanently delete the VM.

10.7.2. Deleting a virtual machine by using the CLI

You can delete a virtual machine (VM) by using the **oc** command-line interface (CLI). The **oc** client enables you to perform actions on multiple VMs.

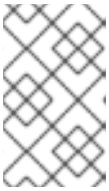
Prerequisites

- You have disabled the VM's delete protection setting.
- You have stopped the VM.
- You have installed the OpenShift CLI (**oc**).

Procedure

- Delete the VM by running the following command:

```
$ oc delete vm <vm_name>
```



NOTE

This command only deletes a VM in the current project. Specify the **-n <project_name>** option if the VM you want to delete is in a different project or namespace.

10.8. ENABLING OR DISABLING VIRTUAL MACHINE DELETE PROTECTION

You can prevent the inadvertent deletion of a virtual machine (VM) by enabling delete protection for the VM. You can also disable delete protection for the VM.

You enable or disable delete protection from either the command line or the VM's **VirtualMachine details** page in the OpenShift Container Platform web console. The option is disabled by default.

You can also choose to remove availability of the delete protection option for any VMs in a cluster you administer. In this case, VMs with the feature already enabled retain the protection, while the option is unavailable for any newly created VMs.

10.8.1. Enabling or disabling virtual machine delete protection by using the web console

To prevent the inadvertent deletion of a virtual machine (VM), you can enable VM delete protection by using the OpenShift Container Platform web console. You can also disable delete protection for a VM.

By default, delete protection is not enabled for VMs. You must set the option for each individual VM.

Procedure

1. From the OpenShift Container Platform web console, choose your view:
   - For a virtualization-focused view, select **Administrator → Virtualization → VirtualMachines**.
   - For a general view, navigate to **Virtualization → VirtualMachines**.
2. From the **VirtualMachines** list, select the VM whose delete protection you want to enable or disable.
3. Click the **Configuration** tab.
4. In the **VirtualMachines details**, choose to enable or disable the protection as follows:
   - To enable the protection:
     a. Set the **Deletion protection** switch to **On**.
     b. Click **Enable** to confirm the protection.
   - To disable the protection:
     a. Set the **Deletion protection** switch to **Off**.
     b. Click **Disable** to disable the protection.

10.8.2. Enabling or disabling VM delete protection by using the CLI

To prevent the inadvertent deletion of a virtual machine (VM), you can enable VM delete protection by using the command line. You can also disable delete protection for a VM.

By default, delete protection is not enabled for VMs. You must set the option for each individual VM.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- Enable delete protection for a VM by running the following command:

```
$ oc patch vm <vm_name> --type merge -p '{"metadata":{"labels":{"kubevirt.io/vm-delete-protection":"True"}}}' -n <namespace>
```

- Disable delete protection for a VM by running the following command:

```
$ oc patch vm <vm_name> --type json -p '[{"op": "remove", "path": "/metadata/labels/kubevirt.io~1vm-delete-protection"}]' -n <namespace>
```

10.8.3. Removing the VM delete protection option

When you enable delete protection on a virtual machine (VM), you ensure that the VM cannot be inadvertently deleted. You can also disable the protection for a VM.

As a cluster administrator, you can choose not to make the VM delete protection option available. VMs with delete protection already enabled retain that setting; for any new VMs that are created, enabling the option is not allowed.

You can remove the delete protection option by establishing a validation admission policy for the cluster and then creating the necessary binding to use the policy in the cluster.

Prerequisites

- You must have cluster administrator privileges.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create the validation admission policy, as shown in the following example:

Example validation admission policy file

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicy
metadata:
  name: "disable-vm-delete-protection"
spec:
  failurePolicy: Fail
  matchConstraints:
    resourceRules:
    - apiGroups:   ["kubevirt.io"]
      apiVersions: ["*"]
      operations:  ["UPDATE", "CREATE"]
      resources:   ["virtualmachines"]
  variables:
    - expression: string('kubevirt.io/vm-delete-protection')
      name: vmDeleteProtectionLabel
  validations:
  - expression: >-
      !has(object.metadata.labels) ||
      !object.metadata.labels.exists(label, label == variables.vmDeleteProtectionLabel) ||
      has(oldObject.metadata.labels) &&
      oldObject.metadata.labels.exists(label, label == variables.vmDeleteProtectionLabel)
    message: "Virtual Machine delete protection feature is disabled"
```

2. Apply the validation admission policy to the cluster:

```
$ oc apply -f disable-vm-delete-protection.yaml
```

3. Create the validation admission policy binding, as shown in the following example:

Example validation admission policy binding file

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingAdmissionPolicyBinding
metadata:
  name: "disable-vm-delete-protection-binding"
spec:
  policyName: "disable-vm-delete-protection"
  validationActions: [Deny]
  matchResources:
```

4. Apply the validation admission policy binding to the cluster:

```
$ oc apply -f disable-vm-delete-protection-binding.yaml
```

10.8.4. Additional resources

- Enabling or disabling virtual machine delete protection by using the web console
- Enabling or disabling virtual machine delete protection by using the CLI

10.9. EXPORTING VIRTUAL MACHINES

You can export a virtual machine (VM) and its associated disks in order to import a VM into another cluster or to analyze the volume for forensic purposes.

You create a **VirtualMachineExport** custom resource (CR) by using the command-line interface.

Alternatively, you can use the **virtctl vmexport** command to create a **VirtualMachineExport** CR and to download exported volumes.

NOTE

You can migrate virtual machines between OpenShift Virtualization clusters by using the Migration Toolkit for Virtualization.

10.9.1. Creating a VirtualMachineExport custom resource

You can create a **VirtualMachineExport** custom resource (CR) to export the following objects:

- Virtual machine (VM): Exports the persistent volume claims (PVCs) of a specified VM.
- VM snapshot: Exports PVCs contained in a **VirtualMachineSnapshot** CR.
- PVC: Exports a PVC. If the PVC is used by another pod, such as the **virt-launcher** pod, the export remains in a **Pending** state until the PVC is no longer in use.

The **VirtualMachineExport** CR creates internal and external links for the exported volumes. Internal links are valid within the cluster. External links can be accessed by using an **Ingress** or **Route**.

The export server supports the following file formats:

- **raw**: Raw disk image file.
- **gzip**: Compressed disk image file.
- **dir**: PVC directory and files.
- **tar.gz**: Compressed PVC file.

Prerequisites

- The VM must be shut down for a VM export.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **VirtualMachineExport** manifest to export a volume from a **VirtualMachine**, **VirtualMachineSnapshot**, or **PersistentVolumeClaim** CR according to the following example and save it as **example-export.yaml**:

   VirtualMachineExport example

```
apiVersion: export.kubevirt.io/v1beta1
kind: VirtualMachineExport
metadata:
  name: example-export
spec:
  source:
    apiGroup: "kubevirt.io" 1
    kind: VirtualMachine 2
    name: example-vm
  ttlDuration: 1h 3
```

   1. Specify the appropriate API group:
      - **"kubevirt.io"** for **VirtualMachine**.
      - **"snapshot.kubevirt.io"** for **VirtualMachineSnapshot**.
      - **""** for **PersistentVolumeClaim**.
   2. Specify **VirtualMachine**, **VirtualMachineSnapshot**, or **PersistentVolumeClaim**.
   3. Optional. The default duration is 2 hours.

2. Create the **VirtualMachineExport** CR:

```
$ oc create -f example-export.yaml
```

3. Get the **VirtualMachineExport** CR:

```
$ oc get vmexport example-export -o yaml
```

The internal and external links for the exported volumes are displayed in the **status** stanza:

Output example

```
apiVersion: export.kubevirt.io/v1beta1
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/v1beta1/namespaces/example/virtualmachineexports/examp
le-export/volumes/example-disk/disk.img
        - format: gzip
          url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1beta1/namespaces/example/virtualmachineexports/examp
le-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. External links are accessible from outside the cluster by using an **Ingress** or **Route**.
2. Internal links are only valid inside the cluster.

10.9.2. Accessing exported virtual machine manifests

After you export a virtual machine (VM) or snapshot, you can get the **VirtualMachine** manifest and related information from the export server.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You exported a virtual machine or VM snapshot by creating a **VirtualMachineExport** custom resource (CR).

NOTE

VirtualMachineExport objects that have the **spec.source.kind: PersistentVolumeClaim** parameter do not generate virtual machine manifests.

Procedure

1. To access the manifests, you must first copy the certificates from the source cluster to the target cluster.
   a. Log in to the source cluster.
   b. Save the certificates to the **cacert.crt** file by running the following command:

```
$ oc get vmexport <export_name> -o jsonpath={.status.links.external.cert} > cacert.crt
```

1. Replace **<export_name>** with the **metadata.name** value from the **VirtualMachineExport** object.

   c. Copy the **cacert.crt** file to the target cluster.
2. Decode the token in the source cluster and save it to the **token_decode** file by running the following command:

```
$ oc get secret export-token-<export_name> -o jsonpath={.data.token} | base64 --decode > token_decode
```

1. Replace **<export_name>** with the **metadata.name** value from the **VirtualMachineExport** object.

3. Copy the **token_decode** file to the target cluster.

4. Get the **VirtualMachineExport** custom resource by running the following command:

```
$ oc get vmexport <export_name> -o yaml
```

5. Review the **status.links** stanza, which is divided into **external** and **internal** sections. Note the **manifests.url** fields within each section:

Example output

```
apiVersion: export.kubevirt.io/v1beta1
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/v1beta1/namespaces/example/virtualmachineexports/examp
le-export/external/manifests/all (1)
      - type: auth-header-secret
        url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1beta1/namespaces/example/virtualmachineexports/examp
le-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. Contains the **VirtualMachine** manifest, **DataVolume** manifest, if present, and a **ConfigMap** manifest that contains the public certificate for the external URL's ingress or route.

2. Contains a secret containing a header that is compatible with Containerized Data Importer (CDI). The header contains a text version of the export token.

3 Contains the **VirtualMachine** manifest, **DataVolume** manifest, if present, and a **ConfigMap** manifest that contains the certificate for the internal URL's export server.

6. Log in to the target cluster.
7. Get the **Secret** manifest by running the following command:

```
$ curl --cacert cacert.crt <secret_manifest_url> -H \ 1
"x-kubevirt-export-token:token_decode" -H \ 2
"Accept:application/yaml"
```

1 Replace **<secret_manifest_url>** with an **auth-header-secret** URL from the **VirtualMachineExport** YAML output.

2 Reference the **token_decode** file that you created earlier.

For example:

```
$ curl --cacert cacert.crt https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1beta1/namespaces/example/virtualmachineexports/examp
le-export/external/manifests/secret -H "x-kubevirt-export-token:token_decode" -H
"Accept:application/yaml"
```

8. Get the manifests of **type: all**, such as the **ConfigMap** and **VirtualMachine** manifests, by running the following command:

```
$ curl --cacert cacert.crt <all_manifest_url> -H \ 1
"x-kubevirt-export-token:token_decode" -H \ 2
"Accept:application/yaml"
```

1 Replace **<all_manifest_url>** with a URL from the **VirtualMachineExport** YAML output.

2 Reference the **token_decode** file that you created earlier.

For example:

```
$ curl --cacert cacert.crt https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1beta1/namespaces/example/virtualmachineexports/examp
le-export/external/manifests/all -H "x-kubevirt-export-token:token_decode" -H
"Accept:application/yaml"
```

Next steps

- You can now create the **ConfigMap** and **VirtualMachine** objects on the target cluster by using the exported manifests.

10.10. MANAGING VIRTUAL MACHINE INSTANCES

If you have standalone virtual machine instances (VMIs) that were created independently outside of the OpenShift Virtualization environment, you can manage them by using the web console or by using **oc** or **virtctl** commands from the command-line interface (CLI).

The **virtctl** command provides more virtualization options than the **oc** command. For example, you can use **virtctl** to pause a VM or expose a port.

10.10.1. About virtual machine instances

A virtual machine instance (VMI) is a representation of a running virtual machine (VM). When a VMI is owned by a VM or by another object, you manage it through its owner in the web console or by using the **oc** command-line interface (CLI).

A standalone VMI is created and started independently with a script, through automation, or by using other methods in the CLI. In your environment, you might have standalone VMIs that were developed and started outside of the OpenShift Virtualization environment. You can continue to manage those standalone VMIs by using the CLI. You can also use the web console for specific tasks associated with standalone VMIs:

- List standalone VMIs and their details.
- Edit labels and annotations for a standalone VMI.
- Delete a standalone VMI.

When you delete a VM, the associated VMI is automatically deleted. You delete a standalone VMI directly because it is not owned by VMs or other objects.

NOTE

Before you uninstall OpenShift Virtualization, list and view the standalone VMIs by using the CLI or the web console. Then, delete any outstanding VMIs.

When you edit a VM, some settings might be applied to the VMIs dynamically and without the need for a restart. Any change made to a VM object that cannot be applied to the VMIs dynamically will trigger the **RestartRequired** VM condition. Changes are effective on the next reboot, and the condition is removed.

10.10.2. Listing all virtual machine instances using the CLI

You can list all virtual machine instances (VMIs) in your cluster, including standalone VMIs and those owned by virtual machines, by using the **oc** command-line interface (CLI).

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- List all VMIs by running the following command:

```
$ oc get vmis -A
```

10.10.3. Listing standalone virtual machine instances using the web console

Using the web console, you can list and view standalone virtual machine instances (VMIs) in your cluster that are not owned by virtual machines (VMs).

NOTE

VMIs that are owned by VMs or other objects are not displayed in the web console. The web console displays only standalone VMIs. If you want to list all VMIs in your cluster, you must use the CLI.

Procedure

- Click **Virtualization → VirtualMachines** from the side menu.
  You can identify a standalone VMI by a dark colored badge next to its name.

10.10.4. Searching for standalone virtual machine instances by using the web console

You can search for virtual machine instances (VMIs) by using the search bar on the **VirtualMachines** page. Use the advanced search to apply additional filters.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization → VirtualMachines** from the side menu.
2. In the search bar at the top of the page, type a VM name, label, or IP address.
3. In the suggestions list, choose one of the following options:
   - Click a VM name to open its details page.
   - Click **All search results found for ...**to view results on a dedicated page.
   - Click a related suggestion to prefill search filters.
4. Optional: To open advanced search options, click the sliders icon next to the search bar. Expand the **Details** section and specify one or more of the available filters: **Name**, **Project**, **Description**, **Labels**, **Date created**, **vCPU**, and **Memory**.
5. Optional: Expand the **Network** section and enter an IP address to filter by.
6. Click **Search**.
7. Optional: If Advanced Cluster Management (ACM) is installed, use the **Cluster** dropdown to search across multiple clusters.
8. Optional: Click the **Save search** icon to store your search in the **kubevirt-user-settings** ConfigMap.

10.10.5. Editing a standalone virtual machine instance using the web console

You can edit the annotations and labels of a standalone virtual machine instance (VMI) using the web console. Other fields are not editable.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization → VirtualMachines** from the side menu.
2. Select a standalone VMI to open the **VirtualMachineInstance details** page.
3. On the **Details** tab, click the pencil icon beside **Annotations** or **Labels**.
4. Make the relevant changes and click **Save**.

10.10.6. Deleting a standalone virtual machine instance using the CLI

You can delete a standalone virtual machine instance (VMI) by using the **oc** command-line interface (CLI).

Prerequisites

- Identify the name of the VMI that you want to delete.
- You have installed the OpenShift CLI (**oc**).

Procedure

- Delete the VMI by running the following command:

```
$ oc delete vmi <vmi_name>
```

10.10.7. Deleting a standalone virtual machine instance using the web console

Delete a standalone virtual machine instance (VMI) from the web console.

Procedure

1. In the OpenShift Container Platform web console, click **Virtualization → VirtualMachines** from the side menu.
2. Click **Actions → Delete VirtualMachineInstance**.
3. In the confirmation pop-up window, click **Delete** to permanently delete the standalone VMI.

10.11. CONTROLLING VIRTUAL MACHINE STATES

You can use **virtctl** to manage virtual machine states and perform other actions from the CLI. For example, you can use **virtctl** to force stop a VM or expose a port.

You can stop, start, restart, pause, and unpause virtual machines from the web console.

10.11.1. Enabling confirmations of virtual machine actions

The **Stop**, **Restart**, and **Pause** actions can display confirmation dialogs if confirmation is enabled. By default, confirmation is disabled.

Procedure

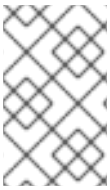
1. In the **Virtualization** section of the OpenShift Container Platform web console, navigate to **Overview → Settings → Cluster → General settings**.
2. Toggle the **VirtualMachine actions confirmation** setting to On.

10.11.2. Starting a virtual machine

You can start a virtual machine (VM) from the web console.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. In the tree view, select the project that contains the VM that you want to start.
3. Navigate to the appropriate menu for your use case:
   - To stay on this page, where you can perform actions on multiple VMs:
     a. Click the Options menu  located at the far right end of the row and click **Start VirtualMachine**.
   - To start the VM from the tree view:
     a. Click the > icon next to the project name to open the list of VMs.
     b. Right-click the name of the VM and select **Start**.
   - To view comprehensive information about the selected VM before you start it:
     a. Access the **VirtualMachine details** page by clicking the name of the VM.
     b. Click **Actions → Start**.



NOTE

When you start VM that is provisioned from a **URL** source for the first time, the VM has a status of **Importing** while OpenShift Virtualization imports the container from the URL endpoint. Depending on the size of the image, this process might take several minutes.

10.11.3. Stopping a virtual machine

You can stop a virtual machine (VM) from the web console.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. In the tree view, select the project that contains the VM that you want to stop.
3. Navigate to the appropriate menu for your use case:
   - To stay on this page, where you can perform actions on multiple VMs:

a. Click the Options menu  located at the far right end of the row and click **Stop VirtualMachine**.

b. If action confirmation is enabled, click **Stop** in the confirmation dialog.

- To stop the VM from the tree view:

a. Click the > icon next to the project name to open the list of VMs.

b. Right-click the name of the VM and select **Stop**.

c. If action confirmation is enabled, click **Stop** in the confirmation dialog.

- To view comprehensive information about the selected VM before you stop it:

a. Access the **VirtualMachine details** page by clicking the name of the VM.

b. Click **Actions → Stop**.

c. If action confirmation is enabled, click **Stop** in the confirmation dialog.

10.11.4. Restarting a virtual machine

You can restart a running virtual machine (VM) from the web console.

IMPORTANT

To avoid errors, do not restart a VM while it has a status of **Importing**.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. In the tree view, select the project that contains the VM that you want to restart.
3. Navigate to the appropriate menu for your use case:

- To stay on this page, where you can perform actions on multiple VMs:

a. Click the Options menu  located at the far right end of the row and click **Restart**.

b. If action confirmation is enabled, click **Restart** in the confirmation dialog.

- To restart the VM from the tree view:

a. Click the > icon next to the project name to open the list of VMs.

b. Right-click the name of the VM and select **Restart**.

c. If action confirmation is enabled, click **Restart** in the confirmation dialog.

- To view comprehensive information about the selected VM before you restart it:

a. Access the **VirtualMachine details** page by clicking the name of the virtual machine.

b. Click **Actions → Restart**.

c. If action confirmation is enabled, click **Restart** in the confirmation dialog.

10.11.5. Pausing a virtual machine

You can pause a virtual machine (VM) from the web console.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. In the tree view, select the project that contains the VM that you want to pause.
3. Navigate to the appropriate menu for your use case:
   - To stay on this page, where you can perform actions on multiple VMs:
     a. Click the Options menu  located at the far right end of the row and click **Pause VirtualMachine**.
     b. If action confirmation is enabled, click **Pause** in the confirmation dialog.
   - To pause the VM from the tree view:
     a. Click the > icon next to the project name to open the list of VMs.
     b. Right-click the name of the VM and select **Pause**.
     c. If action confirmation is enabled, click **Pause** in the confirmation dialog.
   - To view comprehensive information about the selected VM before you pause it:
     a. Access the **VirtualMachine details** page by clicking the name of the VM.
     b. Click **Actions → Pause**.
     c. If action confirmation is enabled, click **Pause** in the confirmation dialog.

10.11.6. Unpausing a virtual machine

You can unpause a paused virtual machine (VM) from the web console.

Prerequisites

- At least one of your VMs must have a status of **Paused**.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. In the tree view, select the project that contains the VM that you want to unpause.
3. Navigate to the appropriate menu for your use case:

- To stay on this page, where you can perform actions on multiple VMs:
    a. Click the Options menu  located at the far right end of the row and click **Unpause VirtualMachine**.
- To unpause the VM from the tree view:
    a. Click the > icon next to the project name to open the list of VMs.
    b. Right-click the name of the VM and select **Unpause**.
- To view comprehensive information about the selected VM before you unpause it:
    a. Access the **VirtualMachine details** page by clicking the name of the virtual machine.
    b. Click **Actions → Unpause**.

10.11.7. Controlling the state of multiple virtual machines

You can start, stop, restart, pause, and unpause multiple virtual machines (VMs) from the web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Optional: Enable the **Show only projects with VirtualMachines** option above the tree view to limit the displayed projects.
3. Select a relevant project from the tree view.
4. Navigate to the appropriate menu for your use case:
    - To change the state of all VMs in the selected project:
        a. Right-click the name of the project in the tree view and select the intended action from the menu.
        b. If action confirmation is enabled, confirm the action in the confirmation dialog.
    - To change the state of specific VMs:
        a. Select a checkbox next to the VMs you want to work with. To select all VMs, click the checkbox in the **VirtualMachines** table header.
        b. Click **Actions** and select the intended action from the menu.
        c. If action confirmation is enabled, confirm the action in the confirmation dialog.

10.12. USING VIRTUAL TRUSTED PLATFORM MODULE DEVICES

Add a virtual Trusted Platform Module (vTPM) device to a new or existing virtual machine by editing the **VirtualMachine** (VM) or **VirtualMachineInstance** (VMI) manifest.

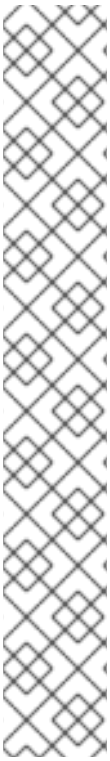
IMPORTANT

With OpenShift Virtualization 4.18 and newer, you can export virtual machines (VMs) with attached vTPM devices, create snapshots of these VMs, and restore VMs from these snapshots. However, cloning a VM with a vTPM device attached to it or creating a new VM from its snapshot is not supported.

10.12.1. About vTPM devices

A virtual Trusted Platform Module (vTPM) device functions like a physical Trusted Platform Module (TPM) hardware chip. You can use a vTPM device with any operating system, but Windows 11 requires the presence of a TPM chip to install or boot. A vTPM device allows VMs created from a Windows 11 image to function without a physical TPM chip.

OpenShift Virtualization supports persisting vTPM device state by using Persistent Volume Claims (PVCs) for VMs. If you do not specify the storage class for this PVC, OpenShift Virtualization uses the default storage class for virtualization workloads. If the default storage class for virtualization workloads is not set, OpenShift Virtualization uses the default storage class for the cluster.



NOTE

The storage class that is marked as default for virtualization workloads has the annotation **storageclass.kubevirt.io/is-default-virt-class** set to "true". You can find this storage class by running the following command:

```
$ oc get sc -o jsonpath='{range .items[?
(.metadata.annotations.storageclass\.kubevirt\.io/is-default-virt-class=="true")]}
{.metadata.name}{"\n"}{end}'
```

Similarly, the default storage class for the cluster has the annotation **storageclass.kubernetes.io/is-default-class** set to "true". To find this storage class, run the following command:

```
$ oc get sc -o jsonpath='{range .items[?
(.metadata.annotations.storageclass\.kubernetes\.io/is-default-class=="true")]}
{.metadata.name}{"\n"}{end}'
```

To ensure consistent behavior, configure only one storage class as the default for virtualization workloads and for the cluster respectively.

It is recommended that you specify the storage class explicitly by setting the **vmStateStorageClass** attribute in the **HyperConverged** custom resource (CR):

```
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  vmStateStorageClass: <storage_class_name>

# ...
```

If you do not enable vTPM, then the VM does not recognize a TPM device, even if the node has one.

10.12.2. Adding a vTPM device to a virtual machine

Adding a virtual Trusted Platform Module (vTPM) device to a virtual machine (VM) allows you to run a VM created from a Windows 11 image without a physical TPM device. A vTPM device also stores secrets for that VM.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Run the following command to update the VM configuration:

```
$ oc edit vm <vm_name> -n <namespace>
```

2. Edit the VM specification to add the vTPM device. For example:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          tpm: 1
            persistent: true 2
# ...
```

1. Adds the vTPM device to the VM.
2. Specifies that the vTPM device state persists after the VM is shut down. The default value is **false**.

3. To apply your changes, save and exit the editor.
4. Optional: If you edited a running virtual machine, you must restart it for the changes to take effect.

10.13. MANAGING VIRTUAL MACHINES WITH OPENSHIFT PIPELINES

Red Hat OpenShift Pipelines is a Kubernetes-native CI/CD framework that allows developers to design and run each step of the CI/CD pipeline in its own container.

By using OpenShift Pipelines tasks and the example pipeline, you can do the following:

- Create and manage virtual machines (VMs), persistent volume claims (PVCs), data volumes, and data sources.
- Run commands in VMs.
- Manipulate disk images with **libguestfs** tools.

The tasks are located in the task catalog (ArtifactHub).

The example Windows pipeline is located in the pipeline catalog (ArtifactHub).

10.13.1. Prerequisites

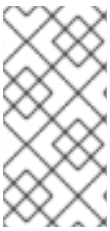
- You have access to an OpenShift Container Platform cluster with **cluster-admin** permissions.
- You have installed the OpenShift CLI (**oc**).
- You have installed OpenShift Pipelines.

10.13.2. Supported virtual machine tasks

The following table shows the supported tasks.

Table 10.2. Supported virtual machine tasks

Task	Description
create-vm-from-manifest	Create a virtual machine from a provided manifest or with **virtctl**.
create-vm-from-template	Create a virtual machine from a template.
copy-template	Copy a virtual machine template.
modify-vm-template	Modify a virtual machine template.
modify-data-object	Create or delete data volumes or data sources.
cleanup-vm	Run a script or a command in a virtual machine and stop or delete the virtual machine afterward.
disk-virt-customize	Use the **virt-customize** tool to run a customization script on a target PVC.
disk-virt-sysprep	Use the **virt-sysprep** tool to run a sysprep script on a target PVC.
wait-for-vmi-status	Wait for a specific status of a virtual machine instance and fail or succeed based on the status.



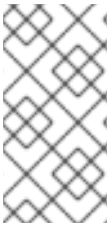
NOTE

Virtual machine creation in pipelines now utilizes **ClusterInstanceType** and **ClusterPreference** instead of template-based tasks, which have been deprecated. The **create-vm-from-template**, **copy-template**, and **modify-vm-template** commands remain available but are not used in default pipeline tasks.

10.13.3. Windows EFI installer pipeline

You can run the Windows EFI installer pipeline by using the web console or CLI.

The Windows EFI installer pipeline installs Windows 10, Windows 11, or Windows Server 2022 into a new data volume from a Windows installation image (ISO file). A custom answer file is used to run the installation process.



NOTE

The Windows EFI installer pipeline uses a config map file with **sysprep** predefined by OpenShift Container Platform and suitable for Microsoft ISO files. For ISO files pertaining to different Windows editions, it may be necessary to create a new config map file with a system-specific **sysprep** definition.

10.13.3.1. Running the example pipelines using the web console

You can run the example pipelines from the **Pipelines** menu in the web console.

Procedure

1. Click **Pipelines** → **Pipelines** in the side menu.
2. Select a pipeline to open the **Pipeline details** page.
3. From the **Actions** list, select **Start**. The **Start Pipeline** dialog is displayed.
4. Keep the default values for the parameters and then click **Start** to run the pipeline. The **Details** tab tracks the progress of each task and displays the pipeline status.

10.13.3.2. Running the example pipelines using the CLI

Use a **PipelineRun** resource to run the example pipelines. A **PipelineRun** object is the running instance of a pipeline. It instantiates a pipeline for execution with specific inputs, outputs, and execution parameters on a cluster. It also creates a **TaskRun** object for each task in the pipeline.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. To run the Microsoft Windows 11 installer pipeline, create the following **PipelineRun** manifest:

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  generateName: windows11-installer-run-
  labels:
    pipelinerun: windows11-installer-run
spec:
    params:
    -   name: winImageDownloadURL
        value: <windows_image_download_url> 1
```

```
    - name: acceptEula
      value: false 2
  pipelineRef:
    params:
    - name: catalog
      value: redhat-pipelines
    - name: type
      value: artifact
    - name: kind
      value: pipeline
    - name: name
      value: windows-efi-installer
    - name: version
      value: 4.20
    resolver: hub
  taskRunSpecs:
  - pipelineTaskName: modify-windows-iso-file
    PodTemplate:
      securityContext:
        fsGroup: 107
        runAsUser: 107
```

1 Specify the URL for the Windows 11 64-bit ISO file. The product's language must be English (United States).

2 Example **PipelineRun** objects have a special parameter, **acceptEula**. By setting this parameter, you are agreeing to the applicable Microsoft user license agreements for each deployment or installation of the Microsoft products. If you set it to false, the pipeline exits at the first task.

2. Apply the **PipelineRun** manifest:

```
$ oc apply -f windows11-customize-run.yaml
```

10.13.4. Removing deprecated or unused resources

You can clean up deprecated or unused resources associated with the Red Hat OpenShift Pipelines Operator.

Procedure

- Remove any remaining OpenShift Pipelines resources from the cluster by running the following command:

```
$ oc delete clusterroles,rolebindings,serviceaccounts,configmaps,pipelines,tasks \
  --selector 'app.kubernetes.io/managed-by=ssp-operator' \
  --selector 'app.kubernetes.io/component in (tektonPipelines,tektonTasks)' \
  --selector 'app.kubernetes.io/name in (tekton-pipelines,tekton-tasks)' \
  --ignore-not-found \
  --all-namespaces
```

If the Red Hat OpenShift Pipelines Operator custom resource definitions (CRDs) have already been removed, the command may return an error. You can safely ignore this, as all other matching resources will still be deleted.

10.13.5. Additional resources

- Creating CI/CD solutions for applications using Red Hat OpenShift Pipelines
- Creating a Windows VM

10.14. MIGRATING VMS IN A SINGLE CLUSTER TO A DIFFERENT STORAGE CLASS

You can migrate virtual machines (VMs) within a single cluster from one storage class to a different storage class. By using the OpenShift Container Platform web console, you can perform the migration for the VMs in bulk.

10.14.1. Migrating VMs in a single cluster to a different storage class by using the web console

By using the OpenShift Container Platform web console, you can migrate single-cluster VMs in bulk from one storage class to another storage class.

Prerequisites

- The VMs you select for each bulk migration must be in the same namespace.
- The Migration Toolkit for Containers (MTC) must be installed.

Procedure

1. From the OpenShift Container Platform web console, navigate to **Virtualization → VirtualMachines**.
2. From the list of VMs in the same namespace, select each VM that you want to move from its current storage class.
3. Select **Actions → Migrate storage**.

   Alternatively, you can access this option by opening the Options menu  for a selected VM, and then selecting **Migration → Storage**.

   The **Migrate VirtualMachine storage** page opens.
4. To review the VMs that you want to migrate, click the link that identifies the number of VMs and volumes. Click **View more** to see the full list.
5. Select either the entire VM or only selected volumes for storage class migration. If you choose to migrate only selected volumes, the page expands to allow you to make specific selections. You can also click **VirtualMachine name** to select all VMs.
6. Click **Next**.
7. From the list of available storage classes, select the destination storage class for the migration.
8. Click **Next**.
9. Review the details, and click **Migrate VirtualMachine storage** to start the migration.

10. Optional: Click **Stop** to interrupt the migration, or click **View storage migrations** to see the status of current and previous migrations.

10.15. ADVANCED VIRTUAL MACHINE MANAGEMENT

10.15.1. Working with resource quotas for virtual machines

Create and manage resource quotas for virtual machines.

10.15.1.1. Setting resource quota limits for virtual machines

By default, OpenShift Virtualization automatically manages CPU and memory limits for virtual machines (VMs) if a namespace enforces resource quotas that require limits to be set. The memory limit is automatically set to twice the requested memory and the CPU limit is set to one per vCPU.

You can customize the memory limit ratio for a specific namespace by adding the **alpha.kubevirt.io/auto-memory-limits-ratio** label to the namespace. For example, the following command sets the memory limit ratio to 1.2:

```
$ oc label ns/my-virtualization-project  alpha.kubevirt.io/auto-memory-limits-ratio=1.2
```

WARNING

Avoid managing resource quota limits manually. To prevent misconfigurations or scheduling issues, rely on the automatic resource limit management provided by OpenShift Virtualization unless you have a specific need to override the defaults.

Resource quotas that only use requests automatically work with VMs. If your resource quota uses limits, you must manually set resource limits on VMs. Resource limits must be at least 100 MiB larger than resource requests.

Procedure

1. Set limits for a VM by editing the **VirtualMachine** manifest. For example:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: with-limits
spec:
  runStrategy: Halted
  template:
    spec:
      domain:
# ...
        resources:
          requests:
            memory: 128Mi
          limits:
```

```
        memory: 256Mi (1)
```

(1) This configuration is supported because the **limits.memory** value is at least **100Mi** larger than the **requests.memory** value.

2. Save the **VirtualMachine** manifest.

10.15.1.2. Additional resources

- Resource quotas per project
- Resource quotas across multiple projects

10.15.2. Configuring the Application-Aware Quota (AAQ) Operator

You can use the Application-Aware Quota (AAQ) Operator to customize and manage resource quotas for individual components in an OpenShift Container Platform cluster.

10.15.2.1. About the AAQ Operator

The Application-Aware Quota (AAQ) Operator provides more flexible and extensible quota management compared to the native **ResourceQuota** object in the OpenShift Container Platform platform.

In a multi-tenant cluster environment, where multiple workloads operate on shared infrastructure and resources, using the Kubernetes native **ResourceQuota** object to limit aggregate CPU and memory consumption presents infrastructure overhead and live migration challenges for OpenShift Virtualization workloads.

OpenShift Virtualization requires significant compute resource allocation to handle virtual machine (VM) live migrations and manage VM infrastructure overhead. When upgrading OpenShift Virtualization, you must migrate VMs to upgrade the **virt-launcher** pod. However, migrating a VM in the presence of a resource quota can cause the migration, and subsequently the upgrade, to fail.

With AAQ, you can allocate resources for VMs without interfering with cluster-level activities such as upgrades and node maintenance. The AAQ Operator also supports non-compute resources which eliminates the need to manage both the native resource quota and AAQ API objects separately.

10.15.2.1.1. AAQ Operator controller and custom resources

The AAQ Operator introduces two new API objects defined as custom resource definitions (CRDs) for managing alternative quota implementations across multiple namespaces:

- **ApplicationAwareResourceQuota**: Sets aggregate quota restrictions enforced per namespace. The **ApplicationAwareResourceQuota** API is compatible with the native **ResourceQuota** object and shares the same specification and status definitions.

  Example manifest

```
apiVersion: aaq.kubevirt.io/v1alpha1
kind: ApplicationAwareResourceQuota
metadata:
  name: example-resource-quota
```

```
spec:
  hard:
    requests.memory: 1Gi
    limits.memory: 1Gi
    requests.cpu/vmi: "1" (1)
    requests.memory/vmi: 1Gi (2)
# ...
```

(1) The maximum amount of CPU that is allowed for VM workloads in the default namespace.

(2) The maximum amount of RAM that is allowed for VM workloads in the default namespace.

- **ApplicationAwareClusterResourceQuota**: Mirrors the **ApplicationAwareResourceQuota** object at a cluster scope. It is compatible with the native **ClusterResourceQuota** API object and shares the same specification and status definitions. When creating an AAQ cluster quota, you can select multiple namespaces based on annotation selection, label selection, or both by editing the **spec.selector.labels** or **spec.selector.annotations** fields.

  Example manifest

```
apiVersion: aaq.kubevirt.io/v1alpha1
kind: ApplicationAwareClusterResourceQuota (1)
metadata:
  name: example-resource-quota
spec:
  quota:
    hard:
      requests.memory: 1Gi
      limits.memory: 1Gi
      requests.cpu/vmi: "1"
      requests.memory/vmi: 1Gi
  selector:
    annotations: null
    labels:
      matchLabels:
        kubernetes.io/metadata.name: default
# ...
```

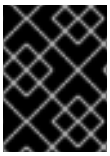
(1) You can only create an **ApplicationAwareClusterResourceQuota** object if the **spec.allowApplicationAwareClusterResourceQuota** field in the **HyperConverged** custom resource (CR) is set to **true**.

NOTE

If both **spec.selector.labels** and **spec.selector.annotations** fields are set, only namespaces that match both are selected.

The AAQ controller uses a scheduling gate mechanism to evaluate whether there is enough of a resource available to run a workload. If so, the scheduling gate is removed from the pod and it is considered ready for scheduling. The quota usage status is updated to indicate how much of the quota is used.

If the CPU and memory requests and limits for the workload exceed the enforced quota usage limit, the pod remains in **SchedulingGated** status until there is enough quota available. The AAQ controller creates an event of type **Warning** with details on why the quota was exceeded. You can view the event details by using the **oc get events** command.



IMPORTANT

Pods that have the **spec.nodeName** field set to a specific node cannot use namespaces that match the **spec.namespaceSelector** labels defined in the **HyperConverged** CR.

10.15.2.2. Enabling the AAQ Operator

To deploy the AAQ Operator, set the **enableApplicationAwareQuota** field value to **true** in the **HyperConverged** custom resource (CR).

Prerequisites

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

Procedure

- Set the **enableApplicationAwareQuota** field value to **true** in the **HyperConverged** CR by running the following command:

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

10.15.2.3. Configuring the AAQ Operator by using the CLI

You can configure the AAQ Operator by specifying the fields of the **spec.applicationAwareConfig** object in the **HyperConverged** custom resource (CR).

Prerequisites

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

Procedure

- Update the **HyperConverged** CR by running the following command:

```
$ oc patch hco kubevirt-hyperconverged -n openshift-cnv --type merge -p '{
  "spec": {
    "applicationAwareConfig": {
      "vmiCalcConfigName": "DedicatedVirtualResources",
      "namespaceSelector": {
        "matchLabels": {
          "app": "my-app"
        }
      },
```

```
      "allowApplicationAwareClusterResourceQuota": true
    }
  }
}'
```

where:

vmiCalcConfigName

Specifies how resource counting is managed for pods that run virtual machine (VM) workloads. Possible values are:

- **VmiPodUsage**: Counts compute resources for pods associated with VMs in the same way as native resource quotas and excludes migration-related resources.
- **VirtualResources**: Counts compute resources based on the VM specifications, using the VM RAM size for memory and virtual CPUs for processing.
- **DedicatedVirtualResources** (default): Similar to **VirtualResources**, but separates resource tracking for pods associated with VMs by adding a **/vmi** suffix to CPU and memory resource names. For example, **requests.cpu/vmi** and **requests.memory/vmi**.

namespaceSelector

Determines the namespaces for which an AAQ scheduling gate is added to pods when they are created. If a namespace selector is not defined, the AAQ Operator targets namespaces with the **application-aware-quota/enable-gating** label as default.

allowApplicationAwareClusterResourceQuota

If set to **true**, you can create and manage the **ApplicationAwareClusterResourceQuota** object. Setting this attribute to **true** can increase scheduling time.

10.15.2.4. Additional resources

- Resource quotas per project
- Resource quotas across multiple projects
- **ResourceQuota** API reference
- **ClusterResourceQuota** API reference
- Pod scheduling gates specification
- Viewing system event information in an OpenShift Container Platform cluster

10.15.3. Specifying nodes for virtual machines

You can place virtual machines (VMs) on specific nodes by using node placement rules.

10.15.3.1. About node placement for virtual machines

To ensure that virtual machines (VMs) run on appropriate nodes, you can configure node placement rules. You might want to do this if:

- You have several VMs. To ensure fault tolerance, you want them to run on different nodes.

- You have two chatty VMs. To avoid redundant inter-node routing, you want the VMs to run on the same node.
- Your VMs require specific hardware features that are not present on all available nodes.
- You have a pod that adds capabilities to a node, and you want to place a VM on that node so that it can use those capabilities.

NOTE

Virtual machine placement relies on any existing node placement rules for workloads. If workloads are excluded from specific nodes on the component level, virtual machines cannot be placed on those nodes.

You can use the following rule types in the **spec** field of a **VirtualMachine** manifest:

nodeSelector

Allows virtual machines to be scheduled on nodes that are labeled with the key-value pair or pairs that you specify in this field. The node must have labels that exactly match all listed pairs.

affinity

Enables you to use more expressive syntax to set rules that match nodes with virtual machines. For example, you can specify that a rule is a preference, rather than a hard requirement, so that virtual machines are still scheduled if the rule is not satisfied. Pod affinity, pod anti-affinity, and node affinity are supported for virtual machine placement. Pod affinity works for virtual machines because the **VirtualMachine** workload type is based on the **Pod** object.

tolerations

Allows virtual machines to be scheduled on nodes that have matching taints. If a taint is applied to a node, that node only accepts virtual machines that tolerate the taint.

NOTE

Affinity rules only apply during scheduling. OpenShift Container Platform does not reschedule running workloads if the constraints are no longer met.

10.15.3.2. Node placement examples

The following example YAML file snippets use **nodePlacement**, **affinity**, and **tolerations** fields to customize node placement for virtual machines.

10.15.3.2.1. Example: VM node placement with nodeSelector

In this example, the virtual machine requires a node that has metadata containing both **example-key-1 = example-value-1** and **example-key-2 = example-value-2** labels.

WARNING

If there are no nodes that fit this description, the virtual machine is not scheduled.

Example VM manifest

```
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
# ...
```

10.15.3.2.2. Example: VM node placement with pod affinity and pod anti-affinity

In this example, the VM must be scheduled on a node that has a running pod with the label **example-key-1 = example-value-1**. If there is no such pod running on any node, the VM is not scheduled.

If possible, the VM is not scheduled on a node that has any pod with the label **example-key-2 = example-value-2**. However, if all candidate nodes have a pod with this label, the scheduler ignores this constraint.

Example VM manifest

```
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. If you use the **requiredDuringSchedulingIgnoredDuringExecution** rule type, the VM is not scheduled if the constraint is not met.
2. If you use the **preferredDuringSchedulingIgnoredDuringExecution** rule type, the VM is still scheduled if the constraint is not met, as long as all required constraints are met.

10.15.3.2.3. Example: VM node placement with node affinity

In this example, the VM must be scheduled on a node that has the label **example.io/example-key = example-value-1** or the label **example.io/example-key = example-value-2**. The constraint is met if only one of the labels is present on the node. If neither label is present, the VM is not scheduled.

If possible, the scheduler avoids nodes that have the label **example-node-label-key = example-node-label-value**. However, if all candidate nodes have this label, the scheduler ignores this constraint.

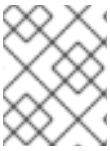
Example VM manifest

```
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. If you use the **requiredDuringSchedulingIgnoredDuringExecution** rule type, the VM is not scheduled if the constraint is not met.
2. If you use the **preferredDuringSchedulingIgnoredDuringExecution** rule type, the VM is still scheduled if the constraint is not met, as long as all required constraints are met.

10.15.3.2.4. Example: VM node placement with tolerations

In this example, nodes that are reserved for virtual machines are already labeled with the **key=virtualization:NoSchedule** taint. Because this virtual machine has matching **tolerations**, it can schedule onto the tainted nodes.



NOTE

A virtual machine that tolerates a taint is not required to schedule onto a node with that taint.

Example VM manifest

```
metadata:
  name: example-vm-tolerations
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  tolerations:
  - key: "key"
    operator: "Equal"
    value: "virtualization"
    effect: "NoSchedule"
# ...
```

10.15.3.3. Additional resources

- Specifying nodes for virtualization components
- Placing pods on specific nodes using node selectors
- Controlling pod placement on nodes using node affinity rules
- Controlling pod placement using node taints

10.15.4. Configuring the default CPU model

Use the **defaultCPUModel** setting in the **HyperConverged** custom resource (CR) to define a cluster-wide default CPU model.

The virtual machine (VM) CPU model depends on the availability of CPU models within the VM and the cluster.

- If the VM does not have a defined CPU model:
  - The **defaultCPUModel** is automatically set using the CPU model defined at the cluster-wide level.

- If both the VM and the cluster have a defined CPU model:
  - The VM's CPU model takes precedence.
- If neither the VM nor the cluster have a defined CPU model:
  - The host-model is automatically set using the CPU model defined at the host level.

10.15.4.1. Configuring the default CPU model

Configure the **defaultCPUModel** by updating the **HyperConverged** custom resource (CR). You can change the **defaultCPUModel** while OpenShift Virtualization is running.

NOTE

The **defaultCPUModel** is case sensitive.

Prerequisites

- Install the OpenShift CLI (oc).

Procedure

1. Open the **HyperConverged** CR by running the following command:

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

2. Add the **defaultCPUModel** field to the CR and set the value to the name of a CPU model that exists in the cluster:

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

3. Apply the YAML file to your cluster.

10.15.5. Using UEFI mode for virtual machines

You can boot a virtual machine (VM) in Unified Extensible Firmware Interface (UEFI) mode.

10.15.5.1. About UEFI mode for virtual machines

Unified Extensible Firmware Interface (UEFI), like legacy BIOS, initializes hardware components and operating system image files when a computer starts. UEFI supports more modern features and customization options than BIOS, enabling faster boot times.

It stores all the information about initialization and startup in a file with a **.efi** extension, which is stored on a special partition called EFI System Partition (ESP). The ESP also contains the boot loader programs for the operating system that is installed on the computer.

10.15.5.2. Booting virtual machines in UEFI mode

You can configure a virtual machine to boot in UEFI mode by editing the **VirtualMachine** manifest.

Prerequisites

- Install the OpenShift CLI (**oc**).

Procedure

1. Edit or create a **VirtualMachine** manifest file. Use the **spec.firmware.bootloader** stanza to configure UEFI mode:

   Booting in UEFI mode with secure boot active

```
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 requires System Management Mode (**SMM**) to be enabled for Secure Boot in UEFI mode to occur.

   (2) OpenShift Virtualization supports a VM with or without Secure Boot when using UEFI mode. If Secure Boot is enabled, then UEFI mode is required. However, UEFI mode can be enabled without using Secure Boot.

2. Apply the manifest to your cluster by running the following command:

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

10.15.5.3. Enabling persistent EFI

You can enable EFI persistence in a VM by configuring an RWX storage class at the cluster level and adjusting the settings in the EFI section of the VM.

Prerequisites

- You must have cluster administrator privileges.
- You must have a storage class that supports RWX access mode and FS volume mode.
- You have installed the OpenShift CLI (**oc**).

Procedure

- Enable the **VMPersistentState** feature gate by running the following command:

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

10.15.5.4. Configuring VMs with persistent EFI

You can configure a VM to have EFI persistence enabled by editing its manifest file.

Prerequisites

- **VMPersistentState** feature gate enabled.

Procedure

- Edit the VM manifest file and save to apply settings.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm
spec:
  template:
    spec:
      domain:
        firmware:
          bootloader:
            efi:
              persistent: true
# ...
```

10.15.6. Configuring PXE booting for virtual machines

PXE booting, or network booting, is available in OpenShift Virtualization. Network booting allows a computer to boot and load an operating system or other program without requiring a locally attached storage device. For example, you can use it to choose your desired OS image from a PXE server when deploying a new host.

10.15.6.1. PXE booting with a specified MAC address

As an administrator, you can boot a client over the network by first creating a **NetworkAttachmentDefinition** object for your PXE network. Then, reference the network attachment definition in your virtual machine instance configuration file before you start the virtual machine instance. You can also specify a MAC address in the virtual machine instance configuration file, if required by the PXE server.

Prerequisites

- A Linux bridge must be connected.
- The PXE server must be connected to the same VLAN as the bridge.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Configure a PXE network on the cluster:
   a. Create the network attachment definition file for PXE network **pxe-net-conf**:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: pxe-net-conf 1
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "pxe-net-conf", 2
      "type": "bridge", 3
      "bridge": "bridge-interface", 4
      "macspoofchk": false, 5
      "vlan": 100, 6
      "disableContainerInterface": true,
      "preserveDefaultVlan": false 7
    }
```

1. The name for the **NetworkAttachmentDefinition** object.
2. The name for the configuration. It is recommended to match the configuration name to the **name** value of the network attachment definition.
3. The actual name of the Container Network Interface (CNI) plugin that provides the network for this network attachment definition. This example uses a Linux bridge CNI plugin. You can also use an OVN-Kubernetes localnet or an SR-IOV CNI plugin.
4. The name of the Linux bridge configured on the node.
5. Optional: A flag to enable the MAC spoof check. When set to **true**, you cannot change the MAC address of the pod or guest interface. This attribute allows only a single MAC address to exit the pod, which provides security against a MAC spoofing attack.
6.

Optional: The VLAN tag. No additional VLAN configuration is required on the node network configuration policy.

7 Optional: Indicates whether the VM connects to the bridge through the default VLAN. The default value is **true**.

2. Create the network attachment definition by using the file you created in the previous step:

```
$ oc create -f pxe-net-conf.yaml
```

3. Edit the virtual machine instance configuration file to include the details of the interface and network.

   a. Specify the network and MAC address, if required by the PXE server. If the MAC address is not specified, a value is assigned automatically.
   Ensure that **bootOrder** is set to **1** so that the interface boots first. In this example, the interface is connected to a network called **<pxe-net>**:

```
interfaces:
- masquerade: {}
  name: default
- bridge: {}
  name: pxe-net
  macAddress: de:00:00:00:00:de
  bootOrder: 1
```

NOTE

Boot order is global for interfaces and disks.

   b. Assign a boot device number to the disk to ensure proper booting after operating system provisioning.
   Set the disk **bootOrder** value to **2**:

```
devices:
  disks:
  - disk:
      bus: virtio
    name: containerdisk
    bootOrder: 2
```

   c. Specify that the network is connected to the previously created network attachment definition. In this scenario, **<pxe-net>** is connected to the network attachment definition called **<pxe-net-conf>**:

```
networks:
- name: default
  pod: {}
- name: pxe-net
  multus:
    networkName: pxe-net-conf
```

4. Create the virtual machine instance:

```
$ oc create -f vmi-pxe-boot.yaml
```

Example output

```
virtualmachineinstance.kubevirt.io "vmi-pxe-boot" created
```

5. Wait for the virtual machine instance to run:

```
$ oc get vmi vmi-pxe-boot -o yaml | grep -i phase
  phase: Running
```

6. View the virtual machine instance using VNC:

```
$ virtctl vnc vmi-pxe-boot
```

7. Watch the boot screen to verify that the PXE boot is successful.

8. Log in to the virtual machine instance:

```
$ virtctl console vmi-pxe-boot
```

Verification

1. Verify the interfaces and MAC address on the virtual machine and that the interface connected to the bridge has the specified MAC address. In this case, we used **eth1** for the PXE boot, without an IP address. The other interface, **eth0**, got an IP address from OpenShift Container Platform.

```
$ ip addr
```

Example output

```
...
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
```

10.15.6.2. OpenShift Virtualization networking glossary

The following terms are used throughout OpenShift Virtualization documentation:

Container Network Interface (CNI)

A Cloud Native Computing Foundation project, focused on container network connectivity. OpenShift Virtualization uses CNI plugins to build upon the basic Kubernetes networking functionality.

Multus

A "meta" CNI plugin that allows multiple CNIs to exist so that a pod or virtual machine can use the interfaces it needs.

Custom resource definition (CRD)

A Kubernetes API resource that allows you to define custom resources, or an object defined by using the CRD API resource.

Network attachment definition (NAD)

A CRD introduced by the Multus project that allows you to attach pods, virtual machines, and virtual machine instances to one or more networks.

UserDefinedNetwork (UDN)

A namespace-scoped CRD introduced by the user-defined network API that can be used to create a tenant network that isolates the tenant namespace from other namespaces.

ClusterUserDefinedNetwork (CUDN)

A cluster-scoped CRD introduced by the user-defined network API that cluster administrators can use to create a shared network across multiple namespaces.

Node network configuration policy (NNCP)

A CRD introduced by the nmstate project, describing the requested network configuration on nodes. You update the node network configuration, including adding and removing interfaces, by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

10.15.7. Using huge pages with virtual machines

You can use huge pages as backing memory for virtual machines in your cluster.

10.15.7.1. What huge pages do

Memory is managed in blocks known as pages. On most systems, a page is 4Ki. 1Mi of memory is equal to 256 pages; 1Gi of memory is 256,000 pages, and so on. CPUs have a built-in memory management unit that manages a list of these pages in hardware. The Translation Lookaside Buffer (TLB) is a small hardware cache of virtual-to-physical page mappings. If the virtual address passed in a hardware instruction can be found in the TLB, the mapping can be determined quickly. If not, a TLB miss occurs, and the system falls back to slower, software-based address translation, resulting in performance issues. Since the size of the TLB is fixed, the only way to reduce the chance of a TLB miss is to increase the page size.

A huge page is a memory page that is larger than 4Ki. On x86_64 architectures, there are two common huge page sizes: 2Mi and 1Gi. Sizes vary on other architectures. To use huge pages, code must be written so that applications are aware of them. Transparent Huge Pages (THP) attempt to automate the management of huge pages without application knowledge, but they have limitations. In particular, they are limited to 2Mi page sizes. THP can lead to performance degradation on nodes with high memory utilization or fragmentation due to defragmenting efforts of THP, which can lock memory pages. For this reason, some applications may be designed to (or recommend) usage of pre-allocated huge pages instead of THP.

In OpenShift Virtualization, virtual machines can be configured to consume pre-allocated huge pages.

10.15.7.2. Configuring huge pages for virtual machines

You can configure virtual machines to use pre-allocated huge pages by including the **memory.hugepages.pageSize** and **resources.requests.memory** parameters in your virtual machine configuration.

The memory request must be divisible by the page size. For example, you cannot request **500Mi** memory with a page size of **1Gi**.

NOTE

The memory layouts of the host and the guest OS are unrelated. Huge pages requested in the virtual machine manifest apply to QEMU. Huge pages inside the guest can only be configured based on the amount of available memory of the virtual machine instance.

If you edit a running virtual machine, the virtual machine must be rebooted for the changes to take effect.

Prerequisites

- Nodes must have pre-allocated huge pages configured.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. In your virtual machine configuration, add the **resources.requests.memory** and **memory.hugepages.pageSize** parameters to the **spec.domain**. The following configuration snippet is for a virtual machine that requests a total of **4Gi** memory with a page size of **1Gi**:

```
kind: VirtualMachine
# ...
spec:
  domain:
    resources:
      requests:
        memory: "4Gi" (1)
    memory:
      hugepages:
        pageSize: "1Gi" (2)
# ...
```

(1) The total amount of memory requested for the virtual machine. This value must be divisible by the page size.

(2) The size of each huge page. Valid values for x86_64 architecture are **1Gi** and **2Mi**. The page size must be smaller than the requested memory.

2. Apply the virtual machine configuration:

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

10.15.8. Enabling dedicated resources for virtual machines

To improve performance, you can dedicate node resources, such as CPU, to a virtual machine.

10.15.8.1. About dedicated resources

When you enable dedicated resources for your virtual machine, your virtual machine's workload is scheduled on CPUs that will not be used by other processes. By using dedicated resources, you can improve the performance of the virtual machine and the accuracy of latency predictions.

10.15.8.2. Enabling dedicated resources for a virtual machine

You enable dedicated resources for a virtual machine in the **Details** tab. Virtual machines that were created from a Red Hat template can be configured with dedicated resources.

Prerequisites

- The CPU Manager must be configured on the node. Verify that the node has the **cpumanager = true** label before scheduling virtual machine workloads.
- The virtual machine must be powered off.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization → VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. On the **Configuration → Scheduling** tab, click the edit icon beside **Dedicated Resources**.
4. Select **Schedule this workload with dedicated resources (guaranteed policy)**
5. Click **Save**.

10.15.9. Scheduling virtual machines

You can schedule a virtual machine (VM) on a node by ensuring that the VM's CPU model and policy attribute are matched for compatibility with the CPU models and policy attributes supported by the node.

10.15.9.1. Policy attributes

You can schedule a virtual machine (VM) by specifying a policy attribute and a CPU feature that is matched for compatibility when the VM is scheduled on a node. A policy attribute specified for a VM determines how that VM is scheduled on a node.

Policy attribute	Description
force	The VM is forced to be scheduled on a node. This is true even if the host CPU does not support the VM's CPU.
require	Default policy that applies to a VM if the VM is not configured with a specific CPU model and feature specification. If a node is not configured to support CPU node discovery with this default policy attribute or any one of the other policy attributes, VMs are not scheduled on that node. Either the host CPU must support the VM's CPU or the hypervisor must be able to emulate the supported CPU model.
optional	The VM is added to a node if that VM is supported by the host's physical machine CPU.
disable	The VM cannot be scheduled with CPU node discovery.

Policy attribute	Description
forbid	The VM is not scheduled even if the feature is supported by the host CPU and CPU node discovery is enabled.

10.15.9.2. Setting a policy attribute and CPU feature

You can set a policy attribute and CPU feature for each virtual machine (VM) to ensure that it is scheduled on a node according to policy and feature. The CPU feature that you set is verified to ensure that it is supported by the host CPU or emulated by the hypervisor.

Procedure

- Edit the **domain** spec of your VM configuration file. The following example sets the CPU feature and the **require** policy for a virtual machine (VM):

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          features:
            - name: apic (1)
              policy: require (2)
```

(1) Name of the CPU feature for the VM.

(2) Policy attribute for the VM.

10.15.9.3. Scheduling virtual machines with the supported CPU model

You can configure a CPU model for a virtual machine (VM) to schedule it on a node where its CPU model is supported.

Procedure

- Edit the **domain** spec of your virtual machine configuration file. The following example shows a specific CPU model defined for a VM:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
```

```
    domain:
      cpu:
        model: Conroe 1
```

1 CPU model for the VM.

10.15.9.4. Scheduling virtual machines with the host model

When the CPU model for a virtual machine (VM) is set to **host-model**, the VM inherits the CPU model of the node where it is scheduled.

Procedure

- Edit the **domain** spec of your VM configuration file. The following example shows **host-model** being specified for the virtual machine:

```
apiVersion: kubevirt/v1alpha3
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          model: host-model 1
```

1 The VM that inherits the CPU model of the node where it is scheduled.

10.15.9.5. Scheduling virtual machines with a custom scheduler

You can use a custom scheduler to schedule a virtual machine (VM) on a node.

Prerequisites

- A secondary scheduler is configured for your cluster.
- You have installed the OpenShift CLI (**oc**).

Procedure

- Add the custom scheduler to the VM configuration by editing the **VirtualMachine** manifest. For example:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-fedora
spec:
  runStrategy: Always
  template:
    spec:
```

```
    schedulerName: my-scheduler (1)
    domain:
      devices:
        disks:
          - name: containerdisk
            disk:
              bus: virtio
# ...
```

(1) The name of the custom scheduler. If the **schedulerName** value does not match an existing scheduler, the **virt-launcher** pod stays in a **Pending** state until the specified scheduler is found.

Verification

- Verify that the VM is using the custom scheduler specified in the **VirtualMachine** manifest by checking the **virt-launcher** pod events:

  a. View the list of pods in your cluster by entering the following command:

  ```
  $ oc get pods
  ```

  Example output

  ```
  NAME                            READY   STATUS    RESTARTS   AGE
  virt-launcher-vm-fedora-dpc87   2/2     Running   0          24m
  ```

  b. Run the following command to display the pod events:

  ```
  $ oc describe pod virt-launcher-vm-fedora-dpc87
  ```

  The value of the **From** field in the output verifies that the scheduler name matches the custom scheduler specified in the **VirtualMachine** manifest:

  Example output

  ```
  [...]
  Events:
    Type    Reason     Age   From            Message
    ----    ------     ----  ----            -------
    Normal  Scheduled  21m   my-scheduler  Successfully assigned default/virt-launcher-vm-fedora-dpc87 to node01
  [...]
  ```

Additional resources

- Deploying a secondary scheduler

10.15.10. Configuring PCI passthrough

The Peripheral Component Interconnect (PCI) passthrough feature enables you to access and manage hardware devices from a virtual machine (VM). When PCI passthrough is configured, the PCI devices function as if they were physically attached to the guest operating system.

Cluster administrators can expose and manage host devices that are permitted to be used in the cluster by using the **oc** command-line interface (CLI).

10.15.10.1. Preparing nodes for GPU passthrough

You can prevent GPU operands from deploying on worker nodes that you designated for GPU passthrough.

10.15.10.1.1. Preventing NVIDIA GPU operands from deploying on nodes

If you use the NVIDIA GPU Operator in your cluster, you can apply the **nvidia.com/gpu.deploy.operands=false** label to nodes that you do not want to configure for GPU or vGPU operands. This label prevents the creation of the pods that configure GPU or vGPU operands and terminates the pods if they already exist.

Prerequisites

- The OpenShift CLI (**oc**) is installed.

Procedure

- Label the node by running the following command:

```
$ oc label node <node_name> nvidia.com/gpu.deploy.operands=false 1
```

1 Replace **<node_name>** with the name of a node where you do not want to install the NVIDIA GPU operands.

Verification

1. Verify that the label was added to the node by running the following command:

```
$ oc describe node <node_name>
```

2. Optional: If GPU operands were previously deployed on the node, verify their removal.

   a. Check the status of the pods in the **nvidia-gpu-operator** namespace by running the following command:

```
$ oc get pods -n nvidia-gpu-operator
```

Example output

```
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. Monitor the pod status until the pods with **Terminating** status are removed:

```
$ oc get pods -n nvidia-gpu-operator
```

Example output

```
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
```

10.15.10.2. Preparing host devices for PCI passthrough

10.15.10.2.1. About preparing a host device for PCI passthrough

To prepare a host device for PCI passthrough by using the CLI, create a **MachineConfig** object and add kernel arguments to enable the Input-Output Memory Management Unit (IOMMU). Bind the PCI device to the Virtual Function I/O (VFIO) driver and then expose it in the cluster by editing the **permittedHostDevices** field of the **HyperConverged** custom resource (CR). The **permittedHostDevices** list is empty when you first install the OpenShift Virtualization Operator.

To remove a PCI host device from the cluster by using the CLI, delete the PCI device information from the **HyperConverged** CR.

10.15.10.2.2. Adding kernel arguments to enable the IOMMU driver

To enable the IOMMU driver in the kernel, create the **MachineConfig** object and add the kernel arguments.

Prerequisites

- You have cluster administrator permissions.
- Your CPU hardware is Intel or AMD.
- You enabled Intel Virtualization Technology for Directed I/O extensions or AMD IOMMU in the BIOS.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **MachineConfig** object that identifies the kernel argument. The following example shows a kernel argument for an Intel CPU.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
```

```
    labels:
      machineconfiguration.openshift.io/role: worker
    name: 100-worker-iommu
  spec:
    config:
      ignition:
        version: 3.2.0
    kernelArguments:
        - intel_iommu=on
  # ...
```

where:

<apiversion>

Applies the new kernel argument only to worker nodes.

<name>

Indicates the ranking of this kernel argument (100) among the machine configs and its purpose. If you have an AMD CPU, specify the kernel argument as **amd_iommu=on**.

<intel_iommu=o>

Identifies the kernel argument as **intel_iommu** for an Intel CPU.

2. Create the new **MachineConfig** object:

```
$ oc create -f 100-worker-kernel-arg-iommu.yaml
```

Verification

1. Verify that the new **MachineConfig** object was added by entering the following command and observing the output:

```
$ oc get MachineConfig
```

Example output

```
NAME                                        IGNITIONVERSION     AGE
00-master                                   3.5.0               164m
00-worker                                   3.5.0               164m
01-master-container-runtime                 3.5.0               164m
01-master-kubelet                           3.5.0               164m
01-worker-container-runtime                 3.5.0               164m
01-worker-kubelet                           3.5.0               164m
100-master-chrony-configuration             3.5.0               169m
100-master-set-core-user-password           3.5.0               169m
100-worker-chrony-configuration             3.5.0               169m
100-worker-iommu                            3.5.0               14s
```

2. Verify that IOMMU is enabled at the operating system (OS) level by entering the following command:

```
$ dmesg | grep -i iommu
```

- If IOMMU is enabled, output is displayed as shown in the following example:

Example output

```
Intel: [ 0.000000] DMAR: Intel(R) IOMMU Driver
AMD: [ 0.000000] AMD-Vi: IOMMU Initialized
```

10.15.10.2.3. Binding PCI devices to the VFIO driver

To bind PCI devices to the VFIO (Virtual Function I/O) driver, obtain the values for **vendor-ID** and **device-ID** from each device and create a list with the values. Add this list to the **MachineConfig** object. The **MachineConfig** Operator generates the **/etc/modprobe.d/vfio.conf** on the nodes with the PCI devices, and binds the PCI devices to the VFIO driver.

Prerequisites

- You added kernel arguments to enable IOMMU for the CPU.
- You have installed the OpenShift CLI (**oc**).

Procedure

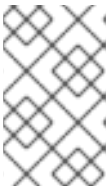
1. Run the **lspci** command to obtain the **vendor-ID** and the **device-ID** for the PCI device.

```
$ lspci -nnv | grep -i nvidia
```

Example output

```
02:01.0 3D controller [0302]: NVIDIA Corporation GV100GL [Tesla V100 PCIe 32GB]
[10de:1eb8] (rev a1)
```

2. Create a Butane config file, **100-worker-vfiopci.bu**, binding the PCI device to the VFIO driver.



NOTE

The Butane version you specify in the config file should match the OpenShift Container Platform version and always ends in **0**. For example, **4.20.0**. See "Creating machine configs with Butane" for information about Butane.

Example

```
variant: openshift
version: 4.20.0
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) Applies the new kernel argument only to worker nodes.

(2) Specify the previously determined **vendor-ID** value (**10de**) and the **device-ID** value (**1eb8**) to bind a single device to the VFIO driver. You can add a list of multiple devices with their vendor and device information.

(3) The file that loads the vfio-pci kernel module on the worker nodes.

3. Use Butane to generate a **MachineConfig** object file, **100-worker-vfiopci.yaml**, containing the configuration to be delivered to the worker nodes:

```
$ butane 100-worker-vfiopci.bu -o 100-worker-vfiopci.yaml
```

4. Apply the **MachineConfig** object to the worker nodes:

```
$ oc apply -f 100-worker-vfiopci.yaml
```

5. Verify that the **MachineConfig** object was added.

```
$ oc get MachineConfig
```

Example output

```
NAME                             GENERATEDBYCONTROLLER                      IGNITIONVERSION
AGE
00-master                        d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.5.0             25h
00-worker                        d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.5.0             25h
01-master-container-runtime      d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.5.0
25h
01-master-kubelet                d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.5.0
25h
01-worker-container-runtime      d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.5.0
25h
01-worker-kubelet                d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.5.0
25h
100-worker-iommu                                                            3.5.0             30s
100-worker-vfiopci-configuration                                            3.5.0             30s
```

Verification

- Verify that the VFIO driver is loaded.

```
$ lspci -nnk -d 10de:
```

The output confirms that the VFIO driver is being used.

Example output

```
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
```

10.15.10.2.4. Exposing PCI host devices in the cluster using the CLI

To expose PCI host devices in the cluster, add details about the PCI devices to the **spec.permittedHostDevices.pciHostDevices** array of the **HyperConverged** custom resource (CR).

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **HyperConverged** CR in your default editor by running the following command:

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

2. Add the PCI device information to the **spec.permittedHostDevices.pciHostDevices** array. For example:

Example configuration file

```
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. The host devices that are permitted to be used in the cluster.
2. The list of PCI devices available on the node.
3. The **vendor-ID** and the **device-ID** required to identify the PCI device.
4. The name of a PCI host device.
5.

Optional: Setting this field to **true** indicates that the resource is provided by an external device plugin. OpenShift Virtualization allows the usage of this device in the cluster but

NOTE

The above example snippet shows two PCI host devices that are named **nvidia.com/GV100GL_Tesla_V100** and **nvidia.com/TU104GL_Tesla_T4** added to the list of permitted host devices in the **HyperConverged** CR. These devices have been tested and verified to work with OpenShift Virtualization.

3. Save your changes and exit the editor.

Verification

- Verify that the PCI host devices were added to the node by running the following command. The example output shows that there is one device each associated with the **nvidia.com/GV100GL_Tesla_V100**, **nvidia.com/TU104GL_Tesla_T4**, and **intel.com/qat** resource names.

```
$ oc describe node <node_name>
```

Example output

```
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
```

10.15.10.2.5. Removing PCI host devices from the cluster using the CLI

To remove a PCI host device from the cluster, delete the information for that device from the **HyperConverged** custom resource (CR).

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **HyperConverged** CR in your default editor by running the following command:

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

2. Remove the PCI device information from the **spec.permittedHostDevices.pciHostDevices** array by deleting the **pciDeviceSelector**, **resourceName** and **externalResourceProvider** (if applicable) fields for the appropriate device. In this example, the **intel.com/qat** resource has been deleted.

Example configuration file

```
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. Save your changes and exit the editor.

Verification

- Verify that the PCI host device was removed from the node by running the following command. The example output shows that there are zero devices associated with the **intel.com/qat** resource name.

```
$ oc describe node <node_name>
```

Example output

```
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
```

10.15.10.3. Configuring virtual machines for PCI passthrough

After the PCI devices have been added to the cluster, you can assign them to virtual machines. The PCI devices are now available as if they are physically connected to the virtual machines.

10.15.10.3.1. Assigning a PCI device to a virtual machine

When a PCI device is available in a cluster, you can assign it to a virtual machine and enable PCI passthrough.

Procedure

- Assign the PCI device to a virtual machine as a host device.

  Example

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  domain:
    devices:
      hostDevices:
      - deviceName: nvidia.com/TU104GL_Tesla_T4 1
        name: hostdevices1
```

1 The name of the PCI device that is permitted on the cluster as a host device. The virtual machine can access this host device.

Verification

- Use the following command to verify that the host device is available from the virtual machine.

```
$ lspci -nnk | grep NVIDIA
```

Example output

```
$ 02:01.0 3D controller [0302]: NVIDIA Corporation GV100GL [Tesla V100 PCIe 32GB]
[10de:1eb8] (rev a1)
```

10.15.10.4. Additional resources

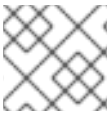
- Enabling Intel VT-X and AMD-V Virtualization Hardware Extensions in BIOS
- Managing file permissions
- Machine Config Overview

10.15.11. Configuring virtual GPUs

If you have graphics processing unit (GPU) cards, OpenShift Virtualization can automatically create virtual GPUs (vGPUs) that you can assign to virtual machines (VMs).

10.15.11.1. About using virtual GPUs with OpenShift Virtualization

Some graphics processing unit (GPU) cards support the creation of virtual GPUs (vGPUs). OpenShift Virtualization can automatically create vGPUs and other mediated devices if an administrator provides configuration details in the **HyperConverged** custom resource (CR). This automation is especially useful for large clusters.



NOTE

Refer to your hardware vendor's documentation for functionality and support details.

Mediated device

A physical device that is divided into one or more virtual devices. A vGPU is a type of mediated device (mdev); the performance of the physical GPU is divided among the virtual devices. You can assign mediated devices to one or more virtual machines (VMs), but the number of guests must be compatible with your GPU. Some GPUs do not support multiple guests.

10.15.11.2. Preparing hosts for mediated devices

You must enable the Input-Output Memory Management Unit (IOMMU) driver before you can configure mediated devices.

10.15.11.2.1. Adding kernel arguments to enable the IOMMU driver

To enable the IOMMU driver in the kernel, create the **MachineConfig** object and add the kernel arguments.

Prerequisites

- You have cluster administrator permissions.
- Your CPU hardware is Intel or AMD.
- You enabled Intel Virtualization Technology for Directed I/O extensions or AMD IOMMU in the BIOS.

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **MachineConfig** object that identifies the kernel argument. The following example shows a kernel argument for an Intel CPU.

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 100-worker-iommu
spec:
  config:
    ignition:
      version: 3.2.0
  kernelArguments:
      - intel_iommu=on
# ...
```

where:

<apiversion>

Applies the new kernel argument only to worker nodes.

<name>

Indicates the ranking of this kernel argument (100) among the machine configs and its purpose. If you have an AMD CPU, specify the kernel argument as **amd_iommu=on**.

<intel_iommu=o>

Identifies the kernel argument as **intel_iommu** for an Intel CPU.

2. Create the new **MachineConfig** object:

```
$ oc create -f 100-worker-kernel-arg-iommu.yaml
```

Verification

1. Verify that the new **MachineConfig** object was added by entering the following command and observing the output:

```
$ oc get MachineConfig
```

Example output

```
NAME                                        IGNITIONVERSION                AGE
00-master                                   3.5.0                          164m
00-worker                                   3.5.0                          164m
01-master-container-runtime                 3.5.0                          164m
01-master-kubelet                           3.5.0                          164m
01-worker-container-runtime                 3.5.0                          164m
01-worker-kubelet                           3.5.0                          164m
100-master-chrony-configuration             3.5.0                          169m
```

```
100-master-set-core-user-password          3.5.0                 169m
100-worker-chrony-configuration            3.5.0                 169m
100-worker-iommu                           3.5.0                 14s
```

2. Verify that IOMMU is enabled at the operating system (OS) level by entering the following command:

```
$ dmesg | grep -i iommu
```

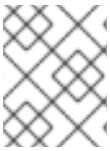
- If IOMMU is enabled, output is displayed as shown in the following example:

Example output

```
Intel: [ 0.000000] DMAR: Intel(R) IOMMU Driver
AMD: [ 0.000000] AMD-Vi: IOMMU Initialized
```

10.15.11.3. Configuring the NVIDIA GPU Operator

You can use the NVIDIA GPU Operator to provision worker nodes for running GPU-accelerated virtual machines (VMs) in OpenShift Virtualization.



NOTE

The NVIDIA GPU Operator is supported only by NVIDIA. For more information, see Obtaining Support from NVIDIA in the Red Hat Knowledgebase.

10.15.11.3.1. About using the NVIDIA GPU Operator

You can use the NVIDIA GPU Operator with OpenShift Virtualization to rapidly provision worker nodes for running GPU-enabled virtual machines (VMs). The NVIDIA GPU Operator manages NVIDIA GPU resources in an OpenShift Container Platform cluster and automates tasks that are required when preparing nodes for GPU workloads.

Before you can deploy application workloads to a GPU resource, you must install components such as the NVIDIA drivers that enable the compute unified device architecture (CUDA), Kubernetes device plugin, container runtime, and other features, such as automatic node labeling and monitoring. By automating these tasks, you can quickly scale the GPU capacity of your infrastructure. The NVIDIA GPU Operator can especially facilitate provisioning complex artificial intelligence and machine learning (AI/ML) workloads.

10.15.11.3.2. Options for configuring mediated devices

There are two available methods for configuring mediated devices when using the NVIDIA GPU Operator. The method that Red Hat tests uses OpenShift Virtualization features to schedule mediated devices, while the NVIDIA method only uses the GPU Operator.

Using the NVIDIA GPU Operator to configure mediated devices

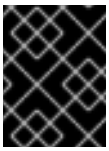
This method exclusively uses the NVIDIA GPU Operator to configure mediated devices. To use this method, refer to NVIDIA GPU Operator with OpenShift Virtualization in the NVIDIA documentation.

Using OpenShift Virtualization to configure mediated devices

This method, which is tested by Red Hat, uses OpenShift Virtualization's capabilities to configure mediated devices. In this case, the NVIDIA GPU Operator is only used for installing drivers with the NVIDIA vGPU Manager. The GPU Operator does not configure mediated devices.

When using the OpenShift Virtualization method, you still configure the GPU Operator by following the NVIDIA documentation. However, this method differs from the NVIDIA documentation in the following ways:

- You must not overwrite the default **disableMDEVConfiguration: false** setting in the **HyperConverged** custom resource (CR).



IMPORTANT

Setting this feature gate as described in the NVIDIA documentation prevents OpenShift Virtualization from configuring mediated devices.

- You must configure your **ClusterPolicy** manifest so that it matches the following example:

Example manifest

```
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. Set this value to **false**. Not required for VMs.
2. Set this value to **true**. Required for using vGPUs with VMs.
3. Substitute **<vgpu_container_registry>** with your registry value.
4. Set this value to **false** to allow OpenShift Virtualization to configure mediated devices instead of the NVIDIA GPU Operator.
5. Set this value to **false** to prevent discovery and advertising of the vGPU devices to the kubelet.
6. Set this value to **false** to prevent loading the **vfio-pci** driver. Instead, follow the OpenShift Virtualization documentation to configure PCI passthrough.

Additional resources

- Configuring PCI passthrough

10.15.11.4. How vGPUs are assigned to nodes

For each physical device, OpenShift Virtualization configures the following values:

- A single mdev type.
- The maximum number of instances of the selected **mdev** type.

The cluster architecture affects how devices are created and assigned to nodes.

Large cluster with multiple cards per node

On nodes with multiple cards that can support similar vGPU types, the relevant device types are created in a round-robin manner. For example:

```
# ...
mediatedDevicesConfiguration:
  mediatedDeviceTypes:
  - nvidia-222
  - nvidia-228
  - nvidia-105
  - nvidia-108
# ...
```

In this scenario, each node has two cards, both of which support the following vGPU types:

-

```
nvidia-105
# ...
nvidia-108
nvidia-217
nvidia-299
# ...
```

On each node, OpenShift Virtualization creates the following vGPUs:

- 16 vGPUs of type nvidia-105 on the first card.
- 2 vGPUs of type nvidia-108 on the second card.

One node has a single card that supports more than one requested vGPU type

OpenShift Virtualization uses the supported type that comes first on the **mediatedDeviceTypes** list. For example, the card on a node card supports **nvidia-223** and **nvidia-224**. The following **mediatedDeviceTypes** list is configured:

```
# ...
mediatedDevicesConfiguration:
  mediatedDeviceTypes:
  - nvidia-22
  - nvidia-223
  - nvidia-224
# ...
```

In this example, OpenShift Virtualization uses the **nvidia-223** type.

10.15.11.5. Managing mediated devices

Before you can assign mediated devices to virtual machines, you must create the devices and expose them to the cluster. You can also reconfigure and remove mediated devices.

10.15.11.5.1. Creating and exposing mediated devices

As an administrator, you can create mediated devices and expose them to the cluster by editing the **HyperConverged** custom resource (CR). Before you edit the CR, explore a worker node to find the configuration values that are specific to your hardware devices.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You enabled the Input-Output Memory Management Unit (IOMMU) driver.
- If your hardware vendor provides drivers, you installed them on the nodes where you want to create mediated devices.
  - If you use NVIDIA cards, you installed the NVIDIA GRID driver.

Procedure

1. Identify the name selector and resource name values for the mediated devices by exploring a worker node:

   a. Start a debugging session with the worker node by using the **oc debug** command. For example:

   ```
   $ oc debug node/node-11.redhat.com
   ```

   b. Change the root directory of the shell process to the file system of the host node by running the following command:

   ```
   # chroot /host
   ```

   c. Navigate to the **mdev_bus** directory and view its contents. Each subdirectory name is a PCI address of a physical GPU. For example:

   ```
   # cd sys/class/mdev_bus && ls
   ```

   Example output:

   ```
   0000:4b:00.4
   ```

   d. Go to the directory for your physical device and list the supported mediated device types as defined by the hardware vendor. For example:

   ```
   # cd 0000:4b:00.4 && ls mdev_supported_types
   ```

   Example output:

   ```
   nvidia-742  nvidia-744 nvidia-746  nvidia-748 nvidia-750  nvidia-752
   nvidia-743  nvidia-745 nvidia-747  nvidia-749 nvidia-751  nvidia-753
   ```

   e. Select the mediated device type that you want to use and identify its name selector value by viewing the contents of its **name** file. For example:

   ```
   # cat nvidia-745/name
   ```

   Example output:

   ```
   NVIDIA A2-2Q
   ```

2. Open the **HyperConverged** CR in your default editor by running the following command:

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

3. Create and expose the mediated devices by updating the configuration:

   a. Create mediated devices by adding them to the **spec.mediatedDevicesConfiguration** stanza.

   b. Expose the mediated devices to the cluster by adding the **mdevNameSelector** and **resourceName** values to the **spec.permittedHostDevices.mediatedDevices** stanza. The

resourceName value is based on the **mdevNameSelector** value, but you use underscores instead of spaces.
Example **HyperConverged** CR:

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  mediatedDevicesConfiguration:
    mediatedDeviceTypes:
    - nvidia-745
    nodeMediatedDeviceTypes:
    - mediatedDeviceTypes:
      - nvidia-746
      nodeSelector:
        kubernetes.io/hostname: node-11.redhat.com
  permittedHostDevices:
    mediatedDevices:
    - mdevNameSelector: NVIDIA A2-2Q
      resourceName: nvidia.com/NVIDIA_A2-2Q
    - mdevNameSelector: NVIDIA A2-4Q
      resourceName: nvidia.com/NVIDIA_A2-4Q
# ...
```

where:

mediatedDeviceTypes

Specifies global settings for the cluster and is required.

nodeMediatedDeviceTypes

Specifies global configuration overrides for a specific node or group of nodes and is optional. Must be used with the global **mediatedDeviceTypes** configuration.

mediatedDeviceTypes

Specifies an override to the global **mediatedDeviceTypes** configuration for the specified nodes. Required if you use **nodeMediatedDeviceTypes**.

nodeSelector

Specifies the node selector and must include a **key:value** pair. Required if you use **nodeMediatedDeviceTypes**.

mdevNameSelector

Specifies the mediated devices that map to this value on the host.

resourceName

Specifies the matching resource name that is allocated on the node.

4. Save your changes and exit the editor.

Verification

- Confirm that the virtual GPU is attached to the node by running the following command:

```
$ oc get node <node_name> -o json \
```

```
| jq '.status.allocatable \
| with_entries(select(.key | startswith("nvidia.com/"))) \
| with_entries(select(.value != "0"))'
```

10.15.11.5.2. About changing and removing mediated devices

You can reconfigure or remove mediated devices in several ways:

- Edit the **HyperConverged** CR and change the contents of the **mediatedDeviceTypes** stanza.
- Change the node labels that match the **nodeMediatedDeviceTypes** node selector.
- Remove the device information from the **spec.mediatedDevicesConfiguration** and **spec.permittedHostDevices** stanzas of the **HyperConverged** CR.

NOTE

If you remove the device information from the **spec.permittedHostDevices** stanza without also removing it from the **spec.mediatedDevicesConfiguration** stanza, you cannot create a new mediated device type on the same node. To properly remove mediated devices, remove the device information from both stanzas.

10.15.11.5.3. Removing mediated devices from the cluster

To remove a mediated device from the cluster, delete the information for that device from the **HyperConverged** custom resource (CR).

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **HyperConverged** CR in your default editor by running the following command:

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

2. Remove the device information from the **spec.mediatedDevicesConfiguration** and **spec.permittedHostDevices** stanzas of the **HyperConverged** CR. Removing both entries ensures that you can later create a new mediated device type on the same node. For example:

Example configuration file

```
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. To remove the **nvidia-231** device type, delete it from the **mediatedDeviceTypes** array.
2. To remove the **GRID T4-2Q** device, delete the **mdevNameSelector** field and its corresponding **resourceName** field.

3. Save your changes and exit the editor.

10.15.11.6. Using mediated devices

You can assign mediated devices to one or more virtual machines.

10.15.11.6.1. Assigning a vGPU to a VM by using the CLI

Assign mediated devices such as virtual GPUs (vGPUs) to virtual machines (VMs).

Prerequisites

- The mediated device is configured in the **HyperConverged** custom resource.
- The VM is stopped.

Procedure

- Assign the mediated device to a virtual machine (VM) by editing the **spec.domain.devices.gpus** stanza of the **VirtualMachine** manifest:

  Example virtual machine manifest

```
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. The resource name associated with the mediated device.
2. A name to identify the device on the VM.

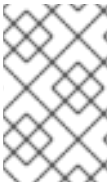
Verification

- To verify that the device is available from the virtual machine, run the following command, substituting **<device_name>** with the **deviceName** value from the **VirtualMachine** manifest:

```
$ lspci -nnk | grep <device_name>
```

10.15.11.6.2. Assigning a vGPU to a VM by using the web console

You can assign virtual GPUs to virtual machines by using the OpenShift Container Platform web console.



NOTE

You can add hardware devices to virtual machines created from customized templates or a YAML file. You cannot add devices to pre-supplied boot source templates for specific operating systems.

Prerequisites

- The vGPU is configured as a mediated device in your cluster.
  - To view the devices that are connected to your cluster, click **Compute → Hardware Devices** from the side menu.
- The VM is stopped.

Procedure

1. In the OpenShift Container Platform web console, click **Virtualization → VirtualMachines** from the side menu.
2. Select the VM that you want to assign the device to.
3. On the **Details** tab, click **GPU devices**.
4. Click **Add GPU device**.
5. Enter an identifying value in the **Name** field.
6. From the **Device name** list, select the device that you want to add to the VM.
7. Click **Save**.

Verification

- To confirm that the devices were added to the VM, click the **YAML** tab and review the **VirtualMachine** configuration. Mediated devices are added to the **spec.domain.devices** stanza.

10.15.11.7. Additional resources

- Enabling Intel VT-X and AMD-V Virtualization Hardware Extensions in BIOS

10.15.12. Configuring USB host passthrough

As a cluster administrator, you can expose USB devices in a cluster, which makes the devices available for virtual machine (VM) owners to assign to VMs. Enabling this passthrough of USB devices allows a VM to connect to USB hardware that is attached to an OpenShift Container Platform node, as if the

hardware and the VM are physically connected.

To expose a USB device, first enable host passthrough and then configure the VM to use the USB device.

10.15.12.1. Enabling USB host passthrough

To attach a USB device to a virtual machine (VM), you must first enable USB host passthrough at the cluster level.

To do this, specify a resource name and USB device name for each device you want first to add and then assign to a VM. You can allocate more than one device, each of which is known as a **selector** in the **HyperConverged** custom resource (CR), to a single resource name. If you have multiple identical USB devices on the cluster, you can choose to allocate a VM to a specific device.

Prerequisites

- You have access to an OpenShift Container Platform cluster as a user who has the **cluster-admin** role.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Ensure that the **HostDevices** feature gate is enabled:

```
$ oc get featuregate cluster -o yaml
```

Successful output

```
  featureGates:
# ...
    enabled:
    - name: HostDevices
```

2. Identify the USB device vendor and product:

```
$ lsusb
```

Example output

```
Bus 003 Device 007: ID 1b1c:0a60 example_manufacturer example_product_name
```

- If you cannot use the **lsusb** command, inspect the USB device configurations in the host's **/sys/bus/usb/devices/** directory:

```
for dev in *; do
    if [[ -f "$dev/idVendor" && -f "$dev/idProduct" ]]; then
        echo "Device: $dev"
        echo -n "  Manufacturer : "; cat "$dev/manufacturer"
        echo -n "  Product: "; cat "$dev/product"
        echo -n "  Vendor ID : "; cat "$dev/idVendor"
        echo -n "  Product ID: "; cat "$dev/idProduct"
```

```
        echo
    fi
done
```

Example output

```
Device: 3-7
  Manufacturer : example_manufacturer
  Product: example_product_name
  Vendor ID : 1b1c
  Product ID: 0a60
```

3. Add the required USB device to the **permittedHostDevices** stanza of the **HyperConvered** CR. The following example adds a device with vendor ID **045e** and product ID **07a5**:

```
oc patch hyperconverged kubevirt-hyperconverged \
  -n openshift-cnv \
  --type=merge \
  -p '{
    "metadata": {
      "annotations": {
        "kubevirt.kubevirt.io/jsonpatch": "[{\"op\": \"add\", \"path\":
\"/spec/permittedHostDevices/usbHostDevices/-\", \"value\": {\"resourceName\":
\"kubevirt.io/peripherals\", \"selectors\": [{\"vendor\": \"045e\", \"product\": \"07a5\"}]}}]"
      }
    }
  }'
```

Verification

- Ensure that the HCO CR contains the required USB devices:

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

Example output

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  permittedHostDevices: 1
    usbHostDevices: 2
      - resourceName: kubevirt.io/peripherals 3
        selectors:
          - vendor: "045e"
            product: "07a5"
          - vendor: "062a"
            product: "4102"
          - vendor: "072f"
            product: "b100"
```

1. Lists the host devices that have permission to be used in the cluster.
2. Lists the available USB devices.
3. Uses **resourceName: deviceName** for each device you want to add and assign to the VM. In this example, the resource is bound to three devices, each of which is identified by **vendor** and **product** and is known as a **selector**.

10.15.12.2. Connecting a USB device to a virtual machine

You can configure virtual machine (VM) access to a USB device. This configuration enables the VM to connect to USB hardware that is attached to an OpenShift Container Platform node, as if the hardware and the VM are physically connected.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have attached the required USB device as a resource at the cluster level.

Procedure

1. In the **HyperConverged** custom resource (CR), find the assigned resource name of the USB device:

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

Example output

```
# ...
  spec:
    permittedHostDevices:
      usbHostDevices:
        - resourceName: kubevirt.io/peripherals
          selectors:
            - vendor: "045e"
              product: "07a5"
            - vendor: "062a"
              product: "4102"
            - vendor: "072f"
              product: "b100"
```

2. Open the VM instance CR:

```
$ oc edit vmi <vmi_usb>
```

where:

<vmi_usb>

Specifies the name of the **VirtualMachineInstance** CR.

3. Edit the CR by adding the USB device, as shown in the following example:
Example configuration

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  labels:
    special: vmi-usb
  name: vmi-usb
spec:
  domain:
    devices:
      hostDevices:
      - deviceName: kubevirt.io/peripherals
        name: local-peripherals (1)
# ...
```

(1) The name of the USB device.

4. Apply the modifications to the VM configurations:

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

where:

<filename>

Specifies the name of the **VirtualMachineInstance** manifest YAML file.

10.15.13. Enabling descheduler evictions on virtual machines

You can use the descheduler to evict pods so that the pods can be rescheduled onto more appropriate nodes. If the pod is a virtual machine, the pod eviction causes the virtual machine to be live migrated to another node.

10.15.13.1. Descheduler profiles

Use the **KubeVirtRelieveAndMigrate** or **LongLifecycle** profile to enable the descheduler on a virtual machine.

IMPORTANT

You cannot have both **KubeVirtRelieveAndMigrate** and **LongLifeCycle** enabled at the same time.

KubeVirtRelieveAndMigrate

This profile is an enhanced version of the **LongLifeCycle** profile.

The **KubeVirtRelieveAndMigrate** profile evicts pods from high-cost nodes to reduce overall resource expenses and enable workload migration. It also periodically rebalances workloads to help maintain similar spare capacity across nodes, which supports better handling of sudden workload spikes. Nodes can experience the following costs:

- **Resource utilization**: Increased resource pressure raises the overhead for running applications.

- **Node maintenance**: A higher number of containers on a node increases resource consumption and maintenance costs.

The profile enables the **LowNodeUtilization** strategy with the **EvictionsInBackground** alpha feature. The profile also exposes the following customization fields:

- **devActualUtilizationProfile**: Enables load-aware descheduling.
- **devLowNodeUtilizationThresholds**: Sets experimental thresholds for the **LowNodeUtilization** strategy. Do not use this field with **devDeviationThresholds**.
- **devDeviationThresholds**: Treats nodes with below-average resource usage as underutilized to help redistribute workloads from overutilized nodes. Do not use this field with **devLowNodeUtilizationThresholds**. Supported values are: **Low** (10%:10%), **Medium** (20%:20%), **High** (30%:30%), **AsymmetricLow** (0%:10%), **AsymmetricMedium** (0%:20%), **AsymmetricHigh** (0%:30%).
- **devEnableSoftTainter**: Enables the soft-tainting component to dynamically apply or remove soft taints as scheduling hints.

Example configuration

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
  name: cluster
  namespace: openshift-kube-descheduler-operator
spec:
  managementState: Managed
  deschedulingIntervalSeconds: 30
  mode: "Automatic"
  profiles:
    - KubeVirtRelieveAndMigrate
  profileCustomizations:
    devEnableSoftTainter: true
    devDeviationThresholds: AsymmetricLow
    devActualUtilizationProfile: PrometheusCPUCombined
```

The **KubeVirtRelieveAndMigrate** profile requires PSI metrics to be enabled on all worker nodes. You can enable this by applying the following **MachineConfig** custom resource (CR):

Example **MachineConfig** CR

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 99-openshift-machineconfig-worker-psi-karg
spec:
  kernelArguments:
    - psi=1
```

NOTE

The name of the **MachineConfig** object is significant because machine configs are processed in lexicographical order. By default, a config that starts with **98-** disables PSI. To ensure that PSI is enabled, name your config with a higher prefix, such as **99-openshift-machineconfig-worker-psi-karg**.

You can use this profile with the **SoftTopologyAndDuplicates** profile to also rebalance pods based on soft topology constraints, which can be useful in hosted control plane environments.

LongLifecycle

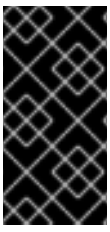
This profile balances resource usage between nodes and enables the following strategies:

- **RemovePodsHavingTooManyRestarts**: removes pods whose containers have been restarted too many times and pods where the sum of restarts over all containers (including Init Containers) is more than 100. Restarting the VM guest operating system does not increase this count.
- **LowNodeUtilization**: evicts pods from overutilized nodes when there are any underutilized nodes. The destination node for the evicted pod will be determined by the scheduler.
  - A node is considered underutilized if its usage is below 20% for all thresholds (CPU, memory, and number of pods).
  - A node is considered overutilized if its usage is above 50% for any of the thresholds (CPU, memory, and number of pods).

10.15.13.2. Installing the descheduler

The descheduler is not available by default. To enable the descheduler, you must install the Kube Descheduler Operator from the software catalog and enable one or more descheduler profiles.

By default, the descheduler runs in predictive mode, which means that it only simulates pod evictions. You must change the mode to automatic for the descheduler to perform the pod evictions.



IMPORTANT

If you have enabled hosted control planes in your cluster, set a custom priority threshold to lower the chance that pods in the hosted control plane namespaces are evicted. Set the priority threshold class name to **hypershift-control-plane**, because it has the lowest priority value (**100000000**) of the hosted control plane priority classes.

Prerequisites

- You are logged in to OpenShift Container Platform as a user with the **cluster-admin** role.
- Access to the OpenShift Container Platform web console.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Create the required namespace for the Kube Descheduler Operator.
   a. Navigate to **Administration → Namespaces** and click **Create Namespace**.

b. Enter **openshift-kube-descheduler-operator** in the **Name** field, enter **openshift.io/cluster-monitoring=true** in the **Labels** field to enable descheduler metrics, and click **Create**.

3. Install the Kube Descheduler Operator.

a. Navigate to **Ecosystem → Software Catalog**.

b. Type **Kube Descheduler Operator** into the filter box.

c. Select the **Kube Descheduler Operator** and click **Install**.

d. On the **Install Operator** page, select **A specific namespace on the cluster**. Select **openshift-kube-descheduler-operator** from the drop-down menu.

e. Adjust the values for the **Update Channel** and **Approval Strategy** to the desired values.

f. Click **Install**.

4. Create a descheduler instance.

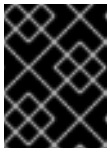
a. From the **Ecosystem → Installed Operators** page, click the **Kube Descheduler Operator**.

b. Select the **Kube Descheduler** tab and click **Create KubeDescheduler**.

c. Edit the settings as necessary.

i. To evict pods instead of simulating the evictions, change the **Mode** field to **Automatic**.

ii. Expand the **Profiles** section and select **LongLifecycle**. The **AffinityAndTaints** profile is enabled by default.



IMPORTANT

The only profile currently available for OpenShift Virtualization is **LongLifecycle**.

You can also configure the profiles and settings for the descheduler later using the OpenShift CLI (**oc**).

10.15.13.3. Configuring descheduler evictions for virtual machines

After the descheduler is installed and configured, all migratable virtual machines (VMs) are eligible for eviction by default. You can configure the descheduler to manage VM evictions across the cluster and optionally exclude specific VMs from eviction.

Prerequisites

- Install the descheduler in the OpenShift Container Platform web console or OpenShift CLI (**oc**).

Procedure

1. Stop the VM.

2. Configure the **KubeDescheduler** object with the **KubeVirtRelieveAndMigrate** profile and enable background evictions for improved VM eviction stability during live migration:

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
  name: cluster
  namespace: openshift-kube-descheduler-operator
spec:
  deschedulingIntervalSeconds: 60
  profiles:
  - KubeVirtRelieveAndMigrate
  mode: Automatic
```

3. Optional: To evict pods, set the **mode** field value to **Automatic**. By default, the descheduler does not evict pods.

4. Optional: Configure limits for the number of parallel evictions to improve stability in large clusters.
   The descheduler can limit the number of concurrent evictions per node and across the cluster by using the **evictionLimits** field. Set these limits to match the migration limits configured in the **HyperConverged** custom resource (CR).

   ```
   spec:
     evictionLimits:
       node: 2
       total: 5
   ```

   Set values that correspond to the migration limits in the **HyperConverged** CR:

   ```
   spec:
     liveMigrationConfig:
       parallelMigrationsPerCluster: 5
       parallelOutboundMigrationsPerNode: 2
   ```

5. Optional: To exclude the VM from eviction, add the **descheduler.alpha.kubernetes.io/prefer-no-eviction** annotation to the **spec.template.metadata.annotations** field. The change is applied dynamically and is propagated to the **VirtualMachineInstance** (VMI) object and the **virt-launcher** pod.
   Only the presence of the annotation is checked. The value is not evaluated, so **"true"** and **"false"** have the same effect.

   ```
   apiVersion: kubevirt.io/v1
   kind: VirtualMachine
   spec:
     template:
       metadata:
         annotations:
           descheduler.alpha.kubernetes.io/prefer-no-eviction: "true"
   ```

6. Start the VM.

The VM is now configured according to the descheduler settings.

10.15.13.4. Additional resources

- Descheduler overview

10.15.14. About high availability for virtual machines

You can enable high availability for virtual machines (VMs) by manually deleting a failed node to trigger VM failover or by configuring remediating nodes.

Manually deleting a failed node

If a node fails and machine health checks are not deployed on your cluster, virtual machines with **runStrategy: Always** configured are not automatically relocated to healthy nodes. To trigger VM failover, you must manually delete the **Node** object.

See Deleting a failed node to trigger virtual machine failover .

Configuring remediating nodes

You can configure remediating nodes by installing the Self Node Remediation Operator or the Fence Agents Remediation Operator from the software catalog and enabling machine health checks or node remediation checks.

For more information on remediation, fencing, and maintaining nodes, see the Workload Availability for Red Hat OpenShift documentation.

10.15.15. Virtual machine control plane tuning

OpenShift Virtualization offers the following tuning options at the control-plane level:

- The **highBurst** profile, which uses fixed **QPS** and **burst** rates, to create hundreds of virtual machines (VMs) in one batch
- Migration setting adjustment based on workload type

10.15.15.1. Configuring a highBurst profile

Use the **highBurst** profile to create and maintain a large number of virtual machines (VMs) in one cluster.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- Apply the following patch to enable the **highBurst** tuning policy profile:

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

Verification

- Run the following command to verify the **highBurst** tuning policy profile is enabled:

```
$ 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"}}
```

10.15.16. Assigning compute resources

In OpenShift Virtualization, compute resources assigned to virtual machines (VMs) are backed by either guaranteed CPUs or time-sliced CPU shares.

Guaranteed CPUs, also known as CPU reservation, dedicate CPU cores or threads to a specific workload, which makes them unavailable to any other workload. Assigning guaranteed CPUs to a VM ensures that the VM will have sole access to a reserved physical CPU. Enable dedicated resources for VMs to use a guaranteed CPU.

Time-sliced CPUs dedicate a slice of time on a shared physical CPU to each workload. You can specify the size of the slice during VM creation, or when the VM is offline. By default, each vCPU receives 100 milliseconds, or 1/10 of a second, of physical CPU time.

The type of CPU reservation depends on the instance type or VM configuration.

10.15.16.1. Overcommitting CPU resources

Time-slicing allows multiple virtual CPUs (vCPUs) to share a single physical CPU. This is known as *CPU overcommitment*. Guaranteed VMs can not be overcommitted.

Configure CPU overcommitment to prioritize VM density over performance when assigning CPUs to VMs. With a higher CPU over-commitment of vCPUs, more VMs fit onto a given node.

10.15.16.2. Setting the CPU allocation ratio

The CPU Allocation Ratio specifies the degree of overcommitment by mapping vCPUs to time slices of physical CPUs.

For example, a mapping or ratio of 10:1 maps 10 virtual CPUs to 1 physical CPU by using time slices.

To change the default number of vCPUs mapped to each physical CPU, set the **vmiCPUAllocationRatio** value in the **HyperConverged** CR. The pod CPU request is calculated by multiplying the number of vCPUs by the reciprocal of the CPU allocation ratio. For example, if **vmiCPUAllocationRatio** is set to 10, OpenShift Virtualization will request 10 times fewer CPUs on the pod for that VM.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

Set the **vmiCPUAllocationRatio** value in the **HyperConverged** CR to define a node CPU allocation ratio.

1. Open the **HyperConverged** CR in your default editor by running the following command:

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

2. Set the **vmiCPUAllocationRatio**:

```
...
spec:
  resourceRequirements:
    vmiCPUAllocationRatio: 1 (1)
# ...
```

(1) When **vmiCPUAllocationRatio** is set to **1**, the maximum amount of vCPUs are requested for the pod.

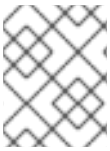
10.15.16.3. Additional resources

- Pod Quality of Service Classes

10.15.17. About multi-queue functionality

Use multi-queue functionality to scale network throughput and performance on virtual machines (VMs) with multiple vCPUs.

By default, the **queueCount** value, which is derived from the domain XML, is determined by the number of vCPUs allocated to a VM. Network performance does not scale as the number of vCPUs increases. Additionally, because virtio-net has only one Tx and Rx queue, guests cannot transmit or retrieve packs in parallel.



NOTE

Enabling virtio-net multiqueue does not offer significant improvements when the number of vNICs in a guest instance is proportional to the number of vCPUs.

10.15.17.1. Known limitations

- MSI vectors are still consumed if virtio-net multiqueue is enabled in the host but not enabled in the guest operating system by the administrator.
- Each virtio-net queue consumes 64 KiB of kernel memory for the vhost driver.
- Starting a VM with more than 16 CPUs results in no connectivity if **networkInterfaceMultiqueue** is set to 'true' (CNV-16107).

10.15.17.2. Enabling multi-queue functionality

Enable multi-queue functionality for interfaces configured with a VirtIO model.

Procedure

1. Set the **networkInterfaceMultiqueue** value to **true** in the **VirtualMachine** manifest file of your VM to enable multi-queue functionality:

```
apiVersion: kubevirt.io/v1
kind: VM
spec:
```

```
domain:
  devices:
    networkInterfaceMultiqueue: true
```

2. Save the **VirtualMachine** manifest file to apply your changes.

10.15.18. Managing virtual machines by using OpenShift GitOps

To automate and optimize virtual machine (VM) management in OpenShift Virtualization, you can use OpenShift GitOps.

With GitOps, you can set up VM deployments based on configuration files stored in a Git repository. This also makes it easier to automate, update, or replicate these configurations, as well to use version control for tracking their changes.

Prerequisites

- You have a GitHub account. For instructions to set up an account, see Creating an account on GitHub.
- OpenShift Virtualuzation has been installed on your OpenShift cluster. For instructions, see OpenShift Virtualization installation.
- The OpenShift GitOps operator has been installed on your OpenShift cluster. For instructions, see Installing GitOps.

Procedure

Follow the *Manage OpenShift virtual machines with GitOps* learning path in performing these steps:

1. Connect an external Git repository to your Argo CD instance.
2. Create the required VM configuration in the Git repository.
3. Use the VM configuration to create VMs on your cluster.

Additional resources

- OpenShift GitOps documentation

10.15.19. Working with NUMA topology for virtual machines

Non-uniform memory access (NUMA) architecture is a multiprocessor architecture model where CPUs do not access all memory in all locations at the same speed. Instead, CPUs can gain faster access to memory that is in closer proximity to them, or *local* to them, but slower access to memory that is further away.

A CPU with multiple memory controllers can use any available memory across CPU complexes, regardless of where the memory is located. However, this increased flexibility comes at the expense of performance.

NUMA resource topology refers to the physical locations of CPUs, memory, and PCI devices relative to each other in a *NUMA zone*. In a NUMA architecture, a NUMA zone is a group of CPUs that has its own processors and memory. Colocated resources are said to be in the same NUMA zone, and CPUs in a zone have faster access to the same local memory than CPUs outside of that zone. A CPU processing a

workload using memory that is outside its NUMA zone is slower than a workload processed in a single NUMA zone. For I/O-constrained workloads, the network interface on a distant NUMA zone slows down how quickly information can reach the application.

Applications can achieve better performance by containing data and processing within the same NUMA zone. For high-performance workloads and applications, such as telecommunications workloads, the cluster must process pod workloads in a single NUMA zone so that the workload can operate to specification.

10.15.19.1. Using NUMA topology with OpenShift Virtualization

You must enable the NUMA functionality for OpenShift Virtualization VMs to prevent performance degradation on nodes with multiple NUMA zones. This feature is vital for high-performance and latency-sensitive workloads.

Without NUMA awareness, a VM's virtual CPUs might run on one physical NUMA zone, while its memory is allocated on another. This "cross-node" communication significantly increases latency and reduces memory bandwidth, and can cause the interconnect buses which link the NUMA zones to become a bottleneck.

When you enable the NUMA functionality for OpenShift Virtualization VMs, you allow the host to pass its physical topology directly to the VM's guest operating system (OS). The guest OS can then make intelligent, NUMA-aware decisions about scheduling and memory allocation. This ensures that process threads and memory are kept on the same physical NUMA node. By aligning the virtual topology with the physical one, you minimize latency and maximize performance.

10.15.19.2. Prerequisites

Before you can enable NUMA functionality with OpenShift Virtualization VMs, you must ensure that your environment meets the following prerequisites.

- Worker nodes must have huge pages enabled.
- The **KubeletConfig** object on worker nodes must be configured with the **cpuManagerPolicy: static** spec to guarantee dedicated CPU allocation, which is a prerequisite for NUMA pinning.

  Example cpuManagerPolicy: static spec

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: cpu-numa-static-config
spec:
  kubeletConfig:
    cpuManagerPolicy: static
# ...
```

10.15.19.3. Creating a VM with NUMA functionality enabled

VM owners can enable NUMA with **ComputeExclusive** (CX) instance types, which are specifically designed for high-performance, compute-intensive workloads, and are configured to use NUMA features.

For information about creating VMs using a CX instance type, see Creating virtual machines from instance types.

10.15.19.4. Verifying vNUMA status of a VM

VM administrators might need to confirm whether non-uniform memory access (NUMA) is configured for a VM, to verify the VM's resource allocation setup for high-performance, latency-sensitive workloads that rely on memory locality.

You can verify whether an already deployed VM is configured for vNUMA by checking the **spec.domain.cpu.numa** attribute. This is displayed as a **vNUMA** badge in the OpenShift Container Platform web console.

Prerequisites

- You have access to an OpenShift Container Platform cluster with OpenShift Virtualization installed.
- If you want to use the command line for verification, you must have installed the OpenShift CLI (**oc**). Otherwise, you only need access to the OpenShift Container Platform web console.

Procedure

- To verify vNUMA status on the command line, check that the **spec.domain.cpu.numa** attribute is configured by using the OpenShift CLI (**oc**). Run the following command:

  ```
  $ oc get vm <vm_name> -n <namespace> -o
  jsonpath='{.spec.template.spec.domain.cpu.numa}'
  ```

  If any output other than an empty string is returned, vNUMA is enabled for the VM.

- To verify vNUMA status in a GUI, check if the VM has a **vNUMA** badge in the OpenShift Container Platform web console. Go to **VirtualMachines → VirtualMachine details**, and check either the **Overview** or the **Configuration** tabs.

10.15.19.5. Disabling the hot plug capability for VMs

Hot plugging is the ability to add resources like memory or CPU dynamically to a VM while it is running.

Default OpenShift Virtualization hot plug multipliers can cause VMs to request an excessive number of sockets. For example, if your VM requests 10 sockets, the default hot plug behavior multiplies this by 4, which means that the total request is 40 sockets. This can exceed the recommended CPUs supported by the Kernel-based Virtual Machine (KVM), which can cause deployment failures.

You can keep VM resource requests aligned with NUMA and optimize performance for resource-intensive workloads by disabling the VM's default hot plug capability.

10.15.19.5.1. Disabling the CPU hot plug by instance type

As a cluster administrator, you can disable the CPU hot plug by instance type. This is the recommended approach to standardize VM configurations and ensure NUMA-aware CPU allocation without hot plugs for specific instance types.

When a VM is created by using an instance type where the CPU hot plug is disabled, the VM inherits these settings and the CPU hot plug is disabled for that VM.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a YAML file for a **VirtualMachineClusterInstancetype** custom resource (CR). Add a **maxSockets** spec to the instance type that you want to configure:

   #### Example VirtualMachineClusterInstancetype CR

   ```
   apiVersion: instancetype.kubevirt.io/v1beta1
   kind: VirtualMachineClusterInstancetype
   metadata:
     name: cx1.mycustom-numa-instance
   spec:
     cpu:
       dedicatedCPUPlacement: true
       isolateEmulatorThread: true
       numa:
         guestMappingPassthrough: {}
       guest: 8
       maxSockets: 8
     memory:
       guest: 16Gi
       hugepages:
         pageSize: 1Gi
   ```

   where:

   spec.cpu.dedicatedCPUPlacement

   Specifies whether dedicated resources are allocated to the VM instance. If this is set to **true**, the VM's VCPUs are pinned to physical host CPUs. This is often used for high-performance workloads to minimize scheduling jitter.

   spec.cpu.isolateEmulatorThread

   Specifies whether the QEMU emulator thread should be isolated and run on a dedicated physical CPU core. This is a performance optimization that is typically used alongside the **dedicatedCPUPlacement** spec.

   spec.cpu.numa

   Specifies the NUMA topology configuration for the VM.

   spec.cpu.numa.guestMappingPassthrough

   Specifies that the VM's NUMA topology should directly pass through the NUMA topology of the underlying host machine. This is critical for applications that are NUMA-aware and require optimal performance.

   spec.cpu.guest

   Specifies the total number of vCPUs to be allocated to the VM.

   spec.cpu.maxSockets

   Specifies the maximum number of CPU sockets the VM is allowed to have.

   spec.memory

   Specifies the memory configuration for the VM.

   spec.memory.guest

   Specifies the total amount of memory to be allocated to the VM.

spec.memory.hugepages

Specifies configuration related to hugepages.

spec.memory.hugepages.pageSize

Specifies the size of the hugepages to be used for the VM's memory.

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

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

Verification

1. Create a VM that uses the updated **VirtualMachineClusterInstancetype** configuration.
2. Inspect the configuration of the created VM by running the following command and inspecting the output:

```
$ oc get vmi <vm_name> -o yaml
```

Example output

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  name: example-vmi
  labels:
    instancetype.kubevirt.io/cluster-instancetype: cx1.example-numa-instance
spec:
  domain:
    cpu:
      dedicatedCPUPlacement: true
      isolateEmulatorThread: true
      sockets: 8
      cores: 1
      threads: 1
      numa:
        guestMappingPassthrough: {}
      guest: 8
      maxSockets: 8
# ...
```

The update has applied successfully if in the **spec.template.spec.domain.cpu** section:

- The **sockets** value matches the **maxSockets** and **guest** values from the instance type, which ensures that no extra hot plug slots are configured.
- The **dedicatedCPUPlacement** and **isolateEmulatorThread** fields are present and set to **true**.

10.15.19.5.2. Adjusting or disabling the CPU hot plug by VM

As a VM owner, you can adjust or disable the CPU hot plug for individual VMs. This is the simplest solution for large, performance-critical VMs where you want to ensure a fixed CPU allocation from the start.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Modify the **VirtualMachine** custom resource (CR) for the VM that you want to configure to add a **maxSockets** and **sockets** spec:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: large-numa-vm
spec:
  template:
    spec:
      domain:
        cpu:
          maxSockets: 10
          sockets: 10
          cores: 1
          threads: 1
```

   By explicitly setting **maxSockets** and **sockets** to a value of 10 or higher, you are specifying that additional capacity is not reserved for hot plugging, which ensures that the entire requested cores are the actual cores allocated.

2. Apply the changes to the **VirtualMachine** CR by running the following command:

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

Verification

1. Check that you have configured the **maxSockets** and **sockets** values correctly, by running the following commands:

```
$ oc get vmi -o jsonpath='{.spec.domain.cpu.maxSockets}'
```

```
$ oc get vmi -o jsonpath='{.spec.domain.cpu.sockets}'
```

   If the configuration was successful, the outputs are the **maxSockets** and **sockets** values that you set in the previous procedure:

   Example output

```
10
```

10.15.19.5.3. Disabling hot plugging for all VMs on a cluster

If you are a cluster administrator and want to disable hot plugging for an entire cluster, you must modify the **spec.configuration.kubevirtConfiguration.developerConfiguration.maxHotplugRatio** setting in the **HyperConverged** custom resource (CR).

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have installed the OpenShift Virtualization Operator.

Procedure

1. Modify the **HyperConverged** CR and set the **maxHotplugRatio** value to **1.0**:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: kubevirt-hyperconverged
spec:
  # ...
  kubevirtConfiguration:
    developerConfiguration:
      maxHotplugRatio: 1.0
# ...
```

2. Apply the changes to the **HyperConverged** CR by running the following command:

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

Verification

1. Check that you have configured the **maxHotplugRatio** value correctly, by running the following command:

```
$ oc get hyperconverged -n openshift-cnv -o
jsonpath='{.spec.liveUpdateConfiguration.maxHotplugRatio}'
```

   If the configuration was successful, the output is the **maxHotplugRatio** value that you set in the previous procedure:

   Example output

```
1.0
```

10.15.19.6. Limitations of NUMA for OpenShift Virtualization

When you use NUMA topology with OpenShift Virtualization VMs, certain limitations can impact performance and VM management.

Asymmetrical topology

The host scheduler cannot guarantee assigning specific NUMA nodes to a VM. For example, if a VM is rescheduled to a different host machine because of a restart or maintenance, the new host might have a different physical NUMA layout. This means that the VM could be presented with an asymmetrical NUMA topology that reflects the new host's configuration, rather than its original or desired layout. This change can have a negative impact on the VM's performance.

Live migration challenges

Migrating a NUMA-enabled VM to a different host node can be challenging if the destination node's NUMA topology differs significantly from the source node's. A mismatch between the NUMA layouts of the source and destination can lead to a degradation of the VM's performance after the migration is complete.

No support for PCI NUMA nodes

There is no explicit support for passing GPU NUMA zone information to the VM. This means that the VM's guest operating system is not aware of the NUMA locality of PCI devices such as GPUs. For workloads that heavily rely on these devices, this lack of awareness could potentially lead to reduced performance if the GPU's memory is not local to the accessing CPU within the NUMA architecture.

10.15.19.7. Live migration outcomes using vNUMA

Migration outcomes for VMs are dependent on the configured Topology Manager policies. These policies determine how CPU and memory resources are allocated with respect to the physical NUMA nodes of the host. There are four available policies: **None**, **single-numa-node**, **best-effort**, and **restricted**.

The following table outlines which policies are supported for different VM configurations, and their effect on live migration.

- A small VM is defined as a VM with less total cores than half of cores in NUMA node.
- A large VM is defined as a VM with more total cores than half of cores in NUMA node.
- An extra large VM is defined as a VM with more cores than 1 NUMA node.

VM size	Topology Manager policy	Tested support status
Any	single-numa-node	The VM fails to start because the pod requests more cpus than a single NUMA node on the host can provide. This triggers a topology affinity error during scheduling, which is expected behavior given the node's hardware limits.
Any	None	Live migration does not work. This is a known issue. The process ends with an incorrect memnode allocation error, and libvirt rejects the XML manifest generated by KubeVirt. See release notes for additional information.
Small	None	Live migration works, as expected.
Small	single-numa-node	Live migration works, as expected.

VM size	Topology Manager policy	Tested support status
Small	best-effort	Live migration works, as expected.
Small	restricted	Live migration works, as expected.
Large	single-numa-node	Live migration works, as expected.
Large	best-effort	Live migration works, as expected.
Large	restricted	Live migration works, as expected.
Extra large	None	Live migration works, as expected.
Extra large	best-effort	Live migration works, as expected.
Extra large	restricted	VMs do not work, as expected.

10.15.19.8. Additional resources

- Topology Manager policies

10.16. VM DISKS

10.16.1. Hot-plugging VM disks

You can add or remove virtual disks without stopping your virtual machine (VM) or virtual machine instance (VMI).

Only data volumes and persistent volume claims (PVCs) can be hot plugged and hot-unplugged. You cannot hot plug or hot-unplug container disks.

A hot plugged disk remains attached to the VM even after reboot. You must unplug the disk to remove it from the VM.

NOTE

Each VM has a **virtio-scsi** controller so that hot plugged disks can use the SCSI bus. The **virtio-scsi** controller overcomes the limitations of VirtIO while retaining its performance advantages. It is highly scalable and supports hot plugging over 4 million disks.

When you hot plug disks to the VirtIO (**virtio-blk**) bus, each disk uses a PCI Express (PCIe) slot in the VM. The number of PCIe slots is limited and pre-set automatically at the VM creation as specified in the Available VirtIO Ports table. Therefore, you can use **virtio-blk** for a small number of disks that does not exceed the number of available slots.

10.16.1.1. Hot plugging and hot unplugging a disk by using the web console

You can hot plug a disk by attaching it to a virtual machine (VM) while the VM is running by using the OpenShift Container Platform web console.

The hot plugged disk remains attached to the VM until you unplug it.

Prerequisites

- You must have a data volume or persistent volume claim (PVC) available for hot plugging.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Select a running VM to view its details.
3. On the **VirtualMachine details** page, click **Configuration → Storage**.
4. Add a hot plugged disk:
   a. Click **Add**.
   b. In the **Add disk (hot plugged)** window, select the disk from the **Source** list and click **Save**.
5. Optional: Select the type of the interface bus. The options are **VirtIO** and **SCSI**. The default bus type is **VirtIO**.
6. Optional: Change the type of the interface bus of an existing hot plugged disk:
   a. Click the Options menu beside the disk and select the **Edit** option.
   b. In the **Interface** field, select the desired option.
7. Optional: Unplug a hot plugged disk:
   a. Click the Options menu beside the disk and select **Detach**.
   b. Click **Detach**.

10.16.1.2. Hot plugging and hot unplugging a disk by using the CLI

You can hot plug and hot unplug a disk while a virtual machine (VM) is running by using the command line.

The hot plugged disk remains attached to the VM until you unplug it.

Prerequisites

- You must have at least one data volume or persistent volume claim (PVC) available for hot plugging.

Procedure

- Hot plug a disk by running the following command:

```
$ virtctl addvolume <virtual-machine|virtual-machine-instance> \
  --volume-name=<datavolume|PVC> \
  [--bus <bus_type>] [--serial=<label_name>]
```

  - The optional **--bus** flag allows you to specify the bus type of the added disk. The options are **virtio** and **scsi**. The default bus type is **virtio**.
  - The optional **--serial** flag allows you to add an alphanumeric string label of your choice. This helps you to identify the hot plugged disk in a guest virtual machine. If you do not specify this option, the label defaults to the name of the hot plugged data volume or PVC.

- Hot unplug a disk by running the following command:

```
$ virtctl removevolume <virtual-machine|virtual-machine-instance> \
  --volume-name=<datavolume|PVC>
```

10.16.2. Expanding virtual machine disks

You can increase the size of a virtual machine (VM) disk by expanding the persistent volume claim (PVC) of the disk.

If your storage provider does not support volume expansion, you can expand the available virtual storage of a VM by adding blank data volumes.

You cannot reduce the size of a VM disk.

10.16.2.1. Increasing a VM disk size by expanding the PVC of the disk

You can increase the size of a virtual machine (VM) disk by expanding the persistent volume claim (PVC) of the disk. To specify the increased PVC volume, you can use the web console with the VM running. Alternatively, you can edit the PVC manifest in the CLI.

NOTE

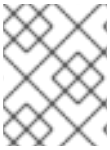
If the PVC uses the file system volume mode, the disk image file expands to the available size while reserving some space for file system overhead.

10.16.2.1.1. Expanding a VM disk PVC in the web console

You can increase the size of a VM disk PVC in the web console without leaving the **VirtualMachines** page and with the VM running.

Procedure

1. In the **Administrator** or **Virtualization** perspective, open the **VirtualMachines** page.
2. Select the running VM to open its **Details** page.
3. Select the **Configuration** tab and click **Storage**.
4. Click the options menu  next to the disk you want to expand. Select the **Edit** option. The **Edit disk** dialog opens.
5. In the **PersistentVolumeClaim size** field, enter the desired size.
6. Click **Save**.



NOTE

You can enter any value greater than the current one. However, if the new value exceeds the available size, an error is displayed.

10.16.2.1.2. Expanding a VM disk PVC by editing its manifest

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **PersistentVolumeClaim** manifest of the VM disk that you want to expand:

```
$ oc edit pvc <pvc_name>
```

2. Update the disk size:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: vm-disk-expand
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 3Gi (1)
# ...
```

(1) Specify the new disk size.

Additional resources for volume expansion

- Extending a basic volume in Windows
- Extending an existing file system partition without destroying data in Red Hat Enterprise Linux
- Extending a logical volume and its file system online in Red Hat Enterprise Linux

10.16.2.2. Expanding available virtual storage by adding blank data volumes

You can expand the available storage of a virtual machine (VM) by adding blank data volumes.

Prerequisites

- You must have at least one persistent volume.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **DataVolume** manifest as shown in the following example:

   Example DataVolume manifest

```
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. Specify the amount of available space requested for the data volume.
   2. Optional: If you do not specify a storage class, the default storage class is used.

2. Create the data volume by running the following command:

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

Additional resources for data volumes

- Configuring preallocation mode for data volumes
- Managing data volume annotations

10.16.3. Configuring shared volumes for virtual machines

You can configure shared disks to allow multiple virtual machines (VMs) to share the same underlying storage. A shared disk's volume must be block mode.

You configure disk sharing by exposing the storage as either of these types:

- An ordinary VM disk
- A logical unit number (LUN) disk with an SCSI connection and raw device mapping, as required for Windows Failover Clustering for shared volumes

In addition to configuring disk sharing, you can also set an error policy for each ordinary VM disk or LUN disk. The error policy controls how the hypervisor behaves when an input/output error occurs on a disk Read or Write.

10.16.3.1. Configuring disk sharing by using virtual machine disks

You can configure block volumes so that multiple virtual machines (VMs) can share storage.

The application running on the guest operating system determines the storage option you must configure for the VM. A disk of type **disk** exposes the volume as an ordinary disk to the VM.

You can set an error policy for each disk. The error policy controls how the hypervisor behaves when an input/output error occurs while a disk is being written to or read. The default behavior stops the VM and generates a Kubernetes event.

You can accept the default behavior, or you can set the error policy to one of the following options:

- **report**, which reports the error in the guest.
- **ignore**, which ignores the error. The Read or Write failure is undetected.
- **enospace**, which produces an error indicating that there is not enough disk space.

Prerequisites

- The volume access mode must be **ReadWriteMany** (RWX) if the VMs that are sharing disks are running on different nodes.
  If the VMs that are sharing disks are running on the same node, **ReadWriteOnce** (RWO) volume access mode is sufficient.
- The storage provider must support the required Container Storage Interface (CSI) driver.

Procedure

1. Create the **VirtualMachine** manifest for your VM to set the required values, as shown in the following example:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: <vm_name>
spec:
  template:
# ...
    spec:
      domain:
```

```
devices:
  disks:
  - disk:
      bus: virtio
    name: rootdisk
    errorPolicy: report 1
  - disk:
      bus: virtio
    name: cluster
    shareable: true 2
  interfaces:
  - masquerade: {}
    name: default
```

1. Identifies the error policy.
2. Identifies a shared disk.

2. Save the **VirtualMachine** manifest file to apply your changes.

10.16.3.2. Configuring disk sharing by using LUN

To secure data on your VM from outside access, you can enable SCSI persistent reservation and configure a LUN-backed virtual machine disk to be shared among multiple virtual machines. By enabling the shared option, you can use advanced SCSI commands, such as those required for a Windows failover clustering implementation, for managing the underlying storage.

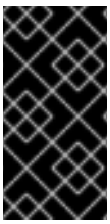
When a storage volume is configured as the **LUN** disk type, a VM can use the volume as a logical unit number (LUN) device. As a result, the VM can deploy and manage the disk by using SCSI commands.

You reserve a LUN through the SCSI persistent reserve options. To enable the reservation:

1. Configure the feature gate option
2. Activate the feature gate option on the LUN disk to issue SCSI device-specific input and output controls (IOCTLs) that the VM requires.

You can set an error policy for each LUN disk. The error policy controls how the hypervisor behaves when an input/output error occurs on a disk Read or Write. The default behavior stops the guest and generates a Kubernetes event.

For a LUN disk with an SCSi connection and a persistent reservation, as required for Windows Failover Clustering for shared volumes, you set the error policy to **report**.



IMPORTANT

OpenShift Virtualization does not currently support SCSI-3 Persistent Reservations (SCSI-3 PR) over multipath storage. As a workaround, disable multipath or ensure the Windows Server Failover Clustering (WSFC) shared disk is setup from a single device and not part of multipath.

Prerequisites

- You must have cluster administrator privileges to configure the feature gate option.

- The volume access mode must be **ReadWriteMany** (RWX) if the VMs that are sharing disks are running on different nodes.
  If the VMs that are sharing disks are running on the same node, **ReadWriteOnce** (RWO) volume access mode is sufficient.
- The storage provider must support a Container Storage Interface (CSI) driver that uses Fibre Channel (FC), Fibre Channel over Ethernet (FCoE), or iSCSI storage protocols.
- If you are a cluster administrator and intend to configure disk sharing by using LUN, you must enable the cluster's feature gate on the **HyperConverged** custom resource (CR).
- Disks that you want to share must be in block mode.

Procedure

1. Edit or create the **VirtualMachine** manifest for your VM to set the required values, as shown in the following example:

```
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) Identifies the error policy.

(2) Identifies a LUN disk.

(3) Identifies that the persistent reservation is enabled.

2. Save the **VirtualMachine** manifest file to apply your changes.

10.16.3.2.1. Configuring disk sharing by using LUN and the web console

You can use the OpenShift Container Platform web console to configure disk sharing by using LUN.

Prerequisites

- The cluster administrator must enable the **persistentreservation** feature gate setting.

Procedure

1. Click **Virtualization → VirtualMachines** in the web console.
2. Select a VM to open the **VirtualMachine details** page.
3. Expand **Storage**.
4. On the **Disks** tab, click **Add disk**.
5. Specify the **Name**, **Source**, **Size**, **Interface**, and **Storage Class**.
6. Select **LUN** as the **Type**.
7. Select **Shared access (RWX)** as the **Access Mode**.
8. Select **Block** as the **Volume Mode**.
9. Expand **Advanced Settings**, and select both checkboxes.
10. Click **Save**.

10.16.3.2.2. Configuring disk sharing by using LUN and the CLI

You can use the command line to configure disk sharing by using LUN.

Procedure

1. Edit or create the **VirtualMachine** manifest for your VM to set the required values, as shown in the following example:

```
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. Identifies a LUN disk.
2. Identifies that the persistent reservation is enabled.

2. Save the **VirtualMachine** manifest file to apply your changes.

10.16.3.3. Enabling the PersistentReservation feature gate

You can enable the SCSI **persistentReservation** feature gate and allow a LUN-backed block mode virtual machine (VM) disk to be shared among multiple virtual machines.

The **persistentReservation** feature gate is disabled by default. You can enable the **persistentReservation** feature gate by using the web console or the command line.

Prerequisites

- Cluster administrator privileges are required.
- The volume access mode **ReadWriteMany** (RWX) is required if the VMs that are sharing disks are running on different nodes. If the VMs that are sharing disks are running on the same node, the **ReadWriteOnce** (RWO) volume access mode is sufficient.
- The storage provider must support a Container Storage Interface (CSI) driver that uses Fibre Channel (FC), Fibre Channel over Ethernet (FCoE), or iSCSI storage protocols.

10.16.3.3.1. Enabling the PersistentReservation feature gate by using the web console

You must enable the PersistentReservation feature gate to allow a LUN-backed block mode virtual machine (VM) disk to be shared among multiple virtual machines. Enabling the feature gate requires cluster administrator privileges.

Procedure

1. Click **Virtualization → Overview** in the web console.
2. Click the **Settings** tab.
3. Select **Cluster**.
4. Expand **SCSI persistent reservation** and set **Enable persistent reservation** to on.

10.16.3.3.2. Enabling the PersistentReservation feature gate by using the CLI

You enable the **persistentReservation** feature gate by using the command line. Enabling the feature gate requires cluster administrator privileges.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Enable the **persistentReservation** feature gate by running the following command:

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

Additional resources

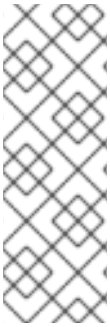
- Persistent reservation helper protocol
- Failover Clustering in Windows Server and Azure Stack HCI

10.16.4. Migrating VM disks to a different storage class

You can migrate one or more virtual disks to a different storage class without stopping your virtual machine (VM) or virtual machine instance (VMI).

10.16.4.1. Migrating VM disks to a different storage class by using the web console

You can migrate one or more disks attached to a virtual machine (VM) to a different storage class by using the OpenShift Container Platform web console. When performing this action on a running VM, the operation of the VM is not interrupted and the data on the migrated disks remains accessible.



NOTE

With the OpenShift Virtualization Operator, you can only start storage class migration for one VM at the time and the VM must be running. If you need to migrate more VMs at once or migrate a mix of running and stopped VMs, consider using the Migration Toolkit for Containers (MTC).

Migration Toolkit for Containers is not part of OpenShift Virtualization and requires separate installation.

Prerequisites

- You must have a data volume or a persistent volume claim (PVC) available for storage class migration.
- The cluster must have a node available for live migration. As part of the storage class migration, the VM is live migrated to a different node.
- The VM must be running.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.

2. Click the Options menu  beside the virtual machine and select **Migration → Storage**. You can also access this option from the **VirtualMachine details** page by selecting **Actions → Migration → Storage**.

   Alternatively, right-click the VM in the tree view and select **Migration** from the pop-up menu.

3. On the **Migration details** page, choose whether to migrate the entire VM storage or selected volumes only. If you click **Selected volumes**, select any disks that you intend to migrate. Click **Next** to proceed.

4. From the list of available options on the **Destination StorageClass** page, select the storage class to migrate to. Click **Next** to proceed.

5. On the **Review** page, review the list of affected disks and the target storage class. To start the migration, click **Migrate VirtualMachine storage**.

6. Stay on the **Migrate VirtualMachine storage** page to watch the progress and wait for the confirmation that the migration completed successfully.

Verification

1. From the **VirtualMachine details** page, navigate to **Configuration → Storage**.

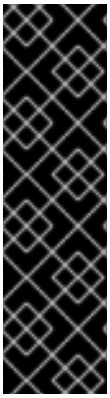
2. Verify that all disks have the expected storage class listed in the **Storage class** column.

CHAPTER 11. NETWORKING

11.1. NETWORKING OVERVIEW

OpenShift Virtualization provides advanced networking functionality by using custom resources and plugins. Virtual machines (VMs) are integrated with OpenShift Container Platform networking and its ecosystem.

OpenShift Virtualization support for single-stack IPv6 clusters is limited to the OVN-Kubernetes localnet and Linux bridge Container Network Interface (CNI) plugins.



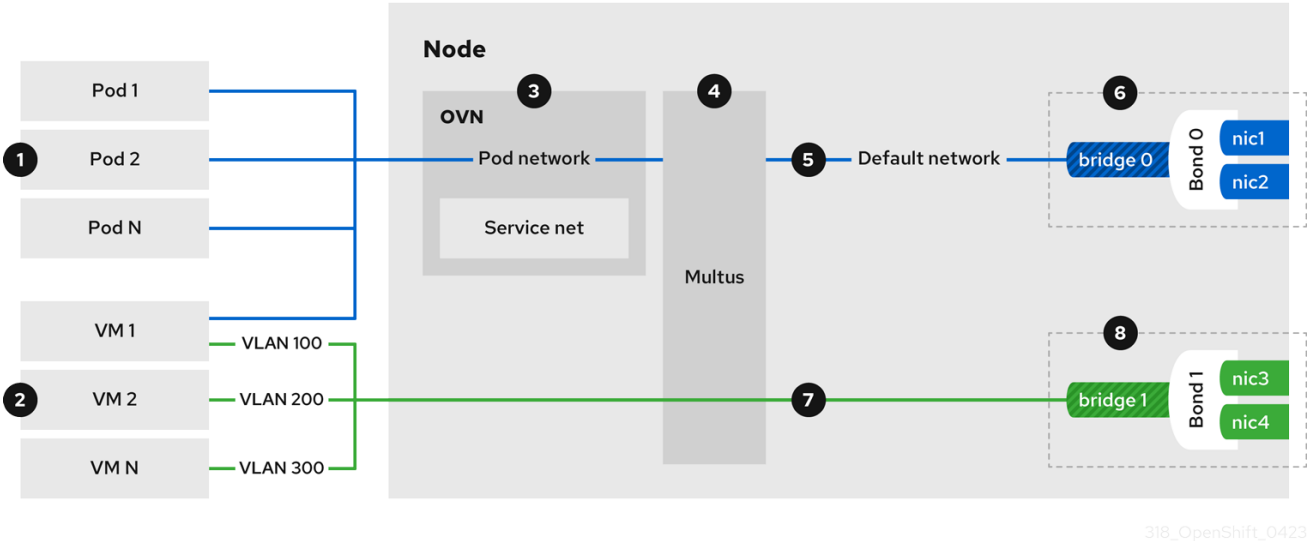
IMPORTANT

Deploying OpenShift Virtualization on a single-stack IPv6 cluster is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

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

The following figure illustrates the typical network setup of OpenShift Virtualization. Other configurations are also possible.

Figure 11.1. OpenShift Virtualization networking overview



1 Pods and VMs run on the same network infrastructure which allows you to easily connect your containerized and virtualized workloads.

2 You can connect VMs to the default pod network and to any number of secondary networks.

3 The default pod network provides connectivity between all its members, service abstraction, IP management, micro segmentation, and other functionality.

4 Multus is a "meta" CNI plugin that enables a pod or virtual machine to connect to additional network interfaces by using other compatible CNI plugins.

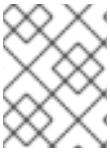
5 The default pod network is overlay-based, tunneled through the underlying machine network.

6 The machine network can be defined over a selected set of network interface controllers (NICs).

7 Secondary VM networks are typically bridged directly to a physical network, with or without VLAN encapsulation. It is also possible to create virtual overlay networks for secondary networks.

IMPORTANT

Connecting VMs directly to the underlay network is not supported on Red Hat OpenShift Service on AWS, Azure for OpenShift Container Platform, Google Cloud, or Oracle® Cloud Infrastructure (OCI).



NOTE

Connecting VMs to user-defined networks with the **layer2** topology is recommended on public clouds.

8 Secondary VM networks can be defined on dedicated set of NICs, as shown in Figure 1, or they can use the machine network.

11.1.1. OpenShift Virtualization networking glossary

The following terms are used throughout OpenShift Virtualization documentation:

Container Network Interface (CNI)

A Cloud Native Computing Foundation project, focused on container network connectivity. OpenShift Virtualization uses CNI plugins to build upon the basic Kubernetes networking functionality.

Multus

A "meta" CNI plugin that allows multiple CNIs to exist so that a pod or virtual machine can use the interfaces it needs.

Custom resource definition (CRD)

A Kubernetes API resource that allows you to define custom resources, or an object defined by using the CRD API resource.

Network attachment definition (NAD)

A CRD introduced by the Multus project that allows you to attach pods, virtual machines, and virtual machine instances to one or more networks.

UserDefinedNetwork (UDN)

A namespace-scoped CRD introduced by the user-defined network API that can be used to create a tenant network that isolates the tenant namespace from other namespaces.

ClusterUserDefinedNetwork (CUDN)

A cluster-scoped CRD introduced by the user-defined network API that cluster administrators can use to create a shared network across multiple namespaces.

Node network configuration policy (NNCP)

A CRD introduced by the nmstate project, describing the requested network configuration on nodes. You update the node network configuration, including adding and removing interfaces, by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

11.1.2. Using the default pod network

Connecting a virtual machine to the default pod network

Each VM is connected by default to the default internal pod network. You can add or remove network interfaces by editing the VM specification.

Exposing a virtual machine as a service

You can expose a VM within the cluster or outside the cluster by creating a **Service** object. For on-premise clusters, you can configure a load balancing service by using the MetalLB Operator. You can install the MetalLB Operator by using the OpenShift Container Platform web console or the CLI.

11.1.3. Configuring a primary user-defined network

Connecting a virtual machine to a primary user-defined network

You can connect a virtual machine (VM) to a user-defined network (UDN) on the primary interface of the VM. The primary UDN replaces the default pod network to connect pods and VMs in selected namespaces.
Cluster administrators can configure a primary **UserDefinedNetwork** CRD to create a tenant network that isolates the tenant namespace from other namespaces without requiring network policies. Additionally, cluster administrators can use the **ClusterUserDefinedNetwork** CRD to create a shared OVN **layer2** network across multiple namespaces.

User-defined networks with the **layer2** overlay topology are useful for VM workloads, and a good alternative to secondary networks in environments where physical network access is limited, such as the public cloud. The **layer2** topology enables seamless migration of VMs without the need for Network Address Translation (NAT), and also provides persistent IP addresses that are preserved between reboots and during live migration.

11.1.4. Configuring VM secondary network interfaces

You can connect a virtual machine to a secondary network by using Linux bridge, SR-IOV and OVN-Kubernetes CNI plugins. You can list multiple secondary networks and interfaces in the VM specification. It is not required to specify the primary pod network in the VM specification when connecting to a secondary network interface.

Connecting a virtual machine to an OVN-Kubernetes secondary network

You can connect a VM to an OVN-Kubernetes secondary network. OpenShift Virtualization supports the **layer2** and **localnet** topologies for OVN-Kubernetes. The **localnet** topology is the recommended way of exposing VMs to the underlying physical network, with or without VLAN encapsulation.

- A **layer2** topology connects workloads by a cluster-wide logical switch. The OVN-Kubernetes CNI plugin uses the Geneve (Generic Network Virtualization Encapsulation) protocol to create an overlay network between nodes. You can use this overlay network to connect VMs on different nodes, without having to configure any additional physical networking infrastructure.

- A **localnet** topology connects the secondary network to the physical underlay. This enables

both east-west cluster traffic and access to services running outside the cluster, but it requires additional configuration of the underlying Open vSwitch (OVS) system on cluster nodes.

To configure an OVN-Kubernetes secondary network and attach a VM to that network, perform the following steps:

1. Choose the appropriate option based on your OVN-Kubernetes network topology:
   - Configure an OVN-Kubernetes layer 2 secondary network by creating a network attachment definition (NAD).
   - Configure an OVN-Kubernetes localnet secondary network by creating a **ClusterUserDefinedNetwork** (CUDN) CR.
2. Choose the appropriate option based on your OVN-Kubernetes network topology:
   - Connect the VM to the OVN-Kubernetes layer 2 secondary network by adding the network details to the VM specification.
   - Connect the VM to the OVN-Kubernetes localnet secondary network by adding the network details to the VM specification.

Connecting a virtual machine to an SR-IOV network

You can use Single Root I/O Virtualization (SR-IOV) network devices with additional networks on your OpenShift Container Platform cluster installed on bare metal or Red Hat OpenStack Platform (RHOSP) infrastructure for applications that require high bandwidth or low latency.
You must install the SR-IOV Network Operator on your cluster to manage SR-IOV network devices and network attachments.

You can connect a VM to an SR-IOV network by performing the following steps:

1. Configure an SR-IOV network device by creating a **SriovNetworkNodePolicy** CRD.
2. Configure an SR-IOV network by creating an **SriovNetwork** object.
3. Connect the VM to the SR-IOV network by including the network details in the VM configuration.

Connecting a virtual machine to a Linux bridge network

Install the Kubernetes NMState Operator to configure Linux bridges, VLANs, and bonding for your secondary networks. The OVN-Kubernetes **localnet** topology is the recommended way of connecting a VM to the underlying physical network, but OpenShift Virtualization also supports Linux bridge networks.

NOTE

You cannot directly attach to the default machine network when using Linux bridge networks.

You can create a Linux bridge network and attach a VM to the network by performing the following steps:

1. Configure a Linux bridge network device by creating a **NodeNetworkConfigurationPolicy** custom resource definition (CRD).
2. Configure a Linux bridge network by creating a **NetworkAttachmentDefinition** CRD.
3. Connect the VM to the Linux bridge network by including the network details in the VM configuration.

Hot plugging secondary network interfaces

You can add or remove secondary network interfaces without stopping your VM. OpenShift Virtualization supports hot plugging and hot unplugging for secondary interfaces that use bridge binding and the VirtIO device driver. OpenShift Virtualization also supports hot plugging secondary interfaces that use the SR-IOV binding.

Using DPDK with SR-IOV

The Data Plane Development Kit (DPDK) provides a set of libraries and drivers for fast packet processing. You can configure clusters and VMs to run DPDK workloads over SR-IOV networks.

Configuring a dedicated network for live migration

You can configure a dedicated Multus network for live migration. A dedicated network minimizes the effects of network saturation on tenant workloads during live migration.

Accessing a virtual machine by using the cluster FQDN

You can access a VM that is attached to a secondary network interface from outside the cluster by using its fully qualified domain name (FQDN).

Configuring and viewing IP addresses

You can configure an IP address of a secondary network interface when you create a VM. The IP address is provisioned with cloud-init. You can view the IP address of a VM by using the OpenShift Container Platform web console or the command line. The network information is collected by the QEMU guest agent.

11.1.4.1. Comparing Linux bridge CNI and OVN-Kubernetes localnet topology

The following table provides a comparison of features available when using the Linux bridge CNI compared to the **localnet** topology for an OVN-Kubernetes plugin:

Table 11.1. Linux bridge CNI compared to an OVN-Kubernetes localnet topology

Feature	Available on Linux bridge CNI	Available on OVN-Kubernetes localnet
Layer 2 access to the underlay native network	Only on secondary network interface controllers (NICs)	Yes
Layer 2 access to underlay VLANs	Yes	Yes
Layer 2 trunk access	Yes	No
Network policies	No	Yes
MAC spoof filtering	Yes	Yes (Always on)

11.1.5. Integrating with OpenShift Service Mesh

Connecting a virtual machine to a service mesh

OpenShift Virtualization is integrated with OpenShift Service Mesh. You can monitor, visualize, and control traffic between pods and virtual machines.

11.1.6. Managing MAC address pools

Managing MAC address pools for network interfaces

The KubeMacPool component allocates MAC addresses for VM network interfaces from a shared MAC address pool. This ensures that each network interface is assigned a unique MAC address. A virtual machine instance created from that VM retains the assigned MAC address across reboots.

11.1.7. Configuring SSH access

Configuring SSH access to virtual machines

You can configure SSH access to VMs by using the following methods:

- **virtctl ssh** command
  You create an SSH key pair, add the public key to a VM, and connect to the VM by running the **virtctl ssh** command with the private key.

  You can add public SSH keys to Red Hat Enterprise Linux (RHEL) 9 VMs at runtime or at first boot to VMs with guest operating systems that can be configured by using a cloud-init data source.

- **virtctl port-forward** command
  You add the **virtctl port-foward** command to your **.ssh/config** file and connect to the VM by using OpenSSH.

- Service
  You create a service, associate the service with the VM, and connect to the IP address and port exposed by the service.

- Secondary network
  You configure a secondary network, attach a VM to the secondary network interface, and connect to its allocated IP address.

11.2. CONNECTING A VIRTUAL MACHINE TO THE DEFAULT POD NETWORK

You can connect a virtual machine to the default internal pod network by configuring its network interface to use the **masquerade** binding mode.

NOTE

Traffic passing through network interfaces to the default pod network is interrupted during live migration.

11.2.1. Configuring masquerade mode from the CLI

You can use masquerade mode to hide a virtual machine's outgoing traffic behind the pod IP address. Masquerade mode uses Network Address Translation (NAT) to connect virtual machines to the pod network backend through a Linux bridge.

Enable masquerade mode and allow traffic to enter the virtual machine by editing your virtual machine configuration file.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- The virtual machine must be configured to use DHCP to acquire IPv4 addresses.

Procedure

1. Edit the **interfaces** spec of your virtual machine configuration file:

```
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. Connect using masquerade mode.
2. Optional: List the ports that you want to expose from the virtual machine, each specified by the **port** field. The **port** value must be a number between 0 and 65536. When the **ports** array is not used, all ports in the valid range are open to incoming traffic. In this example, incoming traffic is allowed on port **80**.

NOTE

Ports 49152 and 49153 are reserved for use by the libvirt platform and all other incoming traffic to these ports is dropped.

2. Create the virtual machine:

```
$ oc create -f <vm-name>.yaml
```

11.2.2. Configuring masquerade mode with dual-stack (IPv4 and IPv6)

You can configure a new virtual machine (VM) to use both IPv6 and IPv4 on the default pod network by using cloud-init.

The **Network.pod.vmIPv6NetworkCIDR** field in the virtual machine instance configuration determines the static IPv6 address of the VM and the gateway IP address. These are used by the virt-launcher pod to route IPv6 traffic to the virtual machine and are not used externally. The **Network.pod.vmIPv6NetworkCIDR** field specifies an IPv6 address block in Classless Inter-Domain Routing (CIDR) notation. The default value is **fd10:0:2::2/120**. You can edit this value based on your network requirements.

When the virtual machine is running, incoming and outgoing traffic for the virtual machine is routed to both the IPv4 address and the unique IPv6 address of the virt-launcher pod. The virt-launcher pod then routes the IPv4 traffic to the DHCP address of the virtual machine, and the IPv6 traffic to the statically set IPv6 address of the virtual machine.

Prerequisites

- The OpenShift Container Platform cluster must use the OVN-Kubernetes Container Network Interface (CNI) network plugin configured for dual-stack.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. In a new virtual machine configuration, include an interface with **masquerade** and configure the IPv6 address and default gateway by using cloud-init.

```
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. Connect using masquerade mode.
2. Allows incoming traffic on port 80 to the virtual machine.
3. The static IPv6 address as determined by the **Network.pod.vmIPv6NetworkCIDR** field in the virtual machine instance configuration. The default value is **fd10:0:2::2/120**.
4. The gateway IP address as determined by the **Network.pod.vmIPv6NetworkCIDR** field in the virtual machine instance configuration. The default value is **fd10:0:2::1**.

2. Create the virtual machine in the namespace:

```
$ oc create -f example-vm-ipv6.yaml
```

Verification

- To verify that IPv6 has been configured, start the virtual machine and view the interface status of the virtual machine instance to ensure it has an IPv6 address:

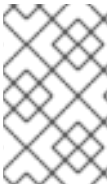
```
$ oc get vmi <vmi-name> -o jsonpath="{.status.interfaces[*].ipAddresses}"
```

11.2.3. About jumbo frames support

When using the OVN-Kubernetes CNI plugin, you can send unfragmented jumbo frame packets between two virtual machines (VMs) that are connected on the default pod network. Jumbo frames have a maximum transmission unit (MTU) value greater than 1500 bytes.

The VM automatically gets the MTU value of the cluster network, set by the cluster administrator, in one of the following ways:

- **libvirt**: If the guest OS has the latest version of the VirtIO driver that can interpret incoming data via a Peripheral Component Interconnect (PCI) config register in the emulated device.
- DHCP: If the guest DHCP client can read the MTU value from the DHCP server response.



NOTE

For Windows VMs that do not have a VirtIO driver, you must set the MTU manually by using **netsh** or a similar tool. This is because the Windows DHCP client does not read the MTU value.

11.2.4. Additional resources

- Changing the MTU for the cluster network
- Optimizing the MTU for your network

11.3. CONNECTING A VIRTUAL MACHINE TO A PRIMARY USER-DEFINED NETWORK

You can connect a virtual machine (VM) to a user-defined network (UDN) on the VM's primary interface by using the OpenShift Container Platform web console or the CLI. The primary user-defined network replaces the default pod network in your specified namespace. Unlike the pod network, you can define the primary UDN per project, where each project can use its specific subnet and topology.

OpenShift Virtualization supports the namespace-scoped **UserDefinedNetwork** and the cluster-scoped **ClusterUserDefinedNetwork** custom resource definitions (CRD).

Cluster administrators can configure a primary **UserDefinedNetwork** CRD to create a tenant network that isolates the tenant namespace from other namespaces without requiring network policies. Additionally, cluster administrators can use the **ClusterUserDefinedNetwork** CRD to create a shared OVN network across multiple namespaces.

NOTE

You must add the **k8s.ovn.org/primary-user-defined-network** label when you create a namespace that is to be used with user-defined networks.

With the layer 2 topology, OVN-Kubernetes creates an overlay network between nodes. You can use this overlay network to connect VMs on different nodes without having to configure any additional physical networking infrastructure.

The layer 2 topology enables seamless migration of VMs without the need for Network Address Translation (NAT) because persistent IP addresses are preserved across cluster nodes during live migration.

You must consider the following limitations before implementing a primary UDN:

- You cannot use the **virtctl ssh** command to configure SSH access to a VM.
- You cannot use the **oc port-forward** command to forward ports to a VM.
- You cannot use headless services to access a VM.

11.3.1. Creating a primary user-defined network by using the web console

You can use the OpenShift Container Platform web console to create a primary namespace-scoped **UserDefinedNetwork** or a cluster-scoped **ClusterUserDefinedNetwork** CRD. The UDN serves as the default primary network for pods and VMs that you create in namespaces associated with the network.

11.3.1.1. Creating a namespace for user-defined networks by using the web console

You can create a namespace to be used with primary user-defined networks (UDNs) by using the OpenShift Container Platform web console.

Prerequisites

- Log in to the OpenShift Container Platform web console as a user with **cluster-admin** permissions.

Procedure

1. From the **Administrator** perspective, click **Administration → Namespaces**.
2. Click **Create Namespace**.
3. In the **Name** field, specify a name for the namespace. The name must consist of lower case alphanumeric characters or '-', and must start and end with an alphanumeric character.
4. In the **Labels** field, add the **k8s.ovn.org/primary-user-defined-network** label.
5. Optional: If the namespace is to be used with an existing cluster-scoped UDN, add the appropriate labels as defined in the **spec.namespaceSelector** field in the **ClusterUserDefinedNetwork** custom resource.
6. Optional: Specify a default network policy.
7. Click **Create** to create the namespace.

11.3.1.2. Creating a primary namespace-scoped user-defined network by using the web console

You can create an isolated primary network in your project namespace by creating a **UserDefinedNetwork** custom resource in the OpenShift Container Platform web console.

Prerequisites

- You have access to the OpenShift Container Platform web console as a user with **cluster-admin** permissions.
- You have created a namespace and applied the **k8s.ovn.org/primary-user-defined-network** label. For more information, see "Creating a namespace for user-defined networks by using the web console".

Procedure

1. From the **Administrator** perspective, click **Networking → UserDefinedNetworks**.
2. Click **Create UserDefinedNetwork**.
3. From the **Project name** list, select the namespace that you previously created.
4. Specify a value in the **Subnet** field.
5. Click **Create**. The user-defined network serves as the default primary network for pods and virtual machines that you create in this namespace.

11.3.1.3. Creating a primary cluster-scoped user-defined network by using the web console

You can connect multiple namespaces to the same primary user-defined network (UDN) by creating a **ClusterUserDefinedNetwork** custom resource in the OpenShift Container Platform web console.

Prerequisites

- You have access to the OpenShift Container Platform web console as a user with **cluster-admin** permissions.

Procedure

1. From the **Administrator** perspective, click **Networking → UserDefinedNetworks**.
2. From the **Create** list, select **ClusterUserDefinedNetwork**.
3. In the **Name** field, specify a name for the cluster-scoped UDN.
4. Specify a value in the **Subnet** field.
5. In the **Project(s) Match Labels** field, add the appropriate labels to select namespaces that the cluster UDN applies to.
6. Click **Create**. The cluster-scoped UDN serves as the default primary network for pods and virtual machines located in namespaces that contain the labels that you specified in step 5.

Next steps

- Create namespaces that are associated with the cluster-scoped UDN

11.3.2. Creating a primary user-defined network by using the CLI

You can create a primary **UserDefinedNetwork** or **ClusterUserDefinedNetwork** CRD by using the CLI.

11.3.2.1. Creating a namespace for user-defined networks by using the CLI

You can create a namespace to be used with primary user-defined networks (UDNs) by using the OpenShift CLI (**oc**).

Prerequisites

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

Procedure

1. Create a **Namespace** object as a YAML file similar to the following example:

```
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace
  labels:
    k8s.ovn.org/primary-user-defined-network: "" 1
# ...
```

1 This label is required for the namespace to be associated with a UDN. If the namespace is to be used with an existing cluster UDN, you must also add the appropriate labels that are defined in the **spec.namespaceSelector** field of the **ClusterUserDefinedNetwork** custom resource.

2. Apply the **Namespace** manifest by running the following command:

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

11.3.2.2. Creating a primary namespace-scoped user-defined network by using the CLI

You can create an isolated primary network in your project namespace by using the CLI. You must use the OVN-Kubernetes layer 2 topology and enable persistent IP address allocation in the user-defined network (UDN) configuration to ensure VM live migration support.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have created a namespace and applied the **k8s.ovn.org/primary-user-defined-network** label.

Procedure

1. Create a **UserDefinedNetwork** object to specify the custom network configuration:

   Example UserDefinedNetwork manifest

```
apiVersion: k8s.ovn.org/v1
kind: UserDefinedNetwork
metadata:
  name: udn-l2-net 1
  namespace: my-namespace 2
spec:
  topology: Layer2 3
  layer2:
    role: Primary 4
    subnets:
      - "10.0.0.0/24"
      - "2001:db8::/60"
  ipam:
    lifecycle: Persistent 5
```

1. Specifies the name of the **UserDefinedNetwork** custom resource.
2. Specifies the namespace in which the VM is located. The namespace must have the **k8s.ovn.org/primary-user-defined-network** label. The namespace must not be **default**, an **openshift-*** namespace, or match any global namespaces that are defined by the Cluster Network Operator (CNO).
3. Specifies the topological configuration of the network. The required value is **Layer2**. A **Layer2** topology creates a logical switch that is shared by all nodes.
4. Specifies whether the UDN is primary or secondary. The **Primary** role means that the UDN acts as the primary network for the VM and all default traffic passes through this network.
5. Specifies that virtual workloads have consistent IP addresses across reboots and migration. The **spec.layer2.subnets** field is required when **ipam.lifecycle: Persistent** is specified.

2. Apply the **UserDefinedNetwork** manifest by running the following command:

```
$ oc apply -f --validate=true <filename>.yaml
```

11.3.2.3. Creating a primary cluster-scoped user-defined network by using the CLI

You can connect multiple namespaces to the same primary user-defined network (UDN) to achieve native tenant isolation by using the CLI.

Prerequisites

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

Procedure

1. Create a **ClusterUserDefinedNetwork** object to specify the custom network configuration:

Example ClusterUserDefinedNetwork manifest

```
kind: ClusterUserDefinedNetwork
metadata:
  name: cudn-l2-net 1
spec:
  namespaceSelector: 2
    matchExpressions: 3
    - key: kubernetes.io/metadata.name
      operator: In 4
      values: ["red-namespace", "blue-namespace"]
  network:
    topology: Layer2 5
    layer2:
      role: Primary 6
      ipam:
        lifecycle: Persistent
      subnets:
        - 203.203.0.0/16
```

1. Specifies the name of the **ClusterUserDefinedNetwork** custom resource.
2. Specifies the set of namespaces that the cluster UDN applies to. The namespace selector must not point to **default**, an **openshift-*** namespace, or any global namespaces that are defined by the Cluster Network Operator (CNO).
3. Specifies the type of selector. In this example, the **matchExpressions** selector selects objects that have the label **kubernetes.io/metadata.name** with the value **red-namespace** or **blue-namespace**.
4. Specifies the type of operator. Possible values are **In**, **NotIn**, and **Exists**.
5. Specifies the topological configuration of the network. The required value is **Layer2**. A **Layer2** topology creates a logical switch that is shared by all nodes.

6 Specifies whether the UDN is primary or secondary. The **Primary** role means that the UDN acts as the primary network for the VM and all default traffic passes through this network.

2. Apply the **ClusterUserDefinedNetwork** manifest by running the following command:

```
$ oc apply -f --validate=true <filename>.yaml
```

Next steps

- Create namespaces that are associated with the cluster-scoped UDN

11.3.3. Attaching a virtual machine to the primary user-defined network

You can connect a virtual machine (VM) to the primary user-defined network (UDN) by requesting the pod network attachment and configuring the interface binding.

OpenShift Virtualization supports the following network binding plugins to connect the network interface to the VM:

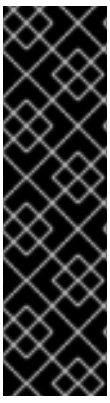
Layer 2 bridge

The Layer 2 bridge binding creates a direct Layer 2 connection between the VM's virtual interface and the virtual switch of the UDN.

Passt

The Plug a Simple Socket Transport (passt) binding provides a user-space networking solution that integrates seamlessly with the pod network, providing better integration with the OpenShift Container Platform networking ecosystem.
Passt binding has the following benefits:

- You can define readiness and liveness HTTP probes to configure VM health checks.
- You can use Red Hat Advanced Cluster Security to monitor TCP traffic within the cluster with detailed insights.



IMPORTANT

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

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

11.3.3.1. Attaching a virtual machine to the primary user-defined network by using the web console

You can connect a virtual machine (VM) to the primary user-defined network (UDN) by using the OpenShift Container Platform web console. VMs that are created in a namespace where the primary UDN is configured are automatically attached to the UDN with the Layer 2 bridge network binding plugin.

To attach a VM to the primary UDN by using the Plug a Simple Socket Transport (passt) binding, enable the plugin and configure the VM network interface in the web console.

IMPORTANT

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

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

Prerequisites

- You are logged in to the OpenShift Container Platform web console.

Procedure

1. Follow these steps to enable the passt network binding plugin Technology Preview feature:
   a. From the **Virtualization** perspective, click **Overview**.
   b. On the **Virtualization** page, click the **Settings** tab.
   c. Click **Preview features** and set **Enable Passt binding for primary user-defined networks** to on.
2. From the **Virtualization** perspective, click **VirtualMachines**.
3. Select a VM to open the **VirtualMachine details** page.
4. Click the **Configuration** tab.
5. Click **Network**.
6. Click the Options menu on the **Network interfaces** page and select **Edit**.
7. In the **Edit network interface** dialog, select the default pod network attachment from the **Network** list.
8. Expand **Advanced** and then select the **Passt** binding.
9. Click **Save**.
10. If your VM is running, restart it for the changes to take effect.

11.3.3.2. Attaching a virtual machine to the primary user-defined network by using the CLI

You can connect a virtual machine (VM) to the primary user-defined network (UDN) by using the CLI.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **VirtualMachine** manifest to add the UDN interface details, as in the following example:
Example **VirtualMachine** manifest:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: my-namespace (1)
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - name: udn-l2-net (2)
              binding:
                name: l2bridge (3)
# ...
      networks:
      - name: udn-l2-net (4)
        pod: {}
# ...
```

(1) The namespace in which the VM is located. This value must match the namespace in which the UDN is defined.

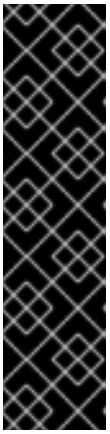
(2) The name of the user-defined network interface.

(3) The name of the binding plugin that is used to connect the interface to the VM. The possible values are **l2bridge** and **passt**. The default value is **l2bridge**.

(4) The name of the network. This must match the value of the **spec.template.spec.domain.devices.interfaces.name** field.

2. Optional: If you are using the Plug a Simple Socket Transport (passt) network binding plugin, set the **hco.kubevirt.io/deployPasstNetworkBinding** annotation to **true** in the **HyperConverged** custom resource (CR) by running the following command:

```
$ oc annotate hco kubevirt-hyperconverged -n kubevirt-hyperconverged
hco.kubevirt.io/deployPasstNetworkBinding=true --overwrite
```



IMPORTANT

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

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

3. Apply the **VirtualMachine** manifest by running the following command:

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

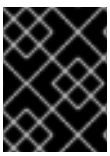
11.3.4. Additional resources

- About user-defined networks

11.4. CONNECTING A VIRTUAL MACHINE TO A SECONDARY LOCALNET USER-DEFINED NETWORK

You can connect a virtual machine (VM) to an OVN-Kubernetes localnet secondary network by using the CLI. Cluster administrators can use the **ClusterUserDefinedNetwork** (CUDN) custom resource definition (CRD) to create a shared OVN-Kubernetes network across multiple namespaces.

An OVN-Kubernetes secondary network is compatible with the multi-network policy API which provides the **MultiNetworkPolicy** custom resource definition (CRD) to control traffic flow to and from VMs.



IMPORTANT

You must use the **ipBlock** attribute to define network policy ingress and egress rules for specific CIDR blocks. Using pod or namespace selector policy peers is not supported.

A localnet topology connects the secondary network to the physical underlay. This enables both east-west cluster traffic and access to services running outside the cluster, but it requires additional configuration of the underlying Open vSwitch (OVS) system on cluster nodes.

11.4.1. Creating a user-defined-network for localnet topology by using the CLI

You can create a secondary cluster-scoped user-defined-network (CUDN) for the localnet network topology by using the CLI.

Prerequisites

- You are logged in to the cluster as a user with **cluster-admin** privileges.
- You have installed the OpenShift CLI (**oc**).
- You installed the Kubernetes NMState Operator.

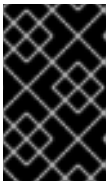
Procedure

1. Create a **NodeNetworkConfigurationPolicy** object to map the OVN-Kubernetes secondary network to an Open vSwitch (OVS) bridge:

   Example NodeNetworkConfigurationPolicy manifest

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: mapping 1
spec:
  nodeSelector:
    node-role.kubernetes.io/worker: '' 2
  desiredState:
    ovn:
      bridge-mappings:
      - localnet: localnet1 3
        bridge: br-ex 4
        state: present 5
```

   1. The name of the configuration object.
   2. Specifies the nodes to which the node network configuration policy is applied. The recommended node selector value is **node-role.kubernetes.io/worker: ''**.
   3. The name of the additional network from which traffic is forwarded to the OVS bridge. This attribute must match the value of the **spec.network.localnet.physicalNetworkName** field of the **ClusterUserDefinedNetwork** object that defines the OVN-Kubernetes additional network. This example uses the name **localnet1**.
   4. The name of the OVS bridge on the node. This value is required if the **state** attribute is **present** or not specified.
   5. The state of the mapping. Must be either **present** to add the mapping or **absent** to remove the mapping. The default value is **present**.



IMPORTANT

OpenShift Virtualization does not support Linux bridge bonding modes 0, 5, and 6. For more information, see Which bonding modes work when used with a bridge that virtual machine guests or containers connect to?.

2. Apply the **NodeNetworkConfigurationPolicy** manifest by running the following command:

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

   where:

   <filename>

   Specifies the name of your **NodeNetworkConfigurationPolicy** manifest YAML file.

3. Create a **ClusterUserDefinedNetwork** object to create a localnet secondary network:

Example ClusterUserDefinedNetwork manifest

```
apiVersion: k8s.ovn.org/v1
kind: ClusterUserDefinedNetwork
metadata:
  name: cudn-localnet 1
spec:
  namespaceSelector: 2
    matchExpressions: 3
    - key: kubernetes.io/metadata.name
      operator: In 4
      values: ["red", "blue"]
  network:
    topology: Localnet 5
    localnet:
      role: Secondary 6
      physicalNetworkName: localnet1 7
      ipam:
        mode: Disabled 8
# ...
```

1. The name of the **ClusterUserDefinedNetwork** custom resource.
2. The set of namespaces that the cluster UDN applies to. The namespace selector must not point to the following values: **default**; an **openshift-*** namespace; or any global namespaces that are defined by the Cluster Network Operator (CNO).
3. The type of selector. In this example, the **matchExpressions** selector selects objects that have the label **kubernetes.io/metadata.name** with the value **red** or **blue**.
4. The type of operator. Possible values are **In**, **NotIn**, and **Exists**.
5. The topological configuration of the network. A **Localnet** topology connects the logical network to the physical underlay.
6. Specifies whether the UDN is primary or secondary. The required value is **Secondary** for **topology: Localnet**.
7. The name of the OVN-Kubernetes bridge mapping that is configured on the node. This value must match the **spec.desiredState.ovn.bridge-mappings.localnet** field in the **NodeNetworkConfigurationPolicy** manifest that you previously created. This ensures that you are bridging to the intended segment of your physical network.
8. Specifies whether IP address management (IPAM) is enabled or disabled. The required value is **Disabled**. OpenShift Virtualization does not support configuring IPAM for virtual machines.

4. Apply the **ClusterUserDefinedNetwork** manifest by running the following command:

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

where:

<filename>
Specifies the name of your **ClusterUserDefinedNetwork** manifest YAML file.

11.4.2. Creating a namespace for secondary user-defined networks by using the CLI

You can create a namespace to be used with an existing secondary cluster-scoped user-defined network (CUDN) by using the CLI.

Prerequisites

- You are logged in to the cluster as a user with **cluster-admin** permissions.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **Namespace** object similar to the following example:

   Example Namespace manifest

   ```
   apiVersion: v1
   kind: Namespace
   metadata:
     name: red
   # ...
   ```

2. Apply the **Namespace** manifest by running the following command:

   ```
   oc apply -f <filename>.yaml
   ```

   where:

   <filename>
   Specifies the name of your **Namespace** manifest YAML file.

11.4.3. Attaching a virtual machine to secondary user-defined networks by using the CLI

You can connect a virtual machine (VM) to multiple secondary cluster-scoped user-defined networks (CUDNs) by configuring the interface binding.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **VirtualMachine** manifest to add the CUDN interface details, as in the following example:

Example VirtualMachine manifest

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: red (1)
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - name: secondary_localnet (2)
              bridge: {}
        machine:
          type: ""
        resources:
          requests:
            memory: 2048M
      networks:
      - name: secondary_localnet (3)
        multus:
          networkName: <localnet_cudn_name> (4)
# ...
```

(1) The namespace in which the VM is located. This value must match a namespace that is associated with the secondary CUDN.

(2) The name of the secondary user-defined network interface.

(3) The name of the network. This must match the value of the **spec.template.spec.domain.devices.interfaces.name** field.

(4) The name of the localnet **ClusterUserDefinedNetwork** object that you previously created.

2. Apply the **VirtualMachine** manifest by running the following command:

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

where:

<filename>

Specifies the name of your **VirtualMachine** manifest YAML file.

NOTE

When running OpenShift Virtualization on IBM Z® using an OSA card, be aware that the OSA card only forwards network traffic to devices that are registered with the OSA device. As a result, any traffic destined for unregistered devices is not forwarded.

11.4.4. Additional resources

- About the **ClusterUserDefinedNetwork** CR
- OSA interface traffic forwarding

11.5. EXPOSING A VIRTUAL MACHINE BY USING A SERVICE

You can expose a virtual machine within the cluster or outside the cluster by creating a **Service** object.

11.5.1. About services

A Kubernetes service exposes network access for clients to an application running on a set of pods. Services offer abstraction, load balancing, and, in the case of the **NodePort** and **LoadBalancer** types, exposure to the outside world.

ClusterIP

Exposes the service on an internal IP address and as a DNS name to other applications within the cluster. A single service can map to multiple virtual machines. When a client tries to connect to the service, the client's request is load balanced among available backends. **ClusterIP** is the default service type.

NodePort

Exposes the service on the same port of each selected node in the cluster. **NodePort** makes a port accessible from outside the cluster, as long as the node itself is externally accessible to the client.

LoadBalancer

Creates an external load balancer in the current cloud (if supported) and assigns a fixed, external IP address to the service.

NOTE

For on-premise clusters, you can configure a load-balancing service by deploying the MetalLB Operator.

Additional resources

- Installing the MetalLB Operator
- Configuring services to use MetalLB

11.5.2. Dual-stack support

If IPv4 and IPv6 dual-stack networking is enabled for your cluster, you can create a service that uses IPv4, IPv6, or both, by defining the **spec.ipFamilyPolicy** and the **spec.ipFamilies** fields in the **Service** object.

The **spec.ipFamilyPolicy** field can be set to one of the following values:

SingleStack

The control plane assigns a cluster IP address for the service based on the first configured service cluster IP range.

PreferDualStack

The control plane assigns both IPv4 and IPv6 cluster IP addresses for the service on clusters that have dual-stack configured.

RequireDualStack

This option fails for clusters that do not have dual-stack networking enabled. For clusters that have dual-stack configured, the behavior is the same as when the value is set to **PreferDualStack**. The control plane allocates cluster IP addresses from both IPv4 and IPv6 address ranges.

You can define which IP family to use for single-stack or define the order of IP families for dual-stack by setting the **spec.ipFamilies** field to one of the following array values:

- **[IPv4]**
- **[IPv6]**
- **[IPv4, IPv6]**
- **[IPv6, IPv4]**

11.5.3. Creating a service by using the CLI

You can create a service and associate it with a virtual machine (VM) by using the command line.

Prerequisites

- You configured the cluster network to support the service.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **VirtualMachine** manifest to add the label for service creation:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  runStrategy: Halted
  template:
    metadata:
      labels:
        special: key 1
# ...
```

1 Add **special: key** to the **spec.template.metadata.labels** stanza.

NOTE

Labels on a virtual machine are passed through to the pod. The **special: key** label must match the label in the **spec.selector** attribute of the **Service** manifest.

2. Save the **VirtualMachine** manifest file to apply your changes.

3. Create a **Service** manifest to expose the VM:

```
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) Specify the label that you added to the **spec.template.metadata.labels** stanza of the **VirtualMachine** manifest.

(2) Specify **ClusterIP**, **NodePort**, or **LoadBalancer**.

(3) Specifies a collection of network ports and protocols that you want to expose from the virtual machine.

4. Save the **Service** manifest file.

5. Create the service by running the following command:

```
$ oc create -f example-service.yaml
```

6. Restart the VM to apply the changes.

Verification

- Query the **Service** object to verify that it is available:

```
$ oc get service -n example-namespace
```

11.5.4. Additional resources

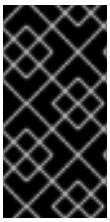
- Configuring ingress cluster traffic using a NodePort
- Configuring ingress cluster traffic using a load balancer

11.6. ACCESSING A VIRTUAL MACHINE BY USING ITS INTERNAL FQDN

You can access a virtual machine (VM) that is connected to the default internal pod network on a stable fully qualified domain name (FQDN) by using headless services.

A Kubernetes *headless service* is a form of service that does not allocate a cluster IP address to

represent a set of pods. Instead of providing a single virtual IP address for the service, a headless service creates a DNS record for each pod associated with the service. You can expose a VM through its FQDN without having to expose a specific TCP or UDP port.



IMPORTANT

If you created a VM by using the OpenShift Container Platform web console, you can find its internal FQDN listed in the **Network** tile on the **Overview** tab of the **VirtualMachine details** page. For more information about connecting to the VM, see Connecting to a virtual machine by using its internal FQDN.

11.6.1. Creating a headless service in a project by using the CLI

To create a headless service in a namespace, add the **clusterIP: None** parameter to the service YAML definition.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **Service** manifest to expose the VM, such as the following example:

```
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) The name of the service. This must match the **spec.subdomain** attribute in the **VirtualMachine** manifest file.

(2) This service selector must match the **expose:me** label in the **VirtualMachine** manifest file.

(3) Specifies a headless service.

(4) The list of ports that are exposed by the service. You must define at least one port. This can be any arbitrary value as it does not affect the headless service.

2. Save the **Service** manifest file.

3. Create the service by running the following command:

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

11.6.2. Mapping a virtual machine to a headless service by using the CLI

To connect to a virtual machine (VM) from within the cluster by using its internal fully qualified domain name (FQDN), you must first map the VM to a headless service. Set the **spec.hostname** and **spec.subdomain** parameters in the VM configuration file.

If a headless service exists with a name that matches the subdomain, a unique DNS A record is created for the VM in the form of **<vm.spec.hostname>.<vm.spec.subdomain>.<vm.metadata.namespace>.svc.cluster.local**.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **VirtualMachine** manifest to add the service selector label and subdomain by running the following command:

```
$ oc edit vm <vm_name>
```

Example VirtualMachine manifest file

```
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. The **expose:me** label must match the **spec.selector** attribute of the **Service** manifest that you previously created.
2. If this attribute is not specified, the resulting DNS A record takes the form of **<vm.metadata.name>.<vm.spec.subdomain>.<vm.metadata.namespace>.svc.cluster.local**.
3. The **spec.subdomain** attribute must match the **metadata.name** value of the **Service** object.

2. Save your changes and exit the editor.
3. Restart the VM to apply the changes.

11.6.3. Connecting to a virtual machine by using its internal FQDN

You can connect to a virtual machine (VM) by using its internal fully qualified domain name (FQDN).

Prerequisites

- You have installed the **virtctl** tool.
- You have identified the internal FQDN of the VM from the web console or by mapping the VM to a headless service. The internal FQDN has the format **<vm.spec.hostname>.<vm.spec.subdomain>.<vm.metadata.namespace>.svc.cluster.local**.

Procedure

1. Connect to the VM console by entering the following command:

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

2. To connect to the VM by using the requested FQDN, run the following command:

```
$ ping myvm.mysubdomain.<namespace>.svc.cluster.local
```

Example output

```
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
```

In the preceding example, the DNS entry for **myvm.mysubdomain.default.svc.cluster.local** points to **10.244.0.57**, which is the cluster IP address that is currently assigned to the VM.

11.6.4. Additional resources

- Exposing a VM by using a service

11.7. CONNECTING A VIRTUAL MACHINE TO A LINUX BRIDGE NETWORK

By default, OpenShift Virtualization is installed with a single, internal pod network.

You can create a Linux bridge network and attach a virtual machine (VM) to the network by performing the following steps:

1. Create a Linux bridge node network configuration policy (NNCP) .
2. Create a Linux bridge network attachment definition (NAD) by using the web console or the command line.
3. Configure the VM to recognize the NAD by using the web console or the command line.

NOTE

OpenShift Virtualization does not support Linux bridge bonding modes 0, 5, and 6. For more information, see Which bonding modes work when used with a bridge that virtual machine guests or containers connect to?.

11.7.1. Creating a Linux bridge NNCP

You can create a **NodeNetworkConfigurationPolicy** (NNCP) manifest for a Linux bridge network.

Prerequisites

- You have installed the Kubernetes NMState Operator.

Procedure

- Create the **NodeNetworkConfigurationPolicy** manifest. This example includes sample values that you must replace with your own information.

```
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) Name of the policy.

(2) Name of the interface.

(3) Optional: Human-readable description of the interface.

(4) The type of interface. This example creates a bridge.

(5) The requested state for the interface after creation.

(6) Disables IPv4 in this example.

(7) Disables STP in this example.

(8) The node NIC to which the bridge is attached.

NOTE

To create the NNCP manifest for a Linux bridge using OSA with IBM Z®, you must disable VLAN filtering by the setting the **rx-vlan-filter** to **false** in the **NodeNetworkConfigurationPolicy** manifest.

Alternatively, if you have SSH access to the node, you can disable VLAN filtering by running the following command:

```
$ sudo ethtool -K <osa-interface-name> rx-vlan-filter off
```

11.7.2. Creating a Linux bridge NAD

You can create a Linux bridge network attachment definition (NAD) by using the OpenShift Container Platform web console or command line.

11.7.2.1. Creating a Linux bridge NAD by using the web console

You can create a network attachment definition (NAD) to provide layer-2 networking to pods and virtual machines by using the OpenShift Container Platform web console.

WARNING

Configuring IP address management (IPAM) in a network attachment definition for virtual machines is not supported.

Procedure

1. In the web console, click **Networking → NetworkAttachmentDefinitions**.
2. Click **Create Network Attachment Definition**

NOTE

The network attachment definition must be in the same namespace as the pod or virtual machine.

3. Enter a unique **Name** and optional **Description**.
4. Select **CNV Linux bridge** from the **Network Type** list.
5. Enter the name of the bridge in the **Bridge Name** field.
6. Optional: If the resource has VLAN IDs configured, enter the ID numbers in the **VLAN Tag Number** field.

NOTE

OSA interfaces on IBM Z® do not support VLAN filtering and VLAN-tagged traffic is dropped. Avoid using VLAN-tagged NADs with OSA interfaces.

7. Optional: Select **MAC Spoof Check** to enable MAC spoof filtering. This feature provides security against a MAC spoofing attack by allowing only a single MAC address to exit the pod.

8. Click **Create**.

11.7.2.2. Creating a Linux bridge NAD by using the CLI

You can create a network attachment definition (NAD) to provide layer-2 networking to pods and virtual machines (VMs) by using the command line.

The NAD and the VM must be in the same namespace.

WARNING

Configuring IP address management (IPAM) in a network attachment definition for virtual machines is not supported.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Add the VM to the **NetworkAttachmentDefinition** configuration, as in the following example:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bridge-network (1)
  annotations:
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/br1 (2)
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "bridge-network", (3)
      "type": "bridge", (4)
      "bridge": "br1", (5)
      "macspoofchk": false, (6)
      "vlan": 100, (7)
      "disableContainerInterface": true,
      "preserveDefaultVlan": false (8)
    }
```

1. The name for the **NetworkAttachmentDefinition** object.
2. Optional: Annotation key-value pair for node selection for the bridge configured on some nodes. If you add this annotation to your network attachment definition, your virtual machine instances will only run on the nodes that have the defined bridge connected.
3. The name for the configuration. It is recommended to match the configuration name to the **name** value of the network attachment definition.
4. The actual name of the Container Network Interface (CNI) plugin that provides the network for this network attachment definition. Do not change this field unless you want to use a different CNI.
5. The name of the Linux bridge configured on the node. The name should match the interface bridge name defined in the **NodeNetworkConfigurationPolicy** manifest.
6. Optional: A flag to enable the MAC spoof check. When set to **true**, you cannot change the MAC address of the pod or guest interface. This attribute allows only a single MAC address to exit the pod, which provides security against a MAC spoofing attack.
7. Optional: The VLAN tag. No additional VLAN configuration is required on the node network configuration policy.

NOTE

OSA interfaces on IBM Z® do not support VLAN filtering and VLAN-tagged traffic is dropped. Avoid using VLAN-tagged NADs with OSA interfaces.

8. Optional: Indicates whether the VM connects to the bridge through the default VLAN. The default value is **true**.

2. Optional: If you want to connect a VM to the native network, configure the Linux bridge **NetworkAttachmentDefinition** manifest without specifying any VLAN:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bridge-network
  annotations:
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/br1
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "bridge-network",
      "type": "bridge",
      "bridge": "br1",
      "macspoofchk": false,
      "disableContainerInterface": true
    }
```

3. Create the network attachment definition:

```
$ oc create -f network-attachment-definition.yaml 1
```

–

1. Where **network-attachment-definition.yaml** is the file name of the network attachment definition manifest.

Verification

- Verify that the network attachment definition was created by running the following command:

```
$ oc get network-attachment-definition bridge-network
```

11.7.2.3. Enabling port isolation for a Linux bridge NAD

You can enable port isolation for a Linux bridge network attachment definition (NAD) so that virtual machines (VMs) or pods that run on the same virtual LAN (VLAN) can operate in isolation from one another. The Linux bridge NAD creates a virtual bridge, or *virtual switch*, between network interfaces and the physical network.

Isolating ports in this way can provide enhanced security for VM workloads that run on the same node.

Prerequisites

- For VMs, you configured either a static or dynamic IP address for each VM. See "Configuring IP addresses for virtual machines".
- You created a Linux bridge NAD by using either the web console or the command-line interface.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the Linux bridge NAD by setting **portIsolation** to **true**:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bridge-network
  annotations:
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/br1
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "bridge-network", 1
      "type": "bridge", 2
      "bridge": "br1", 3
      "preserveDefaultVlan": false,
      "vlan": 100,
      "disableContainerInterface": false,
      "portIsolation": true 4
    }
# ...
```

1. The name for the configuration. The name must match the value in the **metadata.name** of the NAD.
2. The actual name of the Container Network Interface (CNI) plugin that provides the network for this network attachment definition. Do not change this field unless you want to use a different CNI.
3. The name of the Linux bridge that is configured on the node. The name must match the interface bridge name defined in the NodeNetworkConfigurationPolicy manifest.
4. Enables or disables port isolation on the virtual bridge. Default value is **false**. When set to **true**, each VM or pod is assigned to an isolated port. The virtual bridge prevents traffic from one isolated port from reaching another isolated port.

2. Apply the configuration:

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

3. Optional: If you edited a running virtual machine, you must restart it for the changes to take effect.

Additional resources

- Configuring IP addresses for virtual machines

11.7.3. Configuring a VM network interface

You can configure a virtual machine (VM) network interface by using the OpenShift Container Platform web console or command line.

11.7.3.1. Configuring a VM network interface by using the web console

You can configure a network interface for a virtual machine (VM) by using the OpenShift Container Platform web console.

Prerequisites

- You created a network attachment definition for the network.

Procedure

1. Navigate to **Virtualization → VirtualMachines**.
2. Click a VM to view the **VirtualMachine details** page.
3. On the **Configuration** tab, click the **Network interfaces** tab.
4. Click **Add network interface**.
5. Enter the interface name and select the network attachment definition from the **Network** list.
6. Click **Save**.
7. Restart or live migrate the VM to apply the changes.

Networking fields

Name	Description
Name	Name for the network interface controller.
Model	Indicates the model of the network interface controller. Supported values are **e1000e** and **virtio**. For IBM Z® (**s390x**) and ARM64 (**arm64**) systems, use the **virtio** NIC model option. The **e1000e** model is not supported on these architectures.
Network	List of available network attachment definitions.
Type	List of available binding methods. Select the binding method suitable for the network interface: • Default pod network: **masquerade** • Linux bridge network: **bridge** • SR-IOV network: **SR-IOV** On IBM Z®, **SR-IOV** is not supported.
MAC Address	MAC address for the network interface controller. If a MAC address is not specified, one is assigned automatically.

11.7.3.2. Configuring a VM network interface by using the CLI

You can configure a virtual machine (VM) network interface for a bridge network by using the command line.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- Shut down the virtual machine before editing the configuration. If you edit a running virtual machine, you must restart the virtual machine for the changes to take effect.

Procedure

1. Add the bridge interface and the network attachment definition to the VM configuration as in the following example:

```
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: bridge-network 3
```

1. The name of the bridge interface.
2. The name of the network. This value must match the **name** value of the corresponding **spec.template.spec.domain.devices.interfaces** entry.
3. The name of the network attachment definition.

2. Apply the configuration:

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

3. Optional: If you edited a running virtual machine, you must restart it for the changes to take effect.

NOTE

When running OpenShift Virtualization on IBM Z® using an OSA card, you must register the MAC address of the device. For more information, see OSA interface traffic forwarding (IBM documentation).

11.8. CONNECTING A VIRTUAL MACHINE TO AN SR-IOV NETWORK

You can connect a virtual machine (VM) to a Single Root I/O Virtualization (SR-IOV) network by performing the following steps:

- Configuring an SR-IOV network device
- Configuring an SR-IOV network
- Connecting the VM to the SR-IOV network

11.8.1. Configuring SR-IOV network devices

The SR-IOV Network Operator adds the **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CustomResourceDefinition to OpenShift Container Platform. You can configure an SR-IOV network device by creating a SriovNetworkNodePolicy custom resource (CR).

NOTE

When applying the configuration specified in a **SriovNetworkNodePolicy** object, the SR-IOV Operator might drain the nodes, and in some cases, reboot nodes. Reboot only happens in the following cases:

- With Mellanox NICs (**mlx5** driver) a node reboot happens every time the number of virtual functions (VFs) increase on a physical function (PF).
- With Intel NICs, a reboot only happens if the kernel parameters do not include **intel_iommu=on** and **iommu=pt**.

It might take several minutes for a configuration change to apply.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the cluster as a user with the **cluster-admin** role.
- You have installed the SR-IOV Network Operator.
- You have enough available nodes in your cluster to handle the evicted workload from drained nodes.
- You have not selected any control plane nodes for SR-IOV network device configuration.

Procedure

1. Create an **SriovNetworkNodePolicy** object, and then save the YAML in the **<name>-sriov-node-network.yaml** file. Replace **<name>** with the name for this configuration.

```
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 Specify a name for the CR object.

2 Specify the namespace where the SR-IOV Operator is installed.

3 Specify the resource name of the SR-IOV device plugin. You can create multiple **SriovNetworkNodePolicy** objects for a resource name.

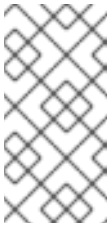
4 Specify the node selector to select which nodes are configured. Only SR-IOV network devices on selected nodes are configured. The SR-IOV Container Network Interface (CNI) plugin and device plugin are deployed only on selected nodes.

5 Optional: Specify an integer value between **0** and **99**. A smaller number gets higher priority, so a priority of **10** is higher than a priority of **99**. The default value is **99**.

6 Optional: Specify a value for the maximum transmission unit (MTU) of the virtual function. The maximum MTU value can vary for different NIC models.

7 Specify the number of the virtual functions (VF) to create for the SR-IOV physical network device. For an Intel network interface controller (NIC), the number of VFs cannot be larger than the total VFs supported by the device. For a Mellanox NIC, the number of VFs cannot be larger than **127**.

8 The **nicSelector** mapping selects the Ethernet device for the Operator to configure. You do not need to specify values for all the parameters.



NOTE

It is recommended to identify the Ethernet adapter with enough precision to minimize the possibility of selecting an Ethernet device unintentionally. If you specify **rootDevices**, you must also specify a value for **vendor**, **deviceID**, or **pfNames**.

If you specify both **pfNames** and **rootDevices** at the same time, ensure that they point to an identical device.

9 Optional: Specify the vendor hex code of the SR-IOV network device. The only allowed values are either **8086** or **15b3**.

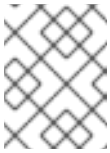
10 Optional: Specify the device hex code of SR-IOV network device. The only allowed values are **158b**, **1015**, **1017**.

11 Optional: The parameter accepts an array of one or more physical function (PF) names for the Ethernet device.

12 The parameter accepts an array of one or more PCI bus addresses for the physical function of the Ethernet device. Provide the address in the following format: **0000:02:00.1**.

13 The **vfio-pci** driver type is required for virtual functions in OpenShift Virtualization.

14 Optional: Specify whether to enable remote direct memory access (RDMA) mode. For a Mellanox card, set **isRdma** to **false**. The default value is **false**.



NOTE

If **isRDMA** flag is set to **true**, you can continue to use the RDMA enabled VF as a normal network device. A device can be used in either mode.

2. Optional: Label the SR-IOV capable cluster nodes with **SriovNetworkNodePolicy.Spec.NodeSelector** if they are not already labeled. For more information about labeling nodes, see "Understanding how to update labels on nodes".

3. Create the **SriovNetworkNodePolicy** object:

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

   where **<name>** specifies the name for this configuration.

   After applying the configuration update, all the pods in **sriov-network-operator** namespace transition to the **Running** status.

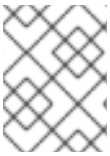
4. To verify that the SR-IOV network device is configured, enter the following command. Replace **<node_name>** with the name of a node with the SR-IOV network device that you just configured.

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

11.8.2. Configuring SR-IOV additional network

You can configure an additional network that uses SR-IOV hardware by creating an **SriovNetwork** object.

When you create an **SriovNetwork** object, the SR-IOV Network Operator automatically creates a **NetworkAttachmentDefinition** object.



NOTE

Do not modify or delete an **SriovNetwork** object if it is attached to pods or virtual machines in a **running** state.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

1. Create the following **SriovNetwork** object, and then save the YAML in the **<name>-sriov-network.yaml** file. Replace **<name>** with a name for this additional network.

```
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. Replace **<name>** with a name for the object. The SR-IOV Network Operator creates a **NetworkAttachmentDefinition** object with same name.
2. Specify the namespace where the SR-IOV Network Operator is installed.
3. Replace **<sriov_resource_name>** with the value for the **.spec.resourceName** parameter from the **SriovNetworkNodePolicy** object that defines the SR-IOV hardware for this additional network.
4. Replace **<target_namespace>** with the target namespace for the SriovNetwork. Only pods or virtual machines in the target namespace can attach to the SriovNetwork.
5. Optional: Replace **<vlan>** with a Virtual LAN (VLAN) ID for the additional network. The integer value must be from **0** to **4095**. The default value is **0**.
6. Optional: Replace **<spoof_check>** with the spoof check mode of the VF. The allowed values are the strings **"on"** and **"off"**.

IMPORTANT

You must enclose the value you specify in quotes or the CR is rejected by the SR-IOV Network Operator.

7. Optional: Replace **<link_state>** with the link state of virtual function (VF). Allowed value are **enable**, **disable** and **auto**.
8. Optional: Replace **<max_tx_rate>** with a maximum transmission rate, in Mbps, for the VF.
9. Optional: Replace **<min_tx_rate>** with a minimum transmission rate, in Mbps, for the VF. This value should always be less than or equal to Maximum transmission rate.

NOTE

Intel NICs do not support the **minTxRate** parameter. For more information, see BZ#1772847.

10. Optional: Replace **<vlan_qos>** with an IEEE 802.1p priority level for the VF. The default value is **0**.
11. Optional: Replace **<trust_vf>** with the trust mode of the VF. The allowed values are the strings **"on"** and **"off"**.

IMPORTANT

You must enclose the value you specify in quotes or the CR is rejected by the SR-IOV Network Operator.

12 Optional: Replace **<capabilities>** with the capabilities to configure for this network.

2. To create the object, enter the following command. Replace **<name>** with a name for this additional network.

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

3. Optional: To confirm that the **NetworkAttachmentDefinition** object associated with the **SriovNetwork** object that you created in the previous step exists, enter the following command. Replace **<namespace>** with the namespace you specified in the **SriovNetwork** object.

```
$ oc get net-attach-def -n <namespace>
```

11.8.3. Connecting a virtual machine to an SR-IOV network by using the CLI

You can connect the virtual machine (VM) to the SR-IOV network by including the network details in the VM configuration.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Add the SR-IOV network details to the **spec.domain.devices.interfaces** and **spec.networks** stanzas of the VM configuration as in the following example:

```
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 Specify a unique name for the SR-IOV interface.

2 Specify the name of the SR-IOV interface. This must be the same as the **interfaces.name** that you defined earlier.

3 Specify the name of the SR-IOV network attachment definition.

2. Apply the virtual machine configuration:

```
$ oc apply -f <vm_sriov>.yaml 1
```

1 The name of the virtual machine YAML file.

11.8.4. Connecting a VM to an SR-IOV network by using the web console

You can connect a VM to the SR-IOV network by including the network details in the VM configuration.

Prerequisites

- You must create a network attachment definition for the network.

Procedure

1. Navigate to **Virtualization → VirtualMachines**.
2. Click a VM to view the **VirtualMachine details** page.
3. On the **Configuration** tab, click the **Network interfaces** tab.
4. Click **Add network interface**.
5. Enter the interface name.
6. Select an SR-IOV network attachment definition from the **Network** list.
7. Select **SR-IOV** from the **Type** list.
8. Optional: Add a network **Model** or **Mac address**.
9. Click **Save**.
10. Restart or live-migrate the VM to apply the changes.

11.8.5. Additional resources

- Configuring DPDK workloads for improved performance

11.9. USING DPDK WITH SR-IOV

The Data Plane Development Kit (DPDK) provides a set of libraries and drivers for fast packet processing.

You can configure clusters and virtual machines (VMs) to run DPDK workloads over SR-IOV networks.

11.9.1. Configuring a cluster for DPDK workloads

You can configure an OpenShift Container Platform cluster to run Data Plane Development Kit (DPDK) workloads for improved network performance.

Prerequisites

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

- You have installed the OpenShift CLI (**oc**).
- You have installed the SR-IOV Network Operator.
- You have installed the Node Tuning Operator.

Procedure

1. Map your compute nodes topology to determine which Non-Uniform Memory Access (NUMA) CPUs are isolated for DPDK applications and which ones are reserved for the operating system (OS).
2. If your OpenShift Container Platform cluster uses separate control plane and compute nodes for high-availability:
   a. Label a subset of the compute nodes with a custom role; for example, **worker-dpdk**:

   ```
   $ oc label node <node_name> node-role.kubernetes.io/worker-dpdk=""
   ```

   b. Create a new **MachineConfigPool** manifest that contains the **worker-dpdk** label in the **spec.machineConfigSelector** object:

   Example MachineConfigPool manifest

   ```
   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: ""
   ```

3. Create a **PerformanceProfile** manifest that applies to the labeled nodes and the machine config pool that you created in the previous steps. The performance profile specifies the CPUs that are isolated for DPDK applications and the CPUs that are reserved for house keeping.

   Example PerformanceProfile manifest

   ```
   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
```

NOTE

The compute nodes automatically restart after you apply the **MachineConfigPool** and **PerformanceProfile** manifests.

4. Retrieve the name of the generated **RuntimeClass** resource from the **status.runtimeClass** field of the **PerformanceProfile** object:

```
$ oc get performanceprofiles.performance.openshift.io profile-1 -
o=jsonpath='{.status.runtimeClass}{"\n"}'
```

5. Set the previously obtained **RuntimeClass** name as the default container runtime class for the **virt-launcher** pods by editing the **HyperConverged** custom resource (CR):

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
    --type='json' -p='[{"op": "add", "path": "/spec/defaultRuntimeClass", "value":"<runtimeclass-
name>"}]'
```

NOTE

Editing the **HyperConverged** CR changes a global setting that affects all VMs that are created after the change is applied.

6. If your DPDK-enabled compute nodes use Simultaneous multithreading (SMT), enable the **AlignCPUs** enabler by editing the **HyperConverged** CR:

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

NOTE

Enabling **AlignCPUs** allows OpenShift Virtualization to request up to two additional dedicated CPUs to bring the total CPU count to an even parity when using emulator thread isolation.

7. Create an **SriovNetworkNodePolicy** object with the **spec.deviceType** field set to **vfio-pci**:

Example SriovNetworkNodePolicy manifest

```
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"
```

Additional resources

- Using CPU Manager and Topology Manager
- Configuring huge pages
- Creating a custom machine config pool

11.9.1.1. Removing a custom machine config pool for high-availability clusters

You can delete a custom machine config pool that you previously created for your high-availability cluster.

Prerequisites

- You have access to the cluster as a user with **cluster-admin** permissions.
- You have installed the OpenShift CLI (**oc**).
- You have created a custom machine config pool by labeling a subset of the compute nodes with a custom role and creating a **MachineConfigPool** manifest with that label.

Procedure

1. Remove the **worker-dpdk** label from the compute nodes by running the following command:

```
$ oc label node <node_name> node-role.kubernetes.io/worker-dpdk-
```

2. Delete the **MachineConfigPool** manifest that contains the **worker-dpdk** label by entering the following command:

```
$ oc delete mcp worker-dpdk
```

11.9.2. Configuring a project for DPDK workloads

You can configure the project to run DPDK workloads on SR-IOV hardware.

Prerequisites

- Your cluster is configured to run DPDK workloads.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a namespace for your DPDK applications:

```
$ oc create ns dpdk-ns
```

2. Create an **SriovNetwork** object that references the **SriovNetworkNodePolicy** object. When you create an **SriovNetwork** object, the SR-IOV Network Operator automatically creates a **NetworkAttachmentDefinition** object.

Example SriovNetwork manifest

```
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-ns (1)
  resourceName: intel_nics_dpdk (2)
  spoofChk: "off"
  trust: "on"
  vlan: 1019
```

(1) The namespace where the **NetworkAttachmentDefinition** object is deployed.

(2) The value of the **spec.resourceName** attribute of the **SriovNetworkNodePolicy** object that was created when configuring the cluster for DPDK workloads.

3. Optional: Run the virtual machine latency checkup to verify that the network is properly configured.

Additional resources

- Working with projects
- Virtual machine latency checkup

11.9.3. Configuring a virtual machine for DPDK workloads

You can run Data Packet Development Kit (DPDK) workloads on virtual machines (VMs) to achieve lower latency and higher throughput for faster packet processing in the user space. DPDK uses the SR-IOV network for hardware-based I/O sharing.

Prerequisites

- Your cluster is configured to run DPDK workloads.
- You have created and configured the project in which the VM will run.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **VirtualMachine** manifest to include information about the SR-IOV network interface, CPU topology, CRI-O annotations, and huge pages:

   Example VirtualMachine manifest

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: rhel-dpdk-vm
spec:
  runStrategy: Always
  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. This annotation specifies that load balancing is disabled for CPUs that are used by the container.
2. This annotation specifies that the CPU quota is disabled for CPUs that are used by the container.
3. This annotation specifies that Interrupt Request (IRQ) load balancing is disabled for CPUs that are used by the container.
4. The number of sockets inside the VM. This field must be set to **1** for the CPUs to be scheduled from the same Non-Uniform Memory Access (NUMA) node.
5. The number of cores inside the VM. This must be a value greater than or equal to **1**. In this example, the VM is scheduled with 5 hyper-threads or 10 CPUs.
6. The size of the huge pages. The possible values for x86-64 architecture are 1Gi and 2Mi. In this example, the request is for 8 huge pages of size 1Gi.
7. The name of the SR-IOV **NetworkAttachmentDefinition** object.

2. Save and exit the editor.
3. Apply the **VirtualMachine** manifest:

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

4. Configure the guest operating system. The following example shows the configuration steps for RHEL 9 operating system:
   a. Configure huge pages by using the GRUB bootloader command-line interface. In the following example, 8 1G huge pages are specified.

```
$ grubby --update-kernel=ALL --args="default_hugepagesz=1GB hugepagesz=1G
hugepages=8"
```

   b. To achieve low-latency tuning by using the **cpu-partitioning** profile in the TuneD application, run the following commands:

```
$ dnf install -y tuned-profiles-cpu-partitioning
```

```
$ echo isolated_cores=2-9 > /etc/tuned/cpu-partitioning-variables.conf
```

The first two CPUs (0 and 1) are set aside for house keeping tasks and the rest are isolated for the DPDK application.

```
$ tuned-adm profile cpu-partitioning
```

c. Override the SR-IOV NIC driver by using the **driverctl** device driver control utility:

```
$ dnf install -y driverctl
```

```
$ driverctl set-override 0000:07:00.0 vfio-pci
```

5. Restart the VM to apply the changes.

11.10. CONNECTING A VIRTUAL MACHINE TO AN OVN-KUBERNETES LAYER 2 SECONDARY NETWORK

You can connect a virtual machine (VM) to an OVN-Kubernetes **layer2** secondary network by using the CLI.

A **layer2** topology connects workloads by a cluster-wide logical switch. The OVN-Kubernetes Container Network Interface (CNI) plugin uses the Geneve (Generic Network Virtualization Encapsulation) protocol to create an overlay network between nodes. You can use this overlay network to connect VMs on different nodes, without having to configure any additional physical networking infrastructure.

NOTE

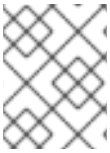
An OVN-Kubernetes secondary network is compatible with the multi-network policy API which provides the **MultiNetworkPolicy** custom resource definition (CRD) to control traffic flow to and from VMs. You must use the **ipBlock** attribute to define network policy ingress and egress rules for specific CIDR blocks. You cannot use pod or namespace selectors for virtualization workloads.

To configure an OVN-Kubernetes **layer2** secondary network and attach a VM to that network, perform the following steps:

1. Configure an OVN-Kubernetes layer 2 secondary network .
2. Connect the VM to the OVN-Kubernetes layer 2 secondary network .

11.10.1. Creating an OVN-Kubernetes layer 2 NAD

You can create an OVN-Kubernetes network attachment definition (NAD) for the layer 2 network topology by using the OpenShift Container Platform web console or the CLI.



NOTE

Configuring IP address management (IPAM) by specifying the **spec.config.ipam.subnet** attribute in a network attachment definition for virtual machines is not supported.

11.10.1.1. Creating a NAD for layer 2 topology by using the CLI

You can create a network attachment definition (NAD) which describes how to attach a pod to the layer 2 overlay network.

Prerequisites

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

Procedure

1. Create a **NetworkAttachmentDefinition** object:

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: l2-network
  namespace: my-namespace
spec:
  config: |-
    {
            "cniVersion": "0.3.1", (1)
            "name": "my-namespace-l2-network", (2)
            "type": "ovn-k8s-cni-overlay", (3)
            "topology":"layer2", (4)
            "mtu": 1400, (5)
            "netAttachDefName": "my-namespace/l2-network" (6)
    }
```

(1) The Container Network Interface (CNI) specification version. The required value is **0.3.1**.

(2) The name of the network. This attribute is not namespaced. For example, you can have a network named **l2-network** referenced from two different **NetworkAttachmentDefinition** objects that exist in two different namespaces. This feature is useful to connect VMs in different namespaces.

(3) The name of the CNI plugin. The required value is **ovn-k8s-cni-overlay**.

(4) The topological configuration for the network. The required value is **layer2**.

(5) Optional: The maximum transmission unit (MTU) value. If you do not set a value, the Cluster Network Operator (CNO) sets a default MTU value by calculating the difference among the underlay MTU of the primary network interface, the overlay MTU of the pod network, such as the Geneve (Generic Network Virtualization Encapsulation), and byte capacity of any enabled features, such as IPsec.

(6) The value of the **namespace** and **name** fields in the **metadata** stanza of the **NetworkAttachmentDefinition** object.

NetworkAttachmentDefinition object.

NOTE

The previous example configures a cluster-wide overlay without a subnet defined. This means that the logical switch implementing the network only provides layer 2 communication. You must configure an IP address when you create the virtual machine by either setting a static IP address or by deploying a DHCP server on the network for a dynamic IP address.

2. Apply the manifest by running the following command:

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

11.10.1.2. Creating a NAD for layer 2 topology by using the web console

You can create a network attachment definition (NAD) that describes how to attach a pod to the layer 2 overlay network.

Prerequisites

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

Procedure

1. Go to **Networking → NetworkAttachmentDefinitions** in the web console.
2. Click **Create Network Attachment Definition** The network attachment definition must be in the same namespace as the pod or virtual machine using it.
3. Enter a unique **Name** and optional **Description**.
4. Select **OVN Kubernetes L2 overlay network** from the **Network Type** list.
5. Click **Create**.

11.10.2. Attaching a virtual machine to the OVN-Kubernetes layer 2 secondary network

You can attach a virtual machine (VM) to the OVN-Kubernetes layer 2 secondary network interface by using the OpenShift Container Platform web console or the CLI.

11.10.2.1. Attaching a virtual machine to an OVN-Kubernetes secondary network using the CLI

You can connect a virtual machine (VM) to the OVN-Kubernetes secondary network by including the network details in the VM configuration.

Prerequisites

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

Procedure

1. Edit the **VirtualMachine** manifest to add the OVN-Kubernetes secondary network interface details, as in the following example:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-server
spec:
  runStrategy: Always
  template:
    spec:
      domain:
        devices:
          interfaces:
          - name: secondary (1)
            bridge: {}
        resources:
          requests:
            memory: 1024Mi
      networks:
      - name: secondary (2)
        multus:
          networkName: <nad_name> (3)
      nodeSelector:
        node-role.kubernetes.io/worker: '' (4)
# ...
```

(1) The name of the OVN-Kubernetes secondary interface.

(2) The name of the network. This must match the value of the **spec.template.spec.domain.devices.interfaces.name** field.

(3) The name of the **NetworkAttachmentDefinition** object.

(4) Specifies the nodes on which the VM can be scheduled. The recommended node selector value is **node-role.kubernetes.io/worker: ''**.

2. Apply the **VirtualMachine** manifest:

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

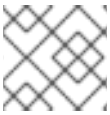
3. Optional: If you edited a running virtual machine, you must restart it for the changes to take effect.

11.10.3. Additional resources

- Creating secondary networks on OVN-Kubernetes
- About the Kubernetes NMState Operator
- Creating primary networks using a NetworkAttachmentDefinition

11.11. HOT PLUGGING SECONDARY NETWORK INTERFACES

You can add or remove secondary network interfaces without stopping your virtual machine (VM). OpenShift Virtualization supports hot plugging and hot unplugging for secondary interfaces that use bridge binding and the VirtIO device driver. OpenShift Virtualization also supports hot plugging secondary interfaces that use SR-IOV binding. To hot plug or hot unplug a secondary interface, you must have permission to create and list **VirtualMachineInstanceMigration** objects.

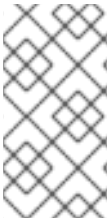


NOTE

Hot unplugging is not supported for Single Root I/O Virtualization (SR-IOV) interfaces.

11.11.1. VirtIO limitations

Each VirtIO interface uses one of the limited Peripheral Connect Interface (PCI) slots in the VM. There are a total of 32 slots available. The PCI slots are also used by other devices and must be reserved in advance, therefore slots might not be available on demand. OpenShift Virtualization reserves up to four slots for hot plugging interfaces. This includes any existing plugged network interfaces. For example, if your VM has two existing plugged interfaces, you can hot plug two more network interfaces.



NOTE

The actual number of slots available for hot plugging also depends on the machine type. For example, the default PCI topology for the q35 machine type supports hot plugging one additional PCIe device. For more information on PCI topology and hot plug support, see the libvirt documentation.

If you restart the VM after hot plugging an interface, that interface becomes part of the standard network interfaces.

11.11.2. Hot plugging a secondary network interface by using the CLI

Hot plug a secondary network interface to a virtual machine (VM) while the VM is running.

Prerequisites

- A network attachment definition is configured in the same namespace as your VM.
- The VM to which you want to hot plug the network interface is running.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Use your preferred text editor to edit the **VirtualMachine** manifest, as shown in the following example:

Example VM configuration

```
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. Specifies the name of the new network interface.
2. Specifies the name of the network. This must be the same as the **name** of the new network interface that you defined in the **template.spec.domain.devices.interfaces** list.
3. Specifies the name of the **NetworkAttachmentDefinition** object.

2. Save your changes and exit the editor.
3. For the new configuration to take effect, apply the changes by running the following command. Applying the changes triggers automatic VM live migration and attaches the network interface to the running VM.

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

where:

<filename>

Specifies the name of your **VirtualMachine** manifest YAML file.

Verification

1. Verify that the VM live migration is successful by using the following command:

```
$ oc get VirtualMachineInstanceMigration -w
```

Example output

```
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. Verify that the new interface is added to the VM by checking the status of the virtual machine instance (VMI):

```
$ oc get vmi vm-fedora -ojsonpath="{ @.status.interfaces }"
```

Example output

```
[
  {
    "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 The hot plugged interface appears in the VMI status.

11.11.3. Hot unplugging a secondary network interface by using the CLI

You can remove a secondary network interface from a running virtual machine (VM).

NOTE

Hot unplugging is not supported for Single Root I/O Virtualization (SR-IOV) interfaces.

Prerequisites

- Your VM must be running.
- The VM must be created on a cluster running OpenShift Virtualization 4.14 or later.
- The VM must have a bridge network interface attached.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Using your preferred text editor, edit the **VirtualMachine** manifest file and set the interface state to **absent**. Setting the interface state to **absent** detaches the network interface from the guest, but the interface still exists in the pod.

   Example VM configuration

```
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) Set the interface state to **absent** to detach it from the running VM. Removing the interface details from the VM specification does not hot unplug the secondary network interface.

2. Save your changes and exit the editor.

3. For the new configuration to take effect, apply the changes by running the following command. Applying the changes triggers automatic VM live migration and removes the interface from the pod.

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

   where:

   <filename>

   Specifies the name of your **VirtualMachine** manifest YAML file.

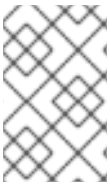
11.11.4. Additional resources

- Installing virtctl
- About live migration permissions

- Creating a Linux bridge network attachment definition
- Connecting a virtual machine to a Linux bridge network
- Creating an SR-IOV network attachment definition
- Connecting a virtual machine to an SR-IOV network

11.12. MANAGING THE LINK STATE OF A VIRTUAL MACHINE INTERFACE

You can manage the link state of a primary or secondary virtual machine (VM) interface by using the OpenShift Container Platform web console or the CLI. By specifying the link state, you can logically connect or disconnect the virtual network interface controller (vNIC) from a network.



NOTE

OpenShift Virtualization does not support link state management for Single Root I/O Virtualization (SR-IOV) secondary network interfaces and their link states are not reported.

You can specify the desired link state when you first create a VM, by editing the configuration of an existing VM that is stopped or running, or when you hot plug a new network interface to a running VM. If you edit a running VM, you do not need to restart or migrate the VM for the changes to be applied. The current link state of a VM interface is reported in the **status.interfaces.linkState** field of the **VirtualMachineInstance** manifest.

11.12.1. Setting the VM interface link state by using the web console

You can set the link state of a primary or secondary virtual machine (VM) network interface by using the web console.

Prerequisites

- You are logged into the OpenShift Container Platform web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines**.
2. Select a VM to view the **VirtualMachine details** page.
3. On the **Configuration** tab, click **Network**. A list of network interfaces is displayed.
4. Click the Options menu  of the interface that you want to edit.
5. Choose the appropriate option to set the interface link state:
   - If the current interface link state is **up**, select **Set link down**.
   - If the current interface link state is **down**, select **Set link up**.

11.12.2. Setting the VM interface link state by using the CLI

You can set the link state of a primary or secondary virtual machine (VM) network interface by using the CLI.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the VM configuration to set the interface link state, as in the following example:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: my-vm
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - name: default
              state: down
              masquerade: { }
      networks:
        - name: default
          pod: { }
# ...
```

1. The name of the interface.
2. The state of the interface. The possible values are:
   - **up**: Represents an active network connection. This is the default if no value is specified.
   - **down**: Represents a network interface link that is switched off.
   - **absent**: Represents a network interface that is hot unplugged.

IMPORTANT

If you have defined readiness or liveness probes to run VM health checks, setting the primary interface's link state to **down** causes the probes to fail. If a liveness probe fails, the VM is deleted and a new VM is created to restore responsiveness.

2. Apply the **VirtualMachine** manifest:

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

Verification

- Verify that the desired link state is set by checking the **status.interfaces.linkState** field of the **VirtualMachineInstance** manifest.

```
$ oc get vmi <vmi-name>
```

Example output

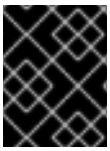
```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  name: my-vm
spec:
  domain:
    devices:
      interfaces:
      - name: default
        state: down
        masquerade: { }
  networks:
  - name: default
    pod: { }
status:
  interfaces:
    - name: default
      linkState: down
# ...
```

11.13. CONNECTING A VIRTUAL MACHINE TO A SERVICE MESH

OpenShift Virtualization is now integrated with OpenShift Service Mesh. You can monitor, visualize, and control traffic between pods that run virtual machine workloads on the default pod network with IPv4.

11.13.1. Adding a virtual machine to a service mesh

To add a virtual machine (VM) workload to a service mesh, enable automatic sidecar injection in the VM configuration file by setting the **sidecar.istio.io/inject** annotation to **true**. Then expose your VM as a service to view your application in the mesh.



IMPORTANT

To avoid port conflicts, do not use ports used by the Istio sidecar proxy. These include ports 15000, 15001, 15006, 15008, 15020, 15021, and 15090.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have installed the Service Mesh Operator.

Procedure

1. Edit the VM configuration file to add the **sidecar.istio.io/inject: "true"** annotation:

Example configuration file

```
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) The key/value pair (label) that must be matched to the service selector attribute.

(2) The annotation to enable automatic sidecar injection.

(3) The binding method (masquerade mode) for use with the default pod network.

2. Apply the VM configuration:

```
$ oc apply -f <vm_name>.yaml (1)
```

1 The name of the virtual machine YAML file.

3. Create a **Service** object to expose your VM to the service mesh.

```
apiVersion: v1
kind: Service
metadata:
  name: vm-istio
spec:
  selector:
    app: vm-istio 1
  ports:
    - port: 8080
      name: http
      protocol: TCP
```

1 The service selector that determines the set of pods targeted by a service. This attribute corresponds to the **spec.metadata.labels** field in the VM configuration file. In the above example, the **Service** object named **vm-istio** targets TCP port 8080 on any pod with the label **app=vm-istio**.

4. Create the service:

```
$ oc create -f <service_name>.yaml 1
```

1 The name of the service YAML file.

11.13.2. Additional resources

- Installing the Service Mesh Operator

11.14. CONFIGURING A DEDICATED NETWORK FOR LIVE MIGRATION

You can configure a dedicated Multus network for live migration. A dedicated network minimizes the effects of network saturation on tenant workloads during live migration.

11.14.1. Configuring a dedicated secondary network for live migration

To configure a dedicated secondary network for live migration, you must first create a bridge network attachment definition (NAD) by using the CLI. Then, you add the name of the **NetworkAttachmentDefinition** object to the **HyperConverged** custom resource (CR).

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You logged in to the cluster as a user with the **cluster-admin** role.
- Each node has at least two Network Interface Cards (NICs).
- The NICs for live migration are connected to the same VLAN.

Procedure

1. Create a **NetworkAttachmentDefinition** manifest according to the following example:

Example configuration file

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: my-secondary-network 1
  namespace: openshift-cnv
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "migration-bridge",
    "type": "macvlan",
    "master": "eth1", 2
    "mode": "bridge",
    "ipam": {
      "type": "whereabouts", 3
      "range": "10.200.5.0/24" 4
    }
  }'
```

1. Specify the name of the **NetworkAttachmentDefinition** object.
2. Specify the name of the NIC to be used for live migration.
3. Specify the name of the CNI plugin that provides the network for the NAD.
4. Specify an IP address range for the secondary network. This range must not overlap the IP addresses of the main network.

2. Open the **HyperConverged** CR in your default editor by running the following command:

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

3. Add the name of the **NetworkAttachmentDefinition** object to the **spec.liveMigrationConfig** stanza of the **HyperConverged** CR:

Example HyperConverged manifest

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  liveMigrationConfig:
    completionTimeoutPerGiB: 800
    network: <network> 1
    parallelMigrationsPerCluster: 5
```

```
    parallelOutboundMigrationsPerNode: 2
    progressTimeout: 150
# ...
```

1. Specify the name of the Multus **NetworkAttachmentDefinition** object to be used for live migrations.

4. Save your changes and exit the editor. The **virt-handler** pods restart and connect to the secondary network.

Verification

- When the node that the virtual machine runs on is placed into maintenance mode, the VM automatically migrates to another node in the cluster. You can verify that the migration occurred over the secondary network and not the default pod network by checking the target IP address in the virtual machine instance (VMI) metadata.

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

11.14.2. Selecting a dedicated network by using the web console

You can select a dedicated network for live migration by using the OpenShift Container Platform web console.

Prerequisites

- You configured a Multus network for live migration.
- You created a network attachment definition for the network.

Procedure

1. Navigate to **Virtualization > Overview** in the OpenShift Container Platform web console.
2. Click the **Settings** tab and then click **Live migration**.
3. Select the network from the **Live migration network** list.

11.14.3. Additional resources

- Configuring live migration limits and timeouts

11.15. CONFIGURING AND VIEWING IP ADDRESSES

You can configure an IP address when you create a virtual machine (VM). The IP address is provisioned with cloud-init.

You can view the IP address of a VM by using the OpenShift Container Platform web console or the command line. The network information is collected by the QEMU guest agent.

11.15.1. Configuring IP addresses for virtual machines

You can configure a static IP address when you create a virtual machine (VM) by using the web console or the command line.

You can configure a dynamic IP address when you create a VM by using the command line.

The IP address is provisioned with cloud-init.

11.15.1.1. Configuring an IP address when creating a virtual machine by using the CLI

You can configure a static or dynamic IP address when you create a virtual machine (VM). The IP address is provisioned with cloud-init.

NOTE

If the VM is connected to the pod network, the pod network interface is the default route unless you update it.

Prerequisites

- The virtual machine is connected to a secondary network.
- You have a DHCP server available on the secondary network to configure a dynamic IP for the virtual machine.

Procedure

- Edit the **spec.template.spec.volumes.cloudInitNoCloud.networkData** stanza of the virtual machine configuration:
  - To configure a dynamic IP address, specify the interface name and enable DHCP:

```
kind: VirtualMachine
spec:
# ...
  template:
  # ...
    spec:
      volumes:
      - cloudInitNoCloud:
          networkData: |
            version: 2
            ethernets:
              eth1: 1
                dhcp4: true
```

1 Specify the interface name.

  - To configure a static IP, specify the interface name and the IP address:

```
kind: VirtualMachine
spec:
# ...
  template:
  # ...
```

```
spec:
  volumes:
  - cloudInitNoCloud:
      networkData: |
        version: 2
        ethernets:
          eth1: 1
            addresses:
            - 10.10.10.14/24 2
```

1. Specify the interface name.
2. Specify the static IP address.

11.15.2. Viewing IP addresses of virtual machines

You can view the IP address of a VM by using the OpenShift Container Platform web console or the command line.

The network information is collected by the QEMU guest agent.

11.15.2.1. Viewing the IP address of a virtual machine by using the web console

You can view the IP address of a virtual machine (VM) by using the OpenShift Container Platform web console.

NOTE

You must install the QEMU guest agent on a VM to view the IP address of a secondary network interface. A pod network interface does not require the QEMU guest agent.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization → VirtualMachines** from the side menu.
2. Select a VM to open the **VirtualMachine details** page.
3. Click the **Details** tab to view the IP address.

11.15.2.2. Viewing the IP address of a virtual machine by using the CLI

You can view the IP address of a virtual machine (VM) by using the command line.

NOTE

You must install the QEMU guest agent on a VM to view the IP address of a secondary network interface. A pod network interface does not require the QEMU guest agent.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- Obtain the virtual machine instance configuration by running the following command:

```
$ oc describe vmi <vmi_name>
```

Example output

```
# ...
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
```

11.15.3. Additional resources

- Installing the QEMU guest agent

11.16. ACCESSING A VIRTUAL MACHINE BY USING ITS EXTERNAL FQDN

You can access a virtual machine (VM) that is attached to a secondary network interface from outside the cluster by using its fully qualified domain name (FQDN).

IMPORTANT

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

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

11.16.1. Configuring a DNS server for secondary networks

The Cluster Network Addons Operator (CNAO) deploys a Domain Name Server (DNS) server and monitoring components when you enable the **deployKubeSecondaryDNS** feature gate in the **HyperConverged** custom resource (CR).

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You configured a load balancer for the cluster.
- You logged in to the cluster with **cluster-admin** permissions.

Procedure

1. Edit the **HyperConverged** CR in your default editor by running the following command:

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

2. Enable the DNS server and monitoring components according to the following example:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
    featureGates:
      deployKubeSecondaryDNS: true (1)
# ...
```

(1) Enables the DNS server

3. Save the file and exit the editor.
4. Create a load balancer service to expose the DNS server outside the cluster by running the **oc expose** command according to the following example:

```
$ oc expose -n openshift-cnv deployment/secondary-dns --name=dns-lb \
  --type=LoadBalancer --port=53 --target-port=5353 --protocol='UDP'
```

5. Retrieve the external IP address by running the following command:

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

Example output

```
NAME      TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)         AGE
dns-lb    LoadBalancer   172.30.27.5    10.46.41.94     53:31829/TCP    5s
```

6. Edit the **HyperConverged** CR again:

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

7. Add the external IP address that you previously retrieved to the **kubeSecondaryDNSNameServerIP** field in the enterprise DNS server records. For example:

```
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) Specify the external IP address exposed by the load balancer service.

8. Save the file and exit the editor.

9. Retrieve the cluster FQDN by running the following command:

```
$ oc get dnses.config.openshift.io cluster -o jsonpath='{.spec.baseDomain}'
```

Example output

```
openshift.example.com
```

10. Point to the DNS server. To do so, add the **kubeSecondaryDNSNameServerIP** value and the cluster FQDN to the enterprise DNS server records. For example:

```
vm.<FQDN>. IN NS ns.vm.<FQDN>.
```

```
ns.vm.<FQDN>. IN A <kubeSecondaryDNSNameServerIP>
```

11.16.2. Connecting to a VM on a secondary network by using the cluster FQDN

You can access a running virtual machine (VM) attached to a secondary network interface by using the fully qualified domain name (FQDN) of the cluster.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You installed the QEMU guest agent on the VM.
- The IP address of the VM is public.
- You configured the DNS server for secondary networks.
- You retrieved the fully qualified domain name (FQDN) of the cluster.
  To obtain the FQDN, use the **oc get** command as follows:

```
$ oc get dnses.config.openshift.io cluster -o json | jq .spec.baseDomain
```

Procedure

1. Retrieve the network interface name from the VM configuration by running the following command:

```
$ oc get vm -n <namespace> <vm_name> -o yaml
```

Example output

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  runStrategy: Always
  template:
    spec:
      domain:
        devices:
          interfaces:
            - bridge: {}
              name: example-nic
# ...
      networks:
      - multus:
          networkName: bridge-conf
        name: example-nic 1
```

1 Note the name of the network interface.

2. Connect to the VM by using the **ssh** command:

```
$ ssh <user_name>@<interface_name>.<vm_name>.<namespace>.vm.<cluster_fqdn>
```

11.16.3. Additional resources

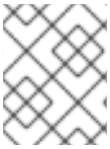
- Configuring ingress cluster traffic using a load balancer

- About MetalLB and the MetalLB Operator
- Configuring IP addresses for virtual machines

11.17. MANAGING MAC ADDRESS POOLS FOR NETWORK INTERFACES

The *KubeMacPool* component allocates MAC addresses for virtual machine (VM) network interfaces from a shared MAC address pool. This ensures that each network interface is assigned a unique MAC address.

A virtual machine instance created from that VM retains the assigned MAC address across reboots.



NOTE

KubeMacPool does not handle virtual machine instances created independently from a virtual machine.

11.17.1. Managing KubeMacPool by using the CLI

You can disable and re-enable KubeMacPool by using the command line.

KubeMacPool is enabled by default.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- To disable KubeMacPool in two namespaces, run the following command:

```
$ oc label namespace <namespace1> <namespace2>
mutatevirtualmachines.kubemacpool.io=ignore
```

- To re-enable KubeMacPool in two namespaces, run the following command:

```
$ oc label namespace <namespace1> <namespace2>
mutatevirtualmachines.kubemacpool.io-
```

CHAPTER 12. STORAGE

12.1. STORAGE CONFIGURATION OVERVIEW

You can configure a default storage class, storage profiles, Containerized Data Importer (CDI), data volumes, and automatic boot source updates.

12.1.1. Storage

The following storage configuration tasks are mandatory:

Configure a default storage class

You must configure a default storage class for your cluster. Otherwise, the cluster cannot receive automated boot source updates.

Configure storage profiles

You must configure storage profiles if your storage provider is not recognized by CDI. A storage profile provides recommended storage settings based on the associated storage class.

The following storage configuration tasks are optional:

Reserve additional PVC space for file system overhead

By default, 5.5% of a file system PVC is reserved for overhead, reducing the space available for VM disks by that amount. You can configure a different overhead value.

Configure local storage by using the hostpath provisioner

You can configure local storage for virtual machines by using the hostpath provisioner (HPP). When you install the OpenShift Virtualization Operator, the HPP Operator is automatically installed.

Configure user permissions to clone data volumes between namespaces

You can configure RBAC roles to enable users to clone data volumes between namespaces.

12.1.2. Containerized Data Importer

You can perform the following Containerized Data Importer (CDI) configuration tasks:

Override the resource request limits of a namespace

You can configure CDI to import, upload, and clone VM disks into namespaces that are subject to CPU and memory resource restrictions.

Configure CDI scratch space

CDI requires scratch space (temporary storage) to complete some operations, such as importing and uploading VM images. During this process, CDI provisions a scratch space PVC equal to the size of the PVC backing the destination data volume (DV).

12.1.3. Data volumes

You can perform the following data volume configuration tasks:

Enable preallocation for data volumes

CDI can preallocate disk space to improve write performance when creating data volumes. You can enable preallocation for specific data volumes.

Manage data volume annotations

Data volume annotations allow you to manage pod behavior. You can add one or more annotations to a data volume, which then propagates to the created importer pods.

12.1.4. Boot source updates

You can perform the following boot source update configuration task:

Manage automatic boot source updates

Boot sources can make virtual machine (VM) creation more accessible and efficient for users. If automatic boot source updates are enabled, CDI imports, polls, and updates the images so that they are ready to be cloned for new VMs. By default, CDI automatically updates Red Hat boot sources. You can enable automatic updates for custom boot sources.

12.2. CONFIGURING STORAGE PROFILES

A storage profile provides recommended storage settings based on the associated storage class. A storage profile is allocated for each storage class.

The Containerized Data Importer (CDI) recognizes a storage provider if it has been configured to identify and interact with the storage provider's capabilities.

For recognized storage types, the CDI provides values that optimize the creation of PVCs. You can also configure automatic settings for the storage class by customizing the storage profile. If the CDI does not recognize your storage provider, you must configure storage profiles.

IMPORTANT

When using OpenShift Virtualization with Red Hat OpenShift Data Foundation, specify RBD block mode persistent volume claims (PVCs) when creating virtual machine disks. RBD block mode volumes are more efficient and provide better performance than Ceph FS or RBD filesystem-mode PVCs.

To specify RBD block mode PVCs, use the 'ocs-storagecluster-ceph-rbd' storage class and **VolumeMode: Block**.

12.2.1. Customizing the storage profile

You can specify default parameters by editing the **StorageProfile** object for the provisioner's storage class. These default parameters only apply to the persistent volume claim (PVC) if they are not configured in the **DataVolume** object.

You cannot modify storage class parameters. To make changes, delete and re-create the storage class. You must then reapply any customizations that were previously made to the storage profile.

An empty **status** section in a storage profile indicates that a storage provisioner is not recognized by the Containerized Data Importer (CDI). Customizing a storage profile is necessary if you have a storage provisioner that is not recognized by CDI. In this case, the administrator sets appropriate values in the storage profile to ensure successful allocations.

If you are creating a snapshot of a VM, a warning appears if the storage class of the disk has more than one **VolumeSnapshotClass** associated with it. In this case, you must specify one volume snapshot class; otherwise, any disk that has more than one volume snapshot class is excluded from the snapshots list.

WARNING

If you create a data volume and omit YAML attributes and these attributes are not defined in the storage profile, then the requested storage will not be allocated and the underlying persistent volume claim (PVC) will not be created.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- Ensure that your planned configuration is supported by the storage class and its provider. Specifying an incompatible configuration in a storage profile causes volume provisioning to fail.

Procedure

1. Edit the storage profile. In this example, the provisioner is not recognized by CDI.

```
$ oc edit storageprofile <storage_class>
```

2. Specify the **accessModes** and **volumeMode** values you want to configure for the storage profile. For example:

Example storage profile

```
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) Specify the **accessModes**.

(2) Specify the **volumeMode**.

12.2.1.1. Specifying a volume snapshot class by using the web console

If you are creating a snapshot of a VM, a warning appears if the storage class of the disk has more than one volume snapshot class associated with it. In this case, you must specify one volume snapshot class; otherwise, any disk that has more than one volume snapshot class is excluded from the snapshots list.

You can specify the default volume snapshot class in the OpenShift Container Platform web console.

Procedure

1. From the **Virtualization** focused view, select **Storage**.
2. Click **VolumeSnapshotClasses**.
3. Select a volume snapshot class from the list.
4. Click the **Annotations** pencil icon.
5. Enter the following **Key**: **snapshot.storage.kubernetes.io/is-default-class**.
6. Enter the following **Value**: **true**.
7. Click **Save**.

12.2.1.2. Specifying a volume snapshot class by using the CLI

If you are creating a snapshot of a VM, a warning appears if the storage class of the disk has more than one volume snapshot class associated with it. In this case, you must specify one volume snapshot class; otherwise, any disk that has more than one volume snapshot class is excluded from the snapshots list.

You can select which volume snapshot class to use by either:

- Setting the **spec.snapshotClass** for the storage profile.
- Setting a default volume snapshot class.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- Set the **VolumeSnapshotClass** you want to use. For example:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: ocs-storagecluster-ceph-rbd-virtualization
spec:
  snapshotClass: ocs-storagecluster-rbdplugin-snapclass
```

- Alternatively, set the default volume snapshot class by running the following command:

```
# oc patch VolumeSnapshotClass ocs-storagecluster-cephfsplugin-snapclass --type=merge -
p '{"metadata":{"annotations":{"snapshot.storage.kubernetes.io/is-default-class":"true"}}}'
```

12.2.1.3. Viewing automatically created storage profiles

The system creates storage profiles for each storage class automatically.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. To view the list of storage profiles, run the following command:

```
$ oc get storageprofile
```

2. To fetch the details of a particular storage profile, run the following command:

```
$ oc describe storageprofile <name>
```

Example storage profile details

```
Name:         ocs-storagecluster-ceph-rbd-virtualization
Namespace:
Labels:       app=containerized-data-importer
              app.kubernetes.io/component=storage
              app.kubernetes.io/managed-by=cdi-controller
              app.kubernetes.io/part-of=hyperconverged-cluster
              app.kubernetes.io/version=4.17.2
              cdi.kubevirt.io=
Annotations:  <none>
API Version:  cdi.kubevirt.io/v1beta1
Kind:         StorageProfile
Metadata:
  Creation Timestamp:  2023-11-13T07:58:02Z
  Generation:          2
  Owner References:
    API Version:           cdi.kubevirt.io/v1beta1
    Block Owner Deletion:  true
    Controller:            true
    Kind:                  CDI
    Name:                  cdi-kubevirt-hyperconverged
    UID:                   2d6f169a-382c-4caf-b614-a640f2ef8abb
  Resource Version:        4186799537
  UID:                     14aef804-6688-4f2e-986b-0297fd3aaa68
Spec:
Status:
  Claim Property Sets:  (1)
    accessModes:
      ReadWriteMany
    volumeMode:  Block
    accessModes:
      ReadWriteOnce
    volumeMode:  Block
    accessModes:
      ReadWriteOnce
    volumeMode:                 Filesystem
  Clone Strategy:               csi-clone  (2)
  Data Import Cron Source Format:  snapshot  (3)
  Provisioner:                  openshift-storage.rbd.csi.ceph.com
```

```
  Snapshot Class:              ocs-storagecluster-rbdplugin-snapclass
  Storage Class:               ocs-storagecluster-ceph-rbd-virtualization
Events:                        <none>
```

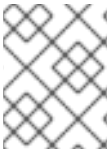
1. **Claim Property Sets** is an ordered list of **AccessMode**/**VolumeMode** pairs, which describe the PVC modes that are used to provision VM disks.
2. The **Clone Strategy** line indicates the clone strategy to be used.
3. **Data Import Cron Source Format** indicates whether golden images on this storage are stored as PVCs or volume snapshots.

12.2.1.4. Setting a default cloning strategy by using a storage profile

You can use storage profiles to set a default cloning method for a storage class by creating a cloning strategy. Setting cloning strategies can be helpful, for example, if your storage vendor supports only certain cloning methods. It also allows you to select a method that limits resource usage or maximizes performance.

Cloning strategies are specified by setting the **cloneStrategy** attribute in a storage profile to one of the following values:

- **snapshot** is used by default when snapshots are configured. The Containerized Data Importer (CDI) will use the snapshot method if it recognizes the storage provider and the provider supports Container Storage Interface (CSI) snapshots. This cloning strategy uses a temporary volume snapshot to clone the volume.
- **copy** uses a source pod and a target pod to copy data from the source volume to the target volume. Host-assisted cloning is the least efficient method of cloning.
- **csi-clone** uses the CSI clone API to efficiently clone an existing volume without using an interim volume snapshot. Unlike **snapshot** or **copy**, which are used by default if no storage profile is defined, CSI volume cloning is only used when you specify it in the **StorageProfile** object for the provisioner's storage class.



NOTE

You can set clone strategies using the CLI without modifying the default **claimPropertySets** in your YAML **spec** section.

Example storage profile

```
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. Specify the **accessModes**.
2. Specify the **volumeMode**.
3. Specify the default **cloneStrategy**.

12.3. MANAGING AUTOMATIC BOOT SOURCE UPDATES

You can manage automatic updates for the following boot sources:

- All Red Hat boot sources
- All custom boot sources
- Individual Red Hat or custom boot sources

Boot sources can make virtual machine (VM) creation more accessible and efficient for users. If automatic boot source updates are enabled, the Containerized Data Importer (CDI) imports, polls, and updates the images so that they are ready to be cloned for new VMs. By default, CDI automatically updates Red Hat boot sources.

12.3.1. Managing Red Hat boot source updates

You can opt out of automatic updates for all system-defined boot sources by setting the **enableCommonBootImageImport** field value to **false**. If you set the value to **false**, all **DataImportCron** objects are deleted. This does not, however, remove previously imported boot source objects that store operating system images, though administrators can delete them manually.

When the **enableCommonBootImageImport** field value is set to **false**, **DataSource** objects are reset so that they no longer point to the original boot source. An administrator can manually provide a boot source by creating a new persistent volume claim (PVC) or volume snapshot for the **DataSource** object, and then populating it with an operating system image.

12.3.1.1. Managing automatic updates for all system-defined boot sources

Disabling automatic boot source imports and updates can lower resource usage. In disconnected environments, disabling automatic boot source updates prevents **CDIDataImportCronOutdated** alerts from filling up logs.

To disable automatic updates for all system-defined boot sources, set the **enableCommonBootImageImport** field value to **false**. Setting this value to **true** turns automatic updates back on.

NOTE

Custom boot sources are not affected by this setting.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- Enable or disable automatic boot source updates by editing the **HyperConverged** custom resource (CR).
  - To disable automatic boot source updates, set the **spec.enableCommonBootImageImport** field value in the **HyperConverged** CR to **false**. For example:

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

  - To re-enable automatic boot source updates, set the **spec.enableCommonBootImageImport** field value in the **HyperConverged** CR to **true**. For example:

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

12.3.2. Managing custom boot source updates

Custom boot sources that are not provided by OpenShift Virtualization are not controlled by the feature gate. You must manage them individually by editing the **HyperConverged** custom resource (CR).

IMPORTANT

You must configure a storage class. Otherwise, the cluster cannot receive automated updates for custom boot sources. See Defining a storage class for details.

12.3.2.1. Configuring the default and virt-default storage classes

A storage class determines how persistent storage is provisioned for workloads. In OpenShift Virtualization, the virt-default storage class takes precedence over the cluster default storage class and is used specifically for virtualization workloads. Only one storage class should be set as virt-default or cluster default at a time. If multiple storage classes are marked as default, the virt-default storage class overrides the cluster default. To ensure consistent behavior, configure only one storage class as the default for virtualization workloads.

IMPORTANT

Boot sources are created using the default storage class. When the default storage class changes, old boot sources are automatically updated using the new default storage class. If your cluster does not have a default storage class, you must define one.

If boot source images were stored as volume snapshots and both the cluster default and virt-default storage class have been unset, the volume snapshots are cleaned up and new data volumes will be created. However the newly created data volumes will not start importing until a default storage class is set.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Patch the current virt-default or a cluster default storage class to false:
   a. Identify all storage classes currently marked as virt-default by running the following command:

```
$ oc get sc -o json| jq '.items[].metadata|select(.annotations."storageclass.kubevirt.io/is-default-virt-class"=="true")|.name'
```

   b. For each storage class returned, remove the virt-default annotation by running the following command:

```
$ oc patch storageclass <storage_class_name> -p '{"metadata": {"annotations": {"storageclass.kubevirt.io/is-default-virt-class": "false"}}}'
```

   c. Identify all storage classes currently marked as cluster default by running the following command:

```
$ oc get sc -o json| jq '.items[].metadata|select(.annotations."storageclass.kubernetes.io/is-default-class"=="true")|.name'
```

   d. For each storage class returned, remove the cluster default annotation by running the following command:

```
$ oc patch storageclass <storage_class_name> -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

2. Set a new default storage class:
   a. Assign the virt-default role to a storage class by running the following command:

```
$ oc patch storageclass <storage_class_name> -p '{"metadata": {"annotations": {"storageclass.kubevirt.io/is-default-virt-class": "true"}}}'
```

   b. Alternatively, assign the cluster default role to a storage class by running the following command:

```
$ oc patch storageclass <storage_class_name> -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

12.3.2.2. Configuring a storage class for boot source images

You can configure a specific storage class in the **HyperConverged** resource.

IMPORTANT

To ensure stable behavior and avoid unnecessary re-importing, you can specify the **storageClassName** in the **dataImportCronTemplates** section of the **HyperConverged** resource.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Open the **HyperConverged** CR in your default editor by running the following command:

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

2. Add the **dataImportCronTemplate** to the spec section of the **HyperConverged** resource and set the **storageClassName**:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      name: rhel9-image-cron
    spec:
      template:
        spec:
          storage:
            storageClassName: <storage_class> 1
      schedule: "0 */12 * * *" 2
      managedDataSource: <data_source> 3
# ...
```

1. Define the storage class.
2. Required: Schedule for the job specified in cron format.
3. Required: The data source to use.

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. Wait for the HyperConverged Operator (HCO) and Scheduling, Scale, and Performance (SSP) resources to complete reconciliation.

4. Delete any outdated **DataVolume** and **VolumeSnapshot** objects from the **openshift-virtualization-os-images** namespace by running the following command.

```
$ oc delete DataVolume,VolumeSnapshot -n openshift-virtualization-os-images --
selector=cdi.kubevirt.io/dataImportCron
```

5. Wait for all **DataSource** objects to reach a "Ready - True" status. Data sources can reference either a PersistentVolumeClaim (PVC) or a VolumeSnapshot. To check the expected source format, run the following command:

```
$ oc get storageprofile <storage_class_name> -o json | jq
.status.dataImportCronSourceFormat
```

12.3.2.3. Enabling automatic updates for custom boot sources

OpenShift Virtualization automatically updates system-defined boot sources by default, but does not automatically update custom boot sources. You must manually enable automatic updates by editing the **HyperConverged** custom resource (CR).

Prerequisites

- The cluster has a default storage class.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Open the **HyperConverged** CR in your default editor by running the following command:

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

2. Edit the **HyperConverged** CR, adding the appropriate template and boot source in the **dataImportCronTemplates** section. For example:

Example custom resource

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      name: centos-stream9-image-cron
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true" 1
    spec:
      schedule: "0 */12 * * *" 2
      template:
        spec:
          source:
            registry: 3
              url: docker://quay.io/containerdisks/centos-stream:9
          storage:
            resources:
              requests:
```

```
            storage: 30Gi
    garbageCollect: Outdated
    managedDataSource: centos-stream9 (4)
```

(1) This annotation is required for storage classes with **volumeBindingMode** set to **WaitForFirstConsumer**.

(2) Schedule for the job specified in cron format.

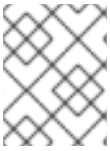
(3) Use to create a data volume from a registry source. Use the default **pod pullMethod** and not **node pullMethod**, which is based on the **node** docker cache. The **node** docker cache is useful when a registry image is available via **Container.Image**, but the CDI importer is not authorized to access it.

(4) For the custom image to be detected as an available boot source, the name of the image's **managedDataSource** must match the name of the template's **DataSource**, which is found under **spec.dataVolumeTemplates.spec.sourceRef.name** in the VM template YAML file.

3. Save the file.

12.3.2.4. Enabling volume snapshot boot sources

Enable volume snapshot boot sources by setting the parameter in the **StorageProfile** associated with the storage class that stores operating system base images. Although **DataImportCron** was originally designed to maintain only PVC sources, **VolumeSnapshot** sources scale better than PVC sources for certain storage types.



NOTE

Use volume snapshots on a storage profile that is proven to scale better when cloning from a single snapshot.

Prerequisites

- You must have access to a volume snapshot with the operating system image.
- The storage must support snapshotting.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Open the storage profile object that corresponds to the storage class used to provision boot sources by running the following command:

```
$ oc edit storageprofile <storage_class>
```

2. Review the **dataImportCronSourceFormat** specification of the **StorageProfile** to confirm whether or not the VM is using PVC or volume snapshot by default.
3. Edit the storage profile, if needed, by updating the **dataImportCronSourceFormat** specification to **snapshot**.

Example storage profile

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
# ...
spec:
  dataImportCronSourceFormat: snapshot
```

Verification

1. Open the storage profile object that corresponds to the storage class used to provision boot sources.

```
$ oc get storageprofile <storage_class>  -oyaml
```

2. Confirm that the **dataImportCronSourceFormat** specification of the **StorageProfile** is set to 'snapshot', and that any **DataSource** objects that the **DataImportCron** points to now reference volume snapshots.

You can now use these boot sources to create virtual machines.

12.3.3. Disabling automatic updates for a single boot source

You can disable automatic updates for an individual boot source, whether it is custom or system-defined, by editing the **HyperConverged** custom resource (CR).

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Open the **HyperConverged** CR in your default editor by running the following command:

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

2. Disable automatic updates for an individual boot source by editing the **spec.dataImportCronTemplates** field.

   Custom boot source

   - Remove the boot source from the **spec.dataImportCronTemplates** field. Automatic updates are disabled for custom boot sources by default.

   System-defined boot source

   a. Add the boot source to **spec.dataImportCronTemplates**.

NOTE

Automatic updates are enabled by default for system-defined boot sources, but these boot sources are not listed in the CR unless you add them.

b. Set the value of the **dataimportcrontemplate.kubevirt.io/enable** annotation to **'false'**. For example:

```
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. Save the file.

12.3.4. Verifying the status of a boot source

You can determine if a boot source is system-defined or custom by viewing the **HyperConverged** custom resource (CR).

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. View the contents of the **HyperConverged** CR by running the following command:

```
$ oc get hyperconverged kubevirt-hyperconverged -n openshift-cnv -o yaml
```

Example output

```
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-9-image-cron
```

```
    spec:
      garbageCollect: Outdated
      managedDataSource: centos-stream9
      schedule: 55 8/12 * * *
      template:
        metadata: {}
        spec:
          source:
            registry:
              url: docker://quay.io/containerdisks/centos-stream:9
          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-stream9
      schedule: 55 8/12 * * *
      template:
        metadata: {}
        spec:
          source:
            registry:
              pullMethod: node
              url: docker://quay.io/containerdisks/centos-stream:9
          storage:
            resources:
              requests:
                storage: 30Gi
        status: {}
    status: {} 2
# ...
```

1 Indicates a system-defined boot source.

2 Indicates a custom boot source.

2. Verify the status of the boot source by reviewing the **status.dataImportCronTemplates.status** field.

   - If the field contains **commonTemplate: true**, it is a system-defined boot source.
   - If the **status.dataImportCronTemplates.status** field has the value **{}**, it is a custom boot source.

12.4. RESERVING PVC SPACE FOR FILE SYSTEM OVERHEAD

When you add a virtual machine disk to a persistent volume claim (PVC) that uses the **Filesystem** volume mode, you must ensure that there is enough space on the PVC for the VM disk and for file system overhead, such as metadata.

By default, OpenShift Virtualization reserves 5.5% of the PVC space for overhead, reducing the space available for virtual machine disks by that amount.

You can configure a different overhead value by editing the **HCO** object. You can change the value globally and you can specify values for specific storage classes.

12.4.1. Overriding the default file system overhead value

Change the amount of persistent volume claim (PVC) space that the OpenShift Virtualization reserves for file system overhead by editing the **spec.filesystemOverhead** attribute of the **HCO** object.

Prerequisites

- Install the OpenShift CLI (**oc**).

Procedure

1. Open the **HCO** object for editing by running the following command:

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

2. Edit the **spec.filesystemOverhead** fields, populating them with your chosen values:

```
# ...
spec:
  filesystemOverhead:
    global: "<new_global_value>" 1
    storageClass:
      <storage_class_name>: "<new_value_for_this_storage_class>" 2
```

1. The default file system overhead percentage used for any storage classes that do not already have a set value. For example, **global: "0.07"** reserves 7% of the PVC for file system overhead.

2. The file system overhead percentage for the specified storage class. For example, **mystorageclass: "0.04"** changes the default overhead value for PVCs in the **mystorageclass** storage class to 4%.

3. Save and exit the editor to update the **HCO** object.

Verification

- View the **CDIConfig** status and verify your changes by running one of the following commands:
  To generally verify changes to **CDIConfig**:

```
$ oc get cdiconfig -o yaml
```

  To view your specific changes to **CDIConfig**:

```
$ oc get cdiconfig -o jsonpath='{.items..status.filesystemOverhead}'
```

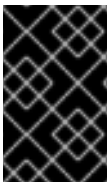
12.5. CONFIGURING LOCAL STORAGE BY USING THE HOSTPATH PROVISIONER

You can configure local storage for virtual machines by using the hostpath provisioner (HPP).

When you install the OpenShift Virtualization Operator, the Hostpath Provisioner Operator is automatically installed. HPP is a local storage provisioner designed for OpenShift Virtualization that is created by the Hostpath Provisioner Operator. To use HPP, you create an HPP custom resource (CR) with a basic storage pool.

12.5.1. Creating a hostpath provisioner with a basic storage pool

You configure a hostpath provisioner (HPP) with a basic storage pool by creating an HPP custom resource (CR) with a **storagePools** stanza. The storage pool specifies the name and path used by the CSI driver.



IMPORTANT

Do not create storage pools in the same partition as the operating system. Otherwise, the operating system partition might become filled to capacity, which will impact performance or cause the node to become unstable or unusable.

Prerequisites

- The directories specified in **spec.storagePools.path** must have read/write access.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create an **hpp_cr.yaml** file with a **storagePools** stanza as in the following example:

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
  storagePools:
  - name: any_name 1
    path: "/var/myvolumes" 2
  workload:
    nodeSelector:
      kubernetes.io/os: linux
```

1. Specifies the name to identify the source to use. It must be the same as the **storagePools** name in the **StorageClass.yaml**. For example, **local**.
2. Specifies the storage pool directories under this node path. Ensure that the path **/var/myvolumes** has been created on each worker node.

2. Save the file and exit.

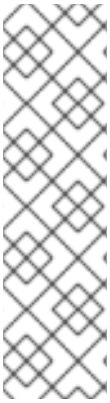
3. Create the HPP by running the following command:

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

12.5.1.1. About creating storage classes

When you create a storage class, you set parameters that affect the dynamic provisioning of persistent volumes (PVs) that belong to that storage class. You cannot update a **StorageClass** object's parameters after you create it.

In order to use the hostpath provisioner (HPP) you must create an associated storage class for the CSI driver with the **storagePools** stanza.



NOTE

Virtual machines use data volumes that are based on local PVs. Local PVs are bound to specific nodes. While the disk image is prepared for consumption by the virtual machine, it is possible that the virtual machine cannot be scheduled to the node where the local storage PV was previously pinned.

To solve this problem, use the Kubernetes pod scheduler to bind the persistent volume claim (PVC) to a PV on the correct node. By using the **StorageClass** value with **volumeBindingMode** parameter set to **WaitForFirstConsumer**, the binding and provisioning of the PV is delayed until a pod is created using the PVC.

12.5.1.2. Creating a storage class for the CSI driver with the storagePools stanza

To use the hostpath provisioner (HPP) you must create an associated storage class for the Container Storage Interface (CSI) driver.

When you create a storage class, you set parameters that affect the dynamic provisioning of persistent volumes (PVs) that belong to that storage class. You cannot update a **StorageClass** object's parameters after you create it.

NOTE

Virtual machines use data volumes that are based on local PVs. Local PVs are bound to specific nodes. While a disk image is prepared for consumption by the virtual machine, it is possible that the virtual machine cannot be scheduled to the node where the local storage PV was previously pinned.

To solve this problem, use the Kubernetes pod scheduler to bind the persistent volume claim (PVC) to a PV on the correct node. By using the **StorageClass** value with **volumeBindingMode** parameter set to **WaitForFirstConsumer**, the binding and provisioning of the PV is delayed until a pod is created using the PVC.

Procedure

1. Create a **storageclass_csi.yaml** file to define the storage class:

```
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. The two possible **reclaimPolicy** values are **Delete** and **Retain**. If you do not specify a value, the default value is **Delete**.
2. The **volumeBindingMode** parameter determines when dynamic provisioning and volume binding occur. Specify **WaitForFirstConsumer** to delay the binding and provisioning of a persistent volume (PV) until after a pod that uses the persistent volume claim (PVC) is created. This ensures that the PV meets the pod's scheduling requirements.
3. Specify the name of the storage pool defined in the HPP CR.

2. Save the file and exit.
3. Create the **StorageClass** object by running the following command:

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

12.5.2. About storage pools created with PVC templates

If you have a single, large persistent volume (PV), you can create a storage pool by defining a PVC template in the hostpath provisioner (HPP) custom resource (CR).

A storage pool created with a PVC template can contain multiple HPP volumes. Splitting a PV into smaller volumes provides greater flexibility for data allocation.

The PVC template is based on the **spec** stanza of the **PersistentVolumeClaim** object:

Example PersistentVolumeClaim object

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: iso-pvc
spec:
  volumeMode: Block 1
  storageClassName: my-storage-class
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

1. This value is only required for block volume mode PVs.

You define a storage pool using a **pvcTemplate** specification in the HPP CR. The Operator creates a

PVC from the **pvcTemplate** specification for each node containing the HPP CSI driver. The PVC created from the PVC template consumes the single large PV, allowing the HPP to create smaller dynamic volumes.

You can combine basic storage pools with storage pools created from PVC templates.

12.5.2.1. Creating a storage pool with a PVC template

You can create a storage pool for multiple hostpath provisioner (HPP) volumes by specifying a PVC template in the HPP custom resource (CR).

IMPORTANT

Do not create storage pools in the same partition as the operating system. Otherwise, the operating system partition might become filled to capacity, which will impact performance or cause the node to become unstable or unusable.

Prerequisites

- The directories specified in **spec.storagePools.path** must have read/write access.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create an **hpp_pvc_template_pool.yaml** file for the HPP CR that specifies a persistent volume (PVC) template in the **storagePools** stanza according to the following example:

```
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) The **storagePools** stanza is an array that can contain both basic and PVC template storage pools.

(2) Specify the storage pool directories under this node path.

3. Optional: The **volumeMode** parameter can be either **Block** or **Filesystem** as long as it matches the provisioned volume format. If no value is specified, the default is **Filesystem**.

4. If the **storageClassName** parameter is omitted, the default storage class is used to create PVCs. If you omit **storageClassName**, ensure that the HPP storage class is not the default storage class.

5. You can specify statically or dynamically provisioned storage. In either case, ensure the requested storage size is appropriate for the volume you want to virtually divide or the PVC cannot be bound to the large PV. If the storage class you are using uses dynamically provisioned storage, pick an allocation size that matches the size of a typical request.

2. Save the file and exit.

3. Create the HPP with a storage pool by running the following command:

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

12.6. ENABLING USER PERMISSIONS TO CLONE DATA VOLUMES ACROSS NAMESPACES

The isolating nature of namespaces means that users cannot by default clone resources between namespaces.

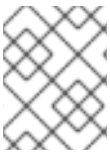
To enable a user to clone a virtual machine to another namespace, a user with the **cluster-admin** role must create a new cluster role. Bind this cluster role to a user to enable them to clone virtual machines to the destination namespace.

12.6.1. Creating RBAC resources for cloning data volumes

Create a new cluster role that enables permissions for all actions for the **datavolumes** resource.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You must have cluster admin privileges.



NOTE

If you are a non-admin user that is an administrator for both the source and target namespaces, you can create a **Role** instead of a **ClusterRole** where appropriate.

Procedure

1. Create a **ClusterRole** manifest:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: <datavolume-cloner> 1
rules:
```

```
- apiGroups: ["cdi.kubevirt.io"]
  resources: ["datavolumes/source"]
  verbs: ["*"]
```

1. Unique name for the cluster role.

2. Create the cluster role in the cluster:

```
$ oc create -f <datavolume-cloner.yaml> 1
```

1. The file name of the **ClusterRole** manifest created in the previous step.

3. Create a **RoleBinding** manifest that applies to both the source and destination namespaces and references the cluster role created in the previous step.

```
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. Unique name for the role binding.
2. The namespace for the source data volume.
3. The namespace to which the data volume is cloned.
4. The name of the cluster role created in the previous step.

4. Create the role binding in the cluster:

```
$ oc create -f <datavolume-cloner.yaml> 1
```

1. The file name of the **RoleBinding** manifest created in the previous step.

12.7. CONFIGURING CDI TO OVERRIDE CPU AND MEMORY QUOTAS

You can configure the Containerized Data Importer (CDI) to import, upload, and clone virtual machine disks into namespaces that are subject to CPU and memory resource restrictions.

12.7.1. About CPU and memory quotas in a namespace

A *resource quota*, defined by the **ResourceQuota** object, imposes restrictions on a namespace that limit the total amount of compute resources that can be consumed by resources within that namespace.

The **HyperConverged** custom resource (CR) defines the user configuration for the Containerized Data Importer (CDI). The CPU and memory request and limit values are set to a default value of **0**. This ensures that pods created by CDI that do not specify compute resource requirements are given the default values and are allowed to run in a namespace that is restricted with a quota.

12.7.2. Overriding CPU and memory defaults

Modify the default settings for CPU and memory requests and limits for your use case by adding the **spec.resourceRequirements.storageWorkloads** stanza to the **HyperConverged** custom resource (CR).

Prerequisites

- Install the OpenShift CLI (**oc**).

Procedure

1. Edit the **HyperConverged** CR by running the following command:

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

2. Add the **spec.resourceRequirements.storageWorkloads** stanza to the CR, setting the values based on your use case. For example:

```
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. Save and exit the editor to update the **HyperConverged** CR.

12.7.3. Additional resources

- Resource quotas per project

12.8. PREPARING CDI SCRATCH SPACE

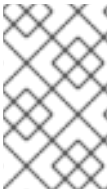
12.8.1. About scratch space

The Containerized Data Importer (CDI) requires scratch space (temporary storage) to complete some operations, such as importing and uploading virtual machine images. During this process, CDI provisions

a scratch space PVC equal to the size of the PVC backing the destination data volume (DV). The scratch space PVC is deleted after the operation completes or aborts.

You can define the storage class that is used to bind the scratch space PVC in the **spec.scratchSpaceStorageClass** field of the **HyperConverged** custom resource.

If the defined storage class does not match a storage class in the cluster, then the default storage class defined for the cluster is used. If there is no default storage class defined in the cluster, the storage class used to provision the original DV or PVC is used.



NOTE

CDI requires requesting scratch space with a **file** volume mode, regardless of the PVC backing the origin data volume. If the origin PVC is backed by **block** volume mode, you must define a storage class capable of provisioning **file** volume mode PVCs.

Manual provisioning
If there are no storage classes, CDI uses any PVCs in the project that match the size requirements for the image. If there are no PVCs that match these requirements, the CDI import pod remains in a **Pending** state until an appropriate PVC is made available or until a timeout function kills the pod.

12.8.2. CDI operations that require scratch space

Type	Reason
Registry imports	CDI must download the image to a scratch space and extract the layers to find the image file. The image file is then passed to QEMU-IMG for conversion to a raw disk.
Upload image	QEMU-IMG does not accept input from STDIN. Instead, the image to upload is saved in scratch space before it can be passed to QEMU-IMG for conversion.
HTTP imports of archived images	QEMU-IMG does not know how to handle the archive formats CDI supports. Instead, the image is unarchived and saved into scratch space before it is passed to QEMU-IMG.
HTTP imports of authenticated images	QEMU-IMG inadequately handles authentication. Instead, the image is saved to scratch space and authenticated before it is passed to QEMU-IMG.
HTTP imports of custom certificates	QEMU-IMG inadequately handles custom certificates of HTTPS endpoints. Instead, CDI downloads the image to scratch space before passing the file to QEMU-IMG.

12.8.3. Defining a storage class

You can define the storage class that the Containerized Data Importer (CDI) uses when allocating scratch space by adding the **spec.scratchSpaceStorageClass** field to the **HyperConverged** custom resource (CR).

Prerequisites

- Install the OpenShift CLI (**oc**).

Procedure

1. Edit the **HyperConverged** CR by running the following command:

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

2. Add the **spec.scratchSpaceStorageClass** field to the CR, setting the value to the name of a storage class that exists in the cluster:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  scratchSpaceStorageClass: "<storage_class>" 1
```

1 If you do not specify a storage class, CDI uses the storage class of the persistent volume claim that is being populated.

3. Save and exit your default editor to update the **HyperConverged** CR.

12.8.4. CDI supported operations matrix

This matrix shows the supported CDI operations for content types against endpoints, and which of these operations requires scratch space.

Content types	HTTP	HTTPS	HTTP basic auth	Registry	Upload
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*

✓ Supported operation

☐ Unsupported operation

* Requires scratch space

** Requires scratch space if a custom certificate authority is required

12.8.5. Additional resources

- Dynamic provisioning

12.9. USING PREALLOCATION FOR DATA VOLUMES

The Containerized Data Importer can preallocate disk space to improve write performance when creating data volumes.

You can enable preallocation for specific data volumes.

12.9.1. About preallocation

The Containerized Data Importer (CDI) can use the QEMU preallocate mode for data volumes to improve write performance. You can use preallocation mode for importing and uploading operations and when creating blank data volumes.

If preallocation is enabled, CDI uses the better preallocation method depending on the underlying file system and device type:

fallocate

If the file system supports it, CDI uses the operating system's **fallocate** call to preallocate space by using the **posix_fallocate** function, which allocates blocks and marks them as uninitialized.

full

If **fallocate** mode cannot be used, **full** mode allocates space for the image by writing data to the underlying storage. Depending on the storage location, all the empty allocated space might be zeroed.

12.9.2. Enabling preallocation for a data volume

You can enable preallocation for specific data volumes by including the **spec.preallocation** field in the data volume manifest. You can enable preallocation mode in either the web console or by using the OpenShift CLI (**oc**).

Preallocation mode is supported for all CDI source types.

Procedure

- Specify the **spec.preallocation** field in the data volume manifest:

```
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. All CDI source types support preallocation. However, preallocation is ignored for cloning operations.
2. Specify the URL of the data source in your registry.

12.10. MANAGING DATA VOLUME ANNOTATIONS

Data volume (DV) annotations allow you to manage pod behavior. You can add one or more annotations to a data volume, which then propagates to the created importer pods.

12.10.1. Example: Data volume annotations

This example shows how you can configure data volume (DV) annotations to control which network the importer pod uses. The **v1.multus-cni.io/default-network: bridge-network** annotation causes the pod to use the multus network named **bridge-network** as its default network. If you want the importer pod to use both the default network from the cluster and the secondary multus network, use the **k8s.v1.cni.cncf.io/networks: <network_name>** annotation.

Multus network annotation example

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: datavolume-example
  annotations:
    v1.multus-cni.io/default-network: bridge-network (1)
# ...
```

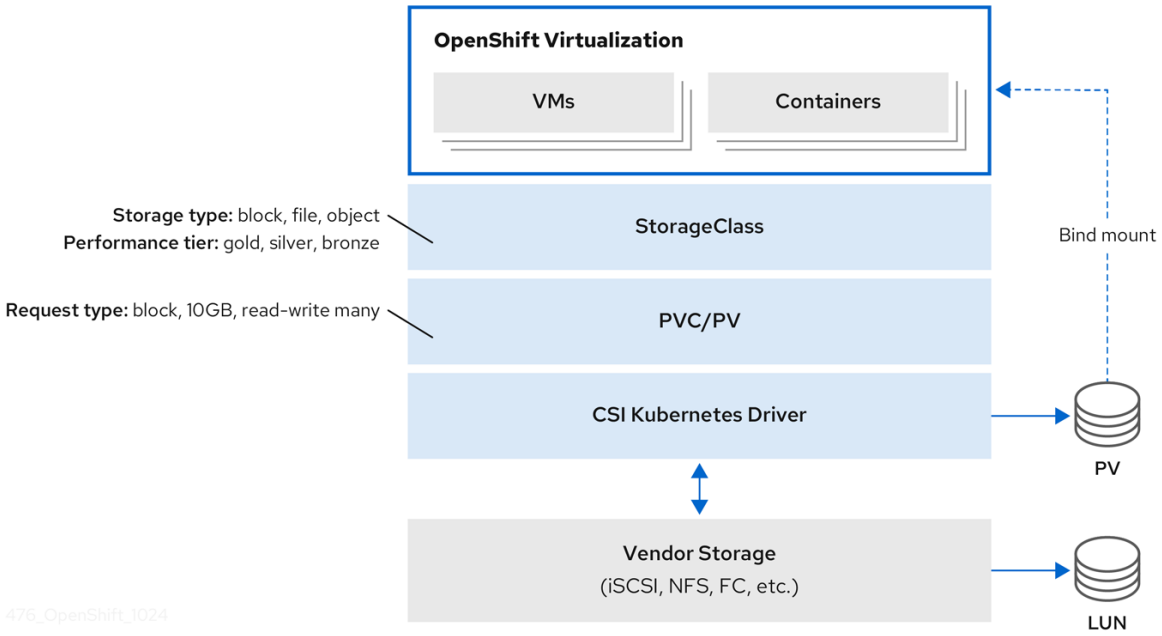
1. Multus network annotation

12.11. UNDERSTANDING VIRTUAL MACHINE STORAGE WITH THE CSI PARADIGM

Virtual machines (VMs) in OpenShift Virtualization use PersistentVolume (PV) and PersistentVolumeClaim (PVC) paradigms to manage storage. This ensures seamless integration with the Container Storage Interface (CSI).

12.11.1. Virtual machine CSI storage overview

OpenShift Virtualization integrates with the Container Storage Interface (CSI) to manage VM storage. Storage classes define storage capabilities such as performance tiers and types. PersistentVolumeClaims (PVCs) request storage resources, which bind to PersistentVolumes (PVs). CSI drivers connect Kubernetes to vendor storage backends, including iSCSI, NFS, and Fibre Channel.


OpenShift Virtualization
VMs
Containers
Storage type: block, file, object
Performance tier: gold, silver, bronze
StorageClass
Request type: block, 10GB, read-write many
PVC/PV
CSI Kubernetes Driver
Bind mount
PV
Vendor Storage
(iSCSI, NFS, FC, etc.)
LUN
476_OpenShift_1024

CHAPTER 13. LIVE MIGRATION

13.1. ABOUT LIVE MIGRATION

Live migration is the process of moving a running virtual machine (VM) to another node in the cluster without interrupting the virtual workload. Live migration enables smooth transitions during cluster upgrades or any time a node needs to be drained for maintenance or configuration changes.

By default, live migration traffic is encrypted using Transport Layer Security (TLS).

13.1.1. Live migration requirements

Live migration has the following requirements:

- The cluster must have shared storage with **ReadWriteMany** (RWX) access mode.
- The cluster must have sufficient RAM and network bandwidth.

NOTE

You must ensure that there is enough memory request capacity in the cluster to support node drains that result in live migrations. You can determine the approximate required spare memory by using the following calculation:

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

The default number of migrations that can run in parallel in the cluster is 5.

- If a VM uses a host model CPU, the nodes must support the CPU.
- Configuring a dedicated Multus network for live migration is highly recommended. A dedicated network minimizes the effects of network saturation on tenant workloads during migration.

13.1.2. About live migration permissions

In OpenShift Virtualization 4.19 and later, live migration operations are restricted to users who are explicitly granted the **kubevirt.io:migrate** cluster role. Users with this role can create, delete, and update virtual machine (VM) live migration requests, which are represented by **VirtualMachineInstanceMigration** (VMIM) custom resources.

Cluster administrators can bind the **kubevirt.io:migrate** role to trusted users or groups at either the namespace or cluster level.

Before OpenShift Virtualization 4.19, namespace administrators had live migration permissions by default. This behavior changed in version 4.19 to prevent unintended or malicious disruptions to infrastructure-critical migration operations.

As a cluster administrator, you can preserve the old behavior by creating a temporary cluster role before updating. After assigning the new role to users, delete the temporary role to enforce the more restrictive permissions. If you have already updated, you can still revert to the old behavior by aggregating the **kubevirt.io:migrate** role into the **admin** cluster role.

13.1.3. Preserving pre-4.19 live migration permissions during update

Before you update to OpenShift Virtualization 4.20, you can create a temporary cluster role to preserve the previous live migration permissions until you are ready for the more restrictive default permissions to take effect.

Prerequisites

- The OpenShift CLI (**oc**) is installed.
- You have cluster administrator permissions.

Procedure

1. Before updating to OpenShift Virtualization 4.20, create a temporary **ClusterRole** object. For example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.authorization.k8s.io/aggregate-to-admin=true 1
  name: kubevirt.io:upgrademigrate
rules:
- apiGroups:
  - subresources.kubevirt.io
  resources:
  - virtualmachines/migrate
  verbs:
  - update
- apiGroups:
  - kubevirt.io
  resources:
  - virtualmachineinstancemigrations
  verbs:
  - get
  - delete
  - create
  - update
  - patch
  - list
  - watch
  - deletecollection
```

1 This cluster role is aggregated into the **admin** role before you update OpenShift Virtualization. The update process does not modify it, ensuring the previous behavior is maintained.

2. Add the cluster role manifest to the cluster by running the following command:

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

3. Update OpenShift Virtualization to version 4.20.

4. Bind the **kubevirt.io:migrate** cluster role to trusted users or groups by running one of the following commands, replacing **<namespace>**, **<first_user>**, **<second_user>**, and **<group_name>** with your own values.

   - To bind the role at the namespace level, run the following command:

     ```
     $ oc create -n <namespace> rolebinding kvmigrate --clusterrole=kubevirt.io:migrate --user=<first_user> --user=<second_user> --group=<group_name>
     ```

   - To bind the role at the cluster level, run the following command:

     ```
     $ oc create clusterrolebinding kvmigrate --clusterrole=kubevirt.io:migrate --user=<first_user> --user=<second_user> --group=<group_name>
     ```

5. When you have bound the **kubevirt.io:migrate** role to all necessary users, delete the temporary **ClusterRole** object by running the following command:

   ```
   $ oc delete clusterrole kubevirt.io:upgrademigrate
   ```

   After you delete the temporary cluster role, only users with the **kubevirt.io:migrate** role can create, delete, and update live migration requests.

13.1.4. Granting live migration permissions

Grant trusted users or groups the ability to create, delete, and update live migration instances.

Prerequisites

- The OpenShift CLI (**oc**) is installed.
- You have cluster administrator permissions.

Procedure

- (Optional) To change the default behavior so that namespace administrators always have permission to create, delete, and update live migrations, aggregate the **kubevirt.io:migrate** role into the **admin** cluster role by running the following command:

  ```
  $ oc label --overwrite clusterrole kubevirt.io:migrate rbac.authorization.k8s.io/aggregate-to-admin=true
  ```

- Bind the **kubevirt.io:migrate** cluster role to trusted users or groups by running one of the following commands, replacing **<namespace>**, **<first_user>**, **<second_user>**, and **<group_name>** with your own values.

  - To bind the role at the namespace level, run the following command:

    ```
    $ oc create -n <namespace> rolebinding kvmigrate --clusterrole=kubevirt.io:migrate --user=<first_user> --user=<second_user> --group=<group_name>
    ```

  - To bind the role at the cluster level, run the following command:

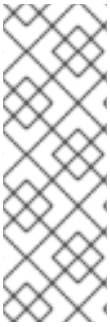
```
$ oc create clusterrolebinding kvmigrate --clusterrole=kubevirt.io:migrate --user=
<first_user> --user=<second_user> --group=<group_name>
```

13.1.5. VM migration tuning

You can adjust your cluster-wide live migration settings based on the type of workload and migration scenario. This enables you to control how many VMs migrate at the same time, the network bandwidth you want to use for each migration, and how long OpenShift Virtualization attempts to complete the migration before canceling the process. Configure these settings in the **HyperConverged** custom resource (CR).

If you are migrating multiple VMs per node at the same time, set a **bandwidthPerMigration** limit to prevent a large or busy VM from using a large portion of the node's network bandwidth. By default, the **bandwidthPerMigration** value is **0**, which means unlimited.

A large VM running a heavy workload (for example, database processing), with higher memory dirty rates, requires a higher bandwidth to complete the migration.



NOTE

Post copy mode, when enabled, triggers if the initial pre-copy phase does not complete within the defined timeout. During post copy, the VM CPUs pause on the source host while transferring the minimum required memory pages. Then the VM CPUs activate on the destination host, and the remaining memory pages transfer into the destination node at runtime. This can impact performance during the transfer.

Post copy mode should not be used for critical data, or with unstable networks.

13.1.6. Common live migration tasks

You can perform the following live migration tasks:

- Configure live migration settings
- Configure live migration for heavy workloads
- Initiate and cancel live migration
- Monitor the progress of all live migrations in the **Migrations** tab of the OpenShift Container Platform web console.
- View VM migration metrics in the **Metrics** tab of the web console.

13.1.7. Additional resources

- Default cluster roles for OpenShift Virtualization
- Prometheus queries for live migration
- VM run strategies
- VM and cluster eviction strategies

13.2. CONFIGURING LIVE MIGRATION

You can configure live migration settings to ensure that the migration processes do not overwhelm the cluster.

You can configure live migration policies to apply different migration configurations to groups of virtual machines (VMs).

13.2.1. Configuring live migration limits and timeouts

Configure live migration limits and timeouts for the cluster by updating the **HyperConverged** custom resource (CR), which is located in the **openshift-cnv** namespace.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- Edit the **HyperConverged** CR and add the necessary live migration parameters:

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

Example configuration file

```
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)
    allowPostCopy: false (6)
```

1. Bandwidth limit of each migration, where the value is the quantity of bytes per second. For example, a value of **2048Mi** means 2048 MiB/s. Default: **0**, which is unlimited.
2. The migration is canceled if it has not completed in this time, in seconds per GiB of memory. For example, a VM with 6GiB memory times out if it has not completed migration in 4800 seconds. If the **Migration Method** is **BlockMigration**, the size of the migrating disks is included in the calculation.
3. Number of migrations running in parallel in the cluster. Default: **5**.
4. Maximum number of outbound migrations per node. Default: **2**.
5. The migration is canceled if memory copy fails to make progress in this time, in seconds. Default: **150**.
6. If a VM is running a heavy workload and the memory dirty rate is too high, this can prevent

NOTE

You can restore the default value for any **spec.liveMigrationConfig** field by deleting that key/value pair and saving the file. For example, delete **progressTimeout: <value>** to restore the default **progressTimeout: 150**.

13.2.2. Configure live migration for heavy workloads

When migrating a VM running a heavy workload (for example, database processing) with higher memory dirty rates, you need a higher bandwidth to complete the migration.

If the dirty rate is too high, the migration from one node to another does not converge. To prevent this, enable post copy mode.

Post copy mode triggers if the initial pre-copy phase does not complete within the defined timeout. During post copy, the VM CPUs pause on the source host while transferring the minimum required memory pages. Then the VM CPUs activate on the destination host, and the remaining memory pages transfer into the destination node at runtime.

Configure live migration for heavy workloads by updating the **HyperConverged** custom resource (CR), which is located in the **openshift-cnv** namespace.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **HyperConverged** CR and add the necessary parameters for migrating heavy workloads:

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

Example configuration file

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  liveMigrationConfig:
    bandwidthPerMigration: 0Mi (1)
    completionTimeoutPerGiB: 150 (2)
    parallelMigrationsPerCluster: 5 (3)
    parallelOutboundMigrationsPerNode: 1 (4)
    progressTimeout: 150 (5)
    allowPostCopy: true (6)
```

(1) Bandwidth limit of each migration, where the value is the quantity of bytes per second. The default is **0**, which is unlimited.

(2)

The migration is canceled if it is not completed in this time, and triggers post copy mode, when post copy is enabled. This value is measured in seconds per GiB of memory. You can

3 Number of migrations running in parallel in the cluster. The default is **5**. Keeping the **parallelMigrationsPerCluster** setting low is better when migrating heavy workloads.

4 Maximum number of outbound migrations per node. Configure a single VM per node for heavy workloads.

5 The migration is canceled if memory copy fails to make progress in this time. This value is measured in seconds. Increase this parameter for large memory sizes running heavy workloads.

6 Use post copy mode when memory dirty rates are high to ensure the migration converges. Set **allowPostCopy** to **true** to enable post copy mode.

2. Optional: If your main network is too busy for the migration, configure a secondary, dedicated migration network.

NOTE

Post copy mode can impact performance during the transfer, and should not be used for critical data, or with unstable networks.

13.2.3. Additional resources

- Configuring a dedicated network for live migration

13.2.4. Live migration policies

You can create live migration policies to apply different migration configurations to groups of VMs that are defined by VM or project labels.

TIP

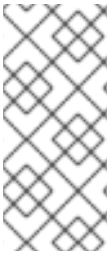
You can create live migration policies by using the OpenShift Container Platform web console.

13.2.4.1. Creating a live migration policy by using the CLI

You can create a live migration policy by using the command line. KubeVirt applies the live migration policy to selected virtual machines (VMs) by using any combination of labels:

- VM labels such as **size**, **os**, or **gpu**
- Project labels such as **priority**, **bandwidth**, or **hpc-workload**

For the policy to apply to a specific group of VMs, all labels on the group of VMs must match the labels of the policy.



NOTE

If multiple live migration policies apply to a VM, the policy with the greatest number of matching labels takes precedence.

If multiple policies meet this criteria, the policies are sorted by alphabetical order of the matching label keys, and the first one in that order takes precedence.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the VM object to which you want to apply a live migration policy, and add the corresponding VM labels.

   a. Open the YAML configuration of the resource:

   ```
   $ oc edit vm <vm_name>
   ```

   b. Adjust the required label values in the **.spec.template.metadata.labels** section of the configuration. For example, to mark the VM as a **production** VM for the purposes of migration policies, add the **kubevirt.io/environment: production** line:

   ```
   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. Save and exit the configuration.

2. Configure a **MigrationPolicy** object with the corresponding labels. The following example configures a policy that applies to all VMs that are labeled as **production**:

   ```
   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. Specify project labels.
2. Specify VM labels.

3. Create the migration policy by running the following command:

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

13.2.5. Migrating a VM to a specific node

You can migrate a running virtual machine (VM) to a specific subset of nodes by using the **addedNodeSelector** field on the **VirtualMachineInstanceMigration** object. This field lets you apply additional node selection rules for a **one-time** migration attempt, without affecting the VM configuration or future migrations.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- The VM you want to migrate is running.
- You have identified the labels of the target nodes. Multiple labels can be specified and are combined with logical **AND**.
- The **oc** CLI tool is installed.

Procedure

1. Create a migration manifest YAML file. For example:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstanceMigration
metadata:
  name: migration-job
spec:
  vmiName: vmi-fedora
  addedNodeSelector:
    accelerator: gpu-enabled23
    kubernetes.io/hostname: "ip-172-28-114-199.example"
```

where:

vmiName

Specifies the name of the running VM (for example, **vmi-fedora**).

addedNodeSelector

Specifies additional constraints for selecting the target node.

2. Apply the manifest to the cluster by running the following command:

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

If no nodes satisfy the constraints, the migration is declared a failure after a timeout. The VM remains unaffected.

13.2.6. Additional resources

- Configuring a dedicated Multus network for live migration

13.3. INITIATING AND CANCELING LIVE MIGRATION

You can initiate the live migration of a virtual machine (VM) to another node by using the OpenShift Container Platform web console or the command line.

You can cancel a live migration by using the web console or the command line. The VM remains on its original node.

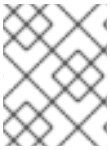
TIP

You can also initiate and cancel live migration by using the **virtctl migrate <vm_name>** and **virtctl migrate-cancel <vm_name>** commands.

13.3.1. Initiating live migration

13.3.1.1. Initiating live migration by using the web console

You can live migrate a running virtual machine (VM) to a different node in the cluster by using the OpenShift Container Platform web console.



NOTE

The **Migrate** action is visible to all users but only cluster administrators can initiate a live migration.

Prerequisites

- You have the **kubevirt.io:migrate** RBAC role or you are a cluster administrator.
- The VM is migratable.
- If the VM is configured with a host model CPU, the cluster has an available node that supports the CPU model.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Take either of the following steps:
   - Click the Options menu  beside the VM you want to migrate, hover over the **Migrate** option, and select **Compute**.

- Open the **VM details** page of the VM you want to migrate, click the **Actions** menu, hover over the **Migrate** option, and select **Compute**.

3. In the **Migrate Virtual Machine to a different Node** dialog box, select either **Automatically Selected Node** or **Specific Node**.
   a. If you selected the **Specific Node** option, choose a node from the list.
4. Click **Migrate Virtual Machine**

13.3.1.2. Initiating live migration by using the CLI

You can initiate the live migration of a running virtual machine (VM) by using the command line to create a **VirtualMachineInstanceMigration** object for the VM.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have the **kubevirt.io:migrate** RBAC role or you are a cluster administrator.

Procedure

1. Create a **VirtualMachineInstanceMigration** manifest for the VM that you want to migrate:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstanceMigration
metadata:
  name: <migration_name>
spec:
  vmiName: <vm_name>
```

2. Create the object by running the following command:

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

The **VirtualMachineInstanceMigration** object triggers a live migration of the VM. This object exists in the cluster for as long as the virtual machine instance is running, unless manually deleted.

Verification

- Obtain the VM status by running the following command:

```
$ oc describe vmi <vm_name> -n <namespace>
```

Example output

```
# ...
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
```

13.3.2. Canceling live migration

13.3.2.1. Canceling live migration by using the web console

You can cancel the live migration of a virtual machine (VM) by using the OpenShift Container Platform web console.

Prerequisites

- You have the **kubevirt.io:migrate** RBAC role or you are a cluster administrator.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Select **Cancel Migration** on the Options menu  beside a VM.

13.3.2.2. Canceling live migration by using the CLI

Cancel the live migration of a virtual machine by deleting the **VirtualMachineInstanceMigration** object associated with the migration.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have the **kubevirt.io:migrate** RBAC role or you are a cluster administrator.

Procedure

- Delete the **VirtualMachineInstanceMigration** object that triggered the live migration, **migration-job** in this example:

```
$ oc delete vmim migration-job
```

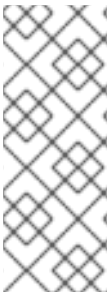
13.3.3. Additional resources

- About live migration permissions

CHAPTER 14. NODES

14.1. NODE MAINTENANCE

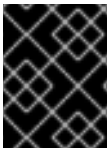
Nodes can be placed into maintenance mode by using the **oc adm** utility or **NodeMaintenance** custom resources (CRs).



NOTE

The **node-maintenance-operator** (NMO) is no longer shipped with OpenShift Virtualization. It is deployed as a standalone Operator from the software catalog in the OpenShift Container Platform web console or by using the OpenShift CLI (**oc**).

For more information on remediation, fencing, and maintaining nodes, see the Workload Availability for Red Hat OpenShift documentation.



IMPORTANT

Virtual machines (VMs) must have a persistent volume claim (PVC) with a shared **ReadWriteMany** (RWX) access mode to be live migrated.

The Node Maintenance Operator watches for new or deleted **NodeMaintenance** CRs. When a new **NodeMaintenance** CR is detected, no new workloads are scheduled and the node is cordoned off from the rest of the cluster. All pods that can be evicted are evicted from the node. When a **NodeMaintenance** CR is deleted, the node that is referenced in the CR is made available for new workloads.

NOTE

Using a **NodeMaintenance** CR for node maintenance tasks achieves the same results as the **oc adm cordon** and **oc adm drain** commands using standard OpenShift Container Platform custom resource processing.

14.1.1. Eviction strategies

Placing a node into maintenance marks the node as unschedulable and drains all the VMs and pods from it.

You can configure eviction strategies for virtual machines (VMs) or for the cluster.

VM eviction strategy

The VM **LiveMigrate** eviction strategy ensures that a virtual machine instance (VMI) is not interrupted if the node is placed into maintenance or drained. VMIs with this eviction strategy will be live migrated to another node.

You can configure eviction strategies for virtual machines (VMs) by using the OpenShift Container Platform web console or the command line.

IMPORTANT

The default eviction strategy is **LiveMigrate**. A non-migratable VM with a **LiveMigrate** eviction strategy might prevent nodes from draining or block an infrastructure upgrade because the VM is not evicted from the node. This situation causes a migration to remain in a **Pending** or **Scheduling** state unless you shut down the VM manually.

You must set the eviction strategy of non-migratable VMs to **LiveMigrateIfPossible**, which does not block an upgrade, or to **None**, for VMs that should not be migrated.

Cluster eviction strategy

You can configure an eviction strategy for the cluster to prioritize workload continuity or infrastructure upgrade.

Table 14.1. Cluster eviction strategies

Eviction strategy	Description	Interrupts workflow	Blocks upgrades
LiveMigrate[1]	Prioritizes workload continuity over upgrades.	No	Yes[2]
LiveMigrateIfPossible	Prioritizes upgrades over workload continuity to ensure that the environment is updated.	Yes	No
None[3]	Shuts down VMs with no eviction strategy.	Yes	No

1. Default eviction strategy for multi-node clusters.
2. If a VM blocks an upgrade, you must shut down the VM manually.
3. Default eviction strategy for single-node OpenShift.

14.1.1.1. Configuring a VM eviction strategy using the CLI

You can configure an eviction strategy for a virtual machine (VM) by using the command line.

IMPORTANT

The default eviction strategy is **LiveMigrate**. A non-migratable VM with a **LiveMigrate** eviction strategy might prevent nodes from draining or block an infrastructure upgrade because the VM is not evicted from the node. This situation causes a migration to remain in a **Pending** or **Scheduling** state unless you shut down the VM manually.

You must set the eviction strategy of non-migratable VMs to **LiveMigrateIfPossible**, which does not block an upgrade, or to **None**, for VMs that should not be migrated.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **VirtualMachine** resource by running the following command:

```
$ oc edit vm <vm_name> -n <namespace>
```

Example eviction strategy

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: <vm_name>
spec:
  template:
    spec:
      evictionStrategy: LiveMigrateIfPossible (1)
# ...
```

(1) Specify the eviction strategy. The default value is **LiveMigrate**.

2. Restart the VM to apply the changes:

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

14.1.1.2. Configuring a cluster eviction strategy by using the CLI

You can configure an eviction strategy for a cluster by using the command line.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **hyperconverged** resource by running the following command:

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

2. Set the cluster eviction strategy as shown in the following example:

Example cluster eviction strategy

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  evictionStrategy: LiveMigrate
# ...
```

14.1.2. Run strategies

The **spec.runStrategy** key determines how a VM behaves under certain conditions.

14.1.2.1. Run strategies

The **spec.runStrategy** key has four possible values:

Always

The virtual machine instance (VMI) is always present when a virtual machine (VM) is created on another node. A new VMI is created if the original stops for any reason.

RerunOnFailure

The VMI is re-created on another node if the previous instance fails. The instance is not re-created if the VM stops successfully, such as when it is shut down.

Manual

You control the VMI state manually with the **start**, **stop**, and **restart** virtctl client commands. The VM is not automatically restarted.

Halted

No VMI is present when a VM is created.

Different combinations of the **virtctl start**, **stop** and **restart** commands affect the run strategy.

The following table describes a VM's transition between states. The first column shows the VM's initial run strategy. The remaining columns show a virtctl command and the new run strategy after that command is run.

Table 14.2. Run strategy before and after virtctl commands

Initial run strategy	Start	Stop	Restart
Always	-	Halted	Always
RerunOnFailure	RerunOnFailure	RerunOnFailure	RerunOnFailure
Manual	Manual	Manual	Manual
Halted	Always	-	-

NOTE

If a node in a cluster installed by using installer-provisioned infrastructure fails the machine health check and is unavailable, VMs with **runStrategy: Always** or **runStrategy: RerunOnFailure** are rescheduled on a new node.

14.1.2.2. Configuring a VM run strategy by using the CLI

You can configure a run strategy for a virtual machine (VM) by using the command line.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- Edit the **VirtualMachine** resource by running the following command:

```
$ oc edit vm <vm_name> -n <namespace>
```

Example run strategy

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  runStrategy: Always
# ...
```

14.1.3. Maintaining bare metal nodes

When you deploy OpenShift Container Platform on bare metal infrastructure, there are additional considerations that must be taken into account compared to deploying on cloud infrastructure. Unlike in cloud environments where the cluster nodes are considered ephemeral, re-provisioning a bare metal node requires significantly more time and effort for maintenance tasks.

When a bare metal node fails, for example, if a fatal kernel error happens or a NIC card hardware failure occurs, workloads on the failed node need to be restarted elsewhere else on the cluster while the problem node is repaired or replaced. Node maintenance mode allows cluster administrators to gracefully power down nodes, moving workloads to other parts of the cluster and ensuring workloads do not get interrupted. Detailed progress and node status details are provided during maintenance.

14.1.4. Additional resources

- About live migration

14.2. MANAGING NODE LABELING FOR OBSOLETE CPU MODELS

You can schedule a virtual machine (VM) on a node as long as the VM CPU model and policy are supported by the node.

14.2.1. About node labeling for obsolete CPU models

The OpenShift Virtualization Operator uses a predefined list of obsolete CPU models to ensure that a node supports only valid CPU models for scheduled VMs.

By default, the following CPU models are eliminated from the list of labels generated for the node:

Example 14.1. Obsolete CPU models

```
"486"
Conroe
athlon
core2duo
coreduo
kvm32
```

```
kvm64
n270
pentium
pentium2
pentium3
pentiumpro
phenom
qemu32
qemu64
```

This predefined list is not visible in the **HyperConverged** CR. You cannot *remove* CPU models from this list, but you can add to the list by editing the **spec.obsoleteCPUs.cpuModels** field of the **HyperConverged** CR.

14.2.2. Configuring obsolete CPU models

You can configure a list of obsolete CPU models by editing the **HyperConverged** custom resource (CR).

Procedure

- Edit the **HyperConverged** custom resource, specifying the obsolete CPU models in the **obsoleteCPUs** array. For example:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  obsoleteCPUs:
    cpuModels: 1
      - "<obsolete_cpu_1>"
      - "<obsolete_cpu_2>"
```

1 Replace the example values in the **cpuModels** array with obsolete CPU models. Any value that you specify is added to a predefined list of obsolete CPU models. The predefined list is not visible in the CR.

14.3. PREVENTING NODE RECONCILIATION

Use **skip-node** annotation to prevent the **node-labeller** from reconciling a node.

14.3.1. Using skip-node annotation

If you want the **node-labeller** to skip a node, annotate that node by using the **oc** CLI.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- Annotate the node that you want to skip by running the following command:

```
$ oc annotate node <node_name> node-labeller.kubevirt.io/skip-node=true 1
```

1 Replace **<node_name>** with the name of the relevant node to skip.

Reconciliation resumes on the next cycle after the node annotation is removed or set to false.

14.3.2. Additional resources

- Managing node labeling for obsolete CPU models

14.4. DELETING A FAILED NODE TO TRIGGER VIRTUAL MACHINE FAILOVER

If a node fails and node health checks are not deployed on your cluster, virtual machines (VMs) with **runStrategy: Always** configured are not automatically relocated to healthy nodes.

14.4.1. Prerequisites

- A node where a virtual machine was running has the **NotReady** condition.
- The virtual machine that was running on the failed node has **runStrategy** set to **Always**.
- You have installed the OpenShift CLI (**oc**).

14.4.2. Deleting nodes from a bare metal cluster

When you delete a node using the CLI, the node object is deleted in Kubernetes, but the pods that exist on the node are not deleted. Any bare pods not backed by a replication controller become inaccessible to OpenShift Container Platform. Pods backed by replication controllers are rescheduled to other available nodes. You must delete local manifest pods.

Procedure

Delete a node from an OpenShift Container Platform cluster running on bare metal by completing the following steps:

1. Mark the node as unschedulable:

```
$ oc adm cordon <node_name>
```

2. Drain all pods on the node:

```
$ oc adm drain <node_name> --force=true
```

This step might fail if the node is offline or unresponsive. Even if the node does not respond, it might still be running a workload that writes to shared storage. To avoid data corruption, power down the physical hardware before you proceed.

3. Delete the node from the cluster:

```
$ oc delete node <node_name>
```

Although the node object is now deleted from the cluster, it can still rejoin the cluster after reboot or if the kubelet service is restarted. To permanently delete the node and all its data, you must decommission the node.

4. If you powered down the physical hardware, turn it back on so that the node can rejoin the cluster.

14.4.3. Verifying virtual machine failover

After all resources are terminated on the unhealthy node, a new virtual machine instance (VMI) is automatically created on a healthy node for each relocated VM. To confirm that the VMI was created, view all VMIs by using the **oc** CLI.

14.4.3.1. Listing all virtual machine instances using the CLI

You can list all virtual machine instances (VMIs) in your cluster, including standalone VMIs and those owned by virtual machines, by using the **oc** command-line interface (CLI).

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- List all VMIs by running the following command:

```
$ oc get vmis -A
```

14.5. ACTIVATING KERNEL SAMEPAGE MERGING (KSM)

OpenShift Virtualization can activate kernel samepage merging (KSM) when nodes are overloaded. KSM deduplicates identical data found in the memory pages of virtual machines (VMs). If you have very similar VMs, KSM can make it possible to schedule more VMs on a single node.

IMPORTANT

You must only use KSM with trusted workloads.

14.5.1. Prerequisites

- Ensure that an administrator has configured KSM support on any nodes where you want OpenShift Virtualization to activate KSM.

14.5.2. About using OpenShift Virtualization to activate KSM

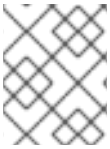
You can configure OpenShift Virtualization to activate kernel samepage merging (KSM) when nodes experience memory overload.

14.5.2.1. Configuration methods

You can enable or disable the KSM activation feature for all nodes by using the OpenShift Container Platform web console or by editing the **HyperConverged** custom resource (CR). The **HyperConverged** CR supports more granular configuration.

CR configuration
You can configure the KSM activation feature by editing the **spec.configuration.ksmConfiguration** stanza of the **HyperConverged** CR.

- You enable the feature and configure settings by editing the **ksmConfiguration** stanza.
- You disable the feature by deleting the **ksmConfiguration** stanza.
- You can allow OpenShift Virtualization to enable KSM on only a subset of nodes by adding node selection syntax to the **ksmConfiguration.nodeLabelSelector** field.



NOTE

Even if the KSM activation feature is disabled in OpenShift Virtualization, an administrator can still enable KSM on nodes that support it.

14.5.2.2. KSM node labels

OpenShift Virtualization identifies nodes that are configured to support KSM and applies the following node labels:

kubevirt.io/ksm-handler-managed: "false"
: This label is set to **"true"** when OpenShift Virtualization activates KSM on a node that is experiencing memory overload. This label is not set to **"true"** if an administrator activates KSM.

kubevirt.io/ksm-enabled: "false"
: This label is set to **"true"** when KSM is activated on a node, even if OpenShift Virtualization did not activate KSM.

These labels are not applied to nodes that do not support KSM.

14.5.3. Configuring KSM activation by using the web console

You can allow OpenShift Virtualization to activate kernel samepage merging (KSM) on all nodes in your cluster by using the OpenShift Container Platform web console.

Procedure

1. From the side menu, click **Virtualization → Overview**.
2. Select the **Settings** tab.
3. Select the **Cluster** tab.
4. Expand **Resource management**.
5. Enable or disable the feature for all nodes:
   - Set **Kernel Samepage Merging (KSM)** to on.
   - Set **Kernel Samepage Merging (KSM)** to off.

14.5.4. Configuring KSM activation by using the CLI

You can enable or disable OpenShift Virtualization's kernel samepage merging (KSM) activation feature by editing the **HyperConverged** custom resource (CR). Use this method if you want OpenShift Virtualization to activate KSM on only a subset of nodes.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Open the **HyperConverged** CR in your default editor by running the following command:

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

2. Edit the **ksmConfiguration** stanza:

   - To enable the KSM activation feature for all nodes, set the **nodeLabelSelector** value to **{}**. For example:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  configuration:
    ksmConfiguration:
      nodeLabelSelector: {}
# ...
```

   - To enable the KSM activation feature on a subset of nodes, edit the **nodeLabelSelector** field. Add syntax that matches the nodes where you want OpenShift Virtualization to enable KSM. For example, the following configuration allows OpenShift Virtualization to enable KSM on nodes where both **<first_example_key>** and **<second_example_key>** are set to **"true"**:

```
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"
# ...
```

   - To disable the KSM activation feature, delete the **ksmConfiguration** stanza. For example:

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

3. Save the file.

14.5.5. Additional resources

- Specifying nodes for virtual machines
- Placing pods on specific nodes using node selectors
- Managing kernel samepage merging in the Red Hat Enterprise Linux (RHEL) documentation

CHAPTER 15. MONITORING

15.1. MONITORING OVERVIEW

You can monitor the health of your cluster and virtual machines (VMs) with the following tools:

Monitoring OpenShift Virtualization VM health status

View the overall health of your OpenShift Virtualization environment in the web console by navigating to the **Home → Overview** page in the OpenShift Container Platform web console. The **Status** card displays the overall health of OpenShift Virtualization based on the alerts and conditions.

OpenShift Container Platform cluster checkup framework

Run automated tests on your cluster with the OpenShift Container Platform cluster checkup framework to check the following conditions:

- Network connectivity and latency between two VMs attached to a secondary network interface
- VM running a Data Plane Development Kit (DPDK) workload with zero packet loss
- Cluster storage is optimally configured for OpenShift Virtualization

Prometheus queries for virtual resources

Query vCPU, network, storage, and guest memory swapping usage and live migration progress.

VM custom metrics

Configure the **node-exporter** service to expose internal VM metrics and processes.

VM health checks

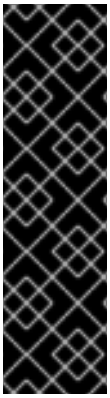
Configure readiness, liveness, and guest agent ping probes and a watchdog for VMs.

Runbooks

Diagnose and resolve issues that trigger OpenShift Virtualization alerts in the OpenShift Container Platform web console.

15.2. OPENSHIFT VIRTUALIZATION CLUSTER CHECKUP FRAMEWORK

A *checkup* is an automated test workload that allows you to verify if a specific cluster functionality works as expected. The cluster checkup framework uses native Kubernetes resources to configure and execute the checkup.



IMPORTANT

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

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

As a developer or cluster administrator, you can use predefined checkups to improve cluster

maintainability, troubleshoot unexpected behavior, minimize errors, and save time. You can review the results of the checkup and share them with experts for further analysis. Vendors can write and publish checkups for features or services that they provide and verify that their customer environments are configured correctly.

15.2.1. Running predefined latency checkups

You can use a latency checkup to verify network connectivity and measure latency between two virtual machines (VMs) that are attached to a secondary network interface. The predefined latency checkup uses the ping utility.

IMPORTANT

Before you run a latency checkup, you must first create a bridge interface on the cluster nodes to connect the VM's secondary interface to any interface on the node. If you do not create a bridge interface, the VMs do not start and the job fails.

Running a predefined checkup in an existing namespace involves setting up a service account for the checkup, creating the **Role** and **RoleBinding** objects for the service account, enabling permissions for the checkup, and creating the input config map and the checkup job. You can run a checkup multiple times.

IMPORTANT

You must always:

- Verify that the checkup image is from a trustworthy source before applying it.
- Review the checkup permissions before creating the **Role** and **RoleBinding** objects.

15.2.1.1. Running a latency checkup by using the web console

Run a latency checkup to verify network connectivity and measure the latency between two virtual machines attached to a secondary network interface.

Prerequisites

- You must add a **NetworkAttachmentDefinition** to the namespace.

Procedure

1. Navigate to **Virtualization → Checkups** in the web console.
2. Click the **Network latency** tab.
3. Click **Install permissions**.
4. Click **Run checkup**.
5. Enter a name for the checkup in the **Name** field.
6. Select a **NetworkAttachmentDefinition** from the drop-down menu.

7. Optional: Set a duration for the latency sample in the **Sample duration (seconds)** field.
8. Optional: Define a maximum latency time interval by enabling **Set maximum desired latency (milliseconds)** and defining the time interval.
9. Optional: Target specific nodes by enabling **Select nodes** and specifying the **Source node** and **Target node**.
10. Click **Run**.

Verification

- To view the status of the latency checkup, go to the **Checkups** list on the **Latency checkup** tab. Click on the name of the checkup for more details.

15.2.1.2. Running a latency checkup by using the CLI

You run a latency checkup using the CLI by performing the following steps:

1. Create a service account, roles, and rolebindings to provide cluster access permissions to the latency checkup.
2. Create a config map to provide the input to run the checkup and to store the results.
3. Create a job to run the checkup.
4. Review the results in the config map.
5. Optional: To rerun the checkup, delete the existing config map and job and then create a new config map and job.
6. When you are finished, delete the latency checkup resources.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- The cluster has at least two worker nodes.
- You configured a network attachment definition for a namespace.

Procedure

1. Create a **ServiceAccount**, **Role**, and **RoleBinding** manifest for the latency checkup:

Example 15.1. Example role manifest file

```
---
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. Apply the **ServiceAccount**, **Role**, and **RoleBinding** manifest:

```
$ oc apply -n <target_namespace> -f <latency_sa_roles_rolebinding>.yaml 1
```

1 **<target_namespace>** is the namespace where the checkup is to be run. This must be an existing namespace where the **NetworkAttachmentDefinition** object resides.

3. Create a **ConfigMap** manifest that contains the input parameters for the checkup:

Example input config map

```
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. The name of the **NetworkAttachmentDefinition** object.
2. Optional: The maximum desired latency, in milliseconds, between the virtual machines. If the measured latency exceeds this value, the checkup fails.
3. Optional: The duration of the latency check, in seconds.
4. Optional: When specified, latency is measured from this node to the target node. If the source node is specified, the **spec.param.targetNode** field cannot be empty.
5. Optional: When specified, latency is measured from the source node to this node.

4. Apply the config map manifest in the target namespace:

```
$ oc apply -n <target_namespace> -f <latency_config_map>.yaml
```

5. Create a **Job** manifest to run the checkup:

Example job manifest

```
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:v4.20.0
```

```
      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. Apply the **Job** manifest:

```
$ oc apply -n <target_namespace> -f <latency_job>.yaml
```

7. Wait for the job to complete:

```
$ oc wait job kubevirt-vm-latency-checkup -n <target_namespace> --for condition=complete --timeout 6m
```

8. Review the results of the latency checkup by running the following command. If the maximum measured latency is greater than the value of the **spec.param.maxDesiredLatencyMilliseconds** attribute, the checkup fails and returns an error.

```
$ oc get configmap kubevirt-vm-latency-checkup-config -n <target_namespace> -o yaml
```

Example output config map (success)

```
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 The maximum measured latency in nanoseconds.

9. Optional: To view the detailed job log in case of checkup failure, use the following command:

```
$ oc logs job.batch/kubevirt-vm-latency-checkup -n <target_namespace>
```

10. Delete the job and config map that you previously created by running the following commands:

```
$ oc delete job -n <target_namespace> kubevirt-vm-latency-checkup
```

```
$ oc delete config-map -n <target_namespace> kubevirt-vm-latency-checkup-config
```

11. Optional: If you do not plan to run another checkup, delete the roles manifest:

```
$ oc delete -f <latency_sa_roles_rolebinding>.yaml
```

15.2.2. Running predefined storage checkups

You can use a storage checkup to verify that the cluster storage is optimally configured for OpenShift Virtualization.

Running a predefined checkup in an existing namespace involves setting up a service account for the checkup, creating the **Role** and **RoleBinding** objects for the service account, enabling permissions for the checkup, and creating the input config map and the checkup job. You can run a checkup multiple times.

IMPORTANT

You must always:

- Verify that the checkup image is from a trustworthy source before applying it.
- Review the checkup permissions before creating the **Role** and **RoleBinding** objects.

15.2.2.1. Retaining resources for troubleshooting storage checkups

The predefined storage checkup includes **skipTeardown** configuration options, which control resource clean up after a storage checkup runs. By default, the **skipTeardown** field value is **Never**, which means that the checkup always performs teardown steps and deletes all resources after the checkup runs.

You can retain resources for further inspection in case a failure occurs by setting the **skipTeardown** field to **onfailure**.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Run the following command to edit the **storage-checkup-config** config map:

```
$ oc edit configmap storage-checkup-config -n <checkup_namespace>
```

2. Configure the **skipTeardown** field to use the **onfailure** value. You can do this by modifying the **storage-checkup-config** config map, stored in the **storage_checkup.yaml** file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: storage-checkup-config
  namespace: <checkup_namespace>
data:
  spec.param.skipTeardown: onfailure
# ...
```

3. Reapply the **storage-checkup-config** config map by running the following command:

```
$ oc apply -f storage_checkup.yaml -n <checkup_namespace>
```

15.2.2.2. Running a storage checkup by using the web console

Run a storage checkup to validate that storage is working correctly for virtual machines.

Procedure

1. Navigate to **Virtualization → Checkups** in the web console.
2. Click the **Storage** tab.
3. Click **Install permissions**.
4. Click **Run checkup**.
5. Enter a name for the checkup in the **Name** field.
6. Enter a timeout value for the checkup in the **Timeout (minutes)** fields.
7. Click **Run**.

You can view the status of the storage checkup in the **Checkups** list on the **Storage** tab. Click on the name of the checkup for more details.

15.2.2.3. Running a storage checkup by using the CLI

Use a predefined checkup to verify that the OpenShift Container Platform cluster storage is configured optimally to run OpenShift Virtualization workloads.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- The cluster administrator has created the required **cluster-reader** permissions for the storage checkup service account and namespace, such as in the following example:

```
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 The namespace where the checkup is to be run.

Procedure

1. Create a **ServiceAccount**, **Role**, and **RoleBinding** manifest file for the storage checkup:

Example 15.2. Example service account, role, and rolebinding manifest

```
---
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. Apply the **ServiceAccount**, **Role**, and **RoleBinding** manifest in the target namespace:

```
$ oc apply -n <target_namespace> -f <storage_sa_roles_rolebinding>.yaml
```

3. Create a **ConfigMap** and **Job** manifest file. The config map contains the input parameters for the checkup job.

Example input config map and job manifest

```
---
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. Apply the **ConfigMap** and **Job** manifest file in the target namespace to run the checkup:

```
$ oc apply -n <target_namespace> -f <storage_configmap_job>.yaml
```

5. Wait for the job to complete:

```
$ oc wait job storage-checkup -n <target_namespace> --for condition=complete --timeout
10m
```

6. Review the results of the checkup by running the following command:

```
$ oc get configmap storage-checkup-config -n <target_namespace> -o yaml
```

Example output config map (success)

```
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: 4.20.2 (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: 4.20.0 (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. Specifies if the checkup is successful (**true**) or not (**false**).
2. The reason for failure if the checkup fails.
3. The time when the checkup started, in RFC 3339 time format.
4. The time when the checkup has completed, in RFC 3339 time format.
5. The OpenShift Virtualization version.
6. Specifies if there is a default storage class.
7. The list of golden images whose data source is not ready.
8. The list of golden images whose data import cron is not up-to-date.
9. The OpenShift Container Platform version.
10. Specifies if a PVC of 10Mi has been created and bound by the provisioner.
11. The list of storage profiles using snapshot-based clone but missing VolumeSnapshotClass.
12. The list of storage profiles with unknown provisioners.
13. The list of storage profiles with smart clone support (CSI/snapshot).
14. The list of storage profiles spec-overriden claimPropertySets.
15. The list of virtual machines that use the Ceph RBD storage class when the virtualization storage class exists.
16. The list of virtual machines that use an Elastic File Store (EFS) storage class where the GID and UID are not set in the storage class.

7. Delete the job and config map that you previously created by running the following commands:

```
$ oc delete job -n <target_namespace> storage-checkup
```

```
$ oc delete config-map -n <target_namespace> storage-checkup-config
```

8. Optional: If you do not plan to run another checkup, delete the **ServiceAccount**, **Role**, and **RoleBinding** manifest:

```
$ oc delete -f <storage_sa_roles_rolebinding>.yaml
```

15.2.2.4. Troubleshooting a failed storage checkup

If a storage checkup fails, there are steps that you can take to identify the reason for failure.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have downloaded the directory provided by the **must-gather** tool.

Procedure

1. Review the **status.failureReason** field in the **storage-checkup-config** config map by running the following command and observing the output:

```
$ oc get configmap storage-checkup-config -n <namespace> -o yaml
```

Example output config map

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: storage-checkup-config
  labels:
    kiagnose/checkup-type: kubevirt-storage
data:
  spec.timeout: 10m
  status.succeeded: "false" (1)
  status.failureReason: "ErrNoDefaultStorageClass" (2)
# ...
```

(1) If the checkup has failed, the **status.succeeded** value is **false**.

(2) If the checkup has failed, the **status.failureReason** field contains an error message. In this example output, the **ErrNoDefaultStorageClass** error message means that no default storage class is configured.

2. Search the directory provided by the **must-gather** tool for logs, events, or terms related to the error in the **data.status.failureReason** field value.

Additional resources

- Collecting data for Red Hat Support
- Using the **must-gather** tool for OpenShift Virtualization

15.2.2.5. Storage checkup error codes

The following error codes might appear in the **storage-checkup-config** config map after a storage checkup fails.

Error code	Meaning
ErrNoDefaultStorageClass	No default storage class is configured.

Error code	Meaning
ErrPvcNotBound	One or more persistent volume claims (PVCs) failed to bind.
ErrMultipleDefaultStorageClasses	Multiple default storage classes are configured.
ErrEmptyClaimPropertySets	There are **StorageProfile** objects containing empty **ClaimPropertySets** specs.
ErrVMsWithUnsetEfsStorageClass	There are VMs using elastic file system (EFS) storage classes, where the GID and UID are not set in the **StorageClass** object.
ErrGoldenImagesNotUpToDate	One or more golden images has a **DataImportCron** object that is either not up to date or has a **DataSource** object which is not ready.
ErrGoldenImageNoDataSource	The **DataSource** object of the golden image has either no PVC or no snapshot source configured.
ErrBootFailedOnSomeVMs	Some VMs failed to boot within the expected time.

15.2.3. Additional resources

- Connecting a virtual machine to a Linux bridge network

15.3. PROMETHEUS QUERIES FOR VIRTUAL RESOURCES

OpenShift Virtualization provides metrics that you can use to monitor the consumption of cluster infrastructure resources, including vCPU, network, storage, and guest memory swapping. You can also use metrics to query live migration status.

15.3.1. Prerequisites

- To use the vCPU metric, the **schedstats=enable** kernel argument must be applied to the **MachineConfig** object. This kernel argument enables scheduler statistics used for debugging and performance tuning and adds a minor additional load to the scheduler. For more information, see Adding kernel arguments to nodes .
- For guest memory swapping queries to return data, memory swapping must be enabled on the virtual guests.

15.3.2. Querying metrics for all projects with the OpenShift Container Platform web console

You can use the OpenShift Container Platform metrics query browser to run Prometheus Query Language (PromQL) queries to examine metrics visualized on a plot. This functionality provides information about the state of a cluster and any user-defined workloads that you are monitoring.

As a cluster administrator or as a user with view permissions for all projects, you can access metrics for all default OpenShift Container Platform and user-defined projects in the Metrics UI.

The Metrics UI includes predefined queries, for example, CPU, memory, bandwidth, or network packet for all projects. You can also run custom Prometheus Query Language (PromQL) queries.

Prerequisites

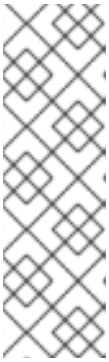
- You have access to the cluster as a user with the **cluster-admin** cluster role or with view permissions for all projects.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. In the OpenShift Container Platform web console, click **Observe → Metrics**.
2. To add one or more queries, perform any of the following actions:

Option	Description
Select an existing query.	From the **Select query** drop-down list, select an existing query.
Create a custom query.	Add your Prometheus Query Language (PromQL) query to the **Expression** field. As you type a PromQL expression, autocomplete suggestions appear in a drop-down list. These suggestions include functions, metrics, labels, and time tokens. Use the keyboard arrows to select one of these suggested items and then press Enter to add the item to your expression. Move your mouse pointer over a suggested item to view a brief description of that item.
Add multiple queries.	Click **Add query**.
Duplicate an existing query.	Click the options menu  next to the query, then choose **Duplicate query**.
Disable a query from being run.	Click the options menu  next to the query and choose **Disable query**.

3. To run queries that you created, click **Run queries**. The metrics from the queries are visualized on the plot. If a query is invalid, the UI shows an error message.



NOTE

- When drawing time series graphs, queries that operate on large amounts of data might time out or overload the browser. To avoid this, click **Hide graph** and calibrate your query by using only the metrics table. Then, after finding a feasible query, enable the plot to draw the graphs.
- By default, the query table shows an expanded view that lists every metric and its current value. Click the ˅ down arrowhead to minimize the expanded view for a query.

4. Optional: Save the page URL to use this set of queries again in the future.

5. Explore the visualized metrics. Initially, all metrics from all enabled queries are shown on the plot. Select which metrics are shown by performing any of the following actions:

Option	Description
Hide all metrics from a query.	Click the options menu ⋮ for the query and click **Hide all series**.
Hide a specific metric.	Go to the query table and click the colored square near the metric name.
Zoom into the plot and change the time range.	Perform one of the following actions: • Visually select the time range by clicking and dragging on the plot horizontally. • Use the menu to select the time range.
Reset the time range.	Click **Reset zoom**.
Display outputs for all queries at a specific point in time.	Hover over the plot at the point you are interested in. The query outputs appear in a pop-up box.
Hide the plot.	Click **Hide graph**.

15.3.3. Querying metrics for user-defined projects with the OpenShift Container Platform web console

You can use the OpenShift Container Platform metrics query browser to run Prometheus Query Language (PromQL) queries to examine metrics visualized on a plot. This functionality provides information about any user-defined workloads that you are monitoring.

As a developer, you must specify a project name when querying metrics. You must have the required privileges to view metrics for the selected project.

The Metrics UI includes predefined queries, for example, CPU, memory, bandwidth, or network packet. These queries are restricted to the selected project. You can also run custom Prometheus Query Language (PromQL) queries for the project.

Prerequisites

- You have access to the cluster as a developer or as a user with view permissions for the project that you are viewing metrics for.
- You have enabled monitoring for user-defined projects.
- You have deployed a service in a user-defined project.
- You have created a **ServiceMonitor** custom resource definition (CRD) for the service to define how the service is monitored.

Procedure

1. In the OpenShift Container Platform web console, click **Observe → Metrics**.
2. To add one or more queries, perform any of the following actions:

Option	Description
Select an existing query.	From the **Select query** drop-down list, select an existing query.
Create a custom query.	Add your Prometheus Query Language (PromQL) query to the **Expression** field. As you type a PromQL expression, autocomplete suggestions appear in a drop-down list. These suggestions include functions, metrics, labels, and time tokens. Use the keyboard arrows to select one of these suggested items and then press Enter to add the item to your expression. Move your mouse pointer over a suggested item to view a brief description of that item.
Add multiple queries.	Click **Add query**.
Duplicate an existing query.	Click the options menu  next to the query, then choose **Duplicate query**.
Disable a query from being run.	Click the options menu  next to the query and choose **Disable query**.

3. To run queries that you created, click **Run queries**. The metrics from the queries are visualized on the plot. If a query is invalid, the UI shows an error message.

NOTE

- When drawing time series graphs, queries that operate on large amounts of data might time out or overload the browser. To avoid this, click **Hide graph** and calibrate your query by using only the metrics table. Then, after finding a feasible query, enable the plot to draw the graphs.
- By default, the query table shows an expanded view that lists every metric and its current value. Click the ˅ down arrowhead to minimize the expanded view for a query.

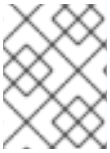
4. Optional: Save the page URL to use this set of queries again in the future.
5. Explore the visualized metrics. Initially, all metrics from all enabled queries are shown on the plot. Select which metrics are shown by performing any of the following actions:

Option	Description
Hide all metrics from a query.	Click the options menu ⋮ for the query and click **Hide all series**.
Hide a specific metric.	Go to the query table and click the colored square near the metric name.
Zoom into the plot and change the time range.	Perform one of the following actions: • Visually select the time range by clicking and dragging on the plot horizontally. • Use the menu to select the time range.
Reset the time range.	Click **Reset zoom**.
Display outputs for all queries at a specific point in time.	Hover over the plot at the point you are interested in. The query outputs appear in a pop-up box.
Hide the plot.	Click **Hide graph**.

```
//
// * virt/support/virt-prometheus-queries.adoc
```

15.3.4. Virtualization metrics

The following metric descriptions include example Prometheus Query Language (PromQL) queries. These metrics are not an API and might change between versions. For a complete list of virtualization metrics, see KubeVirt components metrics.



NOTE

The following examples use **topk** queries that specify a time period. If virtual machines (VMs) are deleted during that time period, they can still appear in the query output.

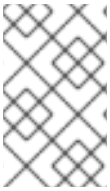
15.3.4.1. vCPU metrics

The following query can identify virtual machines that are waiting for Input/Output (I/O):

kubevirt_vmi_vcpu_wait_seconds_total

Returns the wait time (in seconds) on I/O for vCPUs of a virtual machine. Type: Counter.

A value above '0' means that the vCPU wants to run, but the host scheduler cannot run it yet. This inability to run indicates that there is an issue with I/O.



NOTE

To query the vCPU metric, the **schedstats=enable** kernel argument must first be applied to the **MachineConfig** object. This kernel argument enables scheduler statistics used for debugging and performance tuning and adds a minor additional load to the scheduler.

kubevirt_vmi_vcpu_delay_seconds_total

Returns the cumulative time, in seconds, that a vCPU was enqueued by the host scheduler but could not run immediately. This delay appears to the virtual machine as *steal time*, which is CPU time lost when the host runs other workloads. Steal time can impact performance and often indicates CPU overcommitment or contention on the host. Type: Counter.

Example vCPU delay query

```
irate(kubevirt_vmi_vcpu_delay_seconds_total[5m]) > 0.05 (1)
```

(1) This query returns the average per-second delay over a 5-minute period. A high value may indicate CPU overcommitment or contention on the node.

Example vCPU wait time query

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_vcpu_wait_seconds_total[6m]))) > 0 (1)
```

(1) This query returns the top 3 VMs waiting for I/O at every given moment over a six-minute time period.

15.3.4.2. Network metrics

The following queries can identify virtual machines that are saturating the network:

kubevirt_vmi_network_receive_bytes_total

Returns the total amount of traffic received (in bytes) on the virtual machine's network. Type: Counter.

kubevirt_vmi_network_transmit_bytes_total

Returns the total amount of traffic transmitted (in bytes) on the virtual machine's network. Type: Counter.

Example network traffic query

```
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 This query returns the top 3 VMs transmitting the most network traffic at every given moment over a six-minute time period.

15.3.4.3. Storage metrics

15.3.4.3.1. Storage-related traffic

The following queries can identify VMs that are writing large amounts of data:

kubevirt_vmi_storage_read_traffic_bytes_total

Returns the total amount (in bytes) of the virtual machine's storage-related traffic. Type: Counter.

kubevirt_vmi_storage_write_traffic_bytes_total

Returns the total amount of storage writes (in bytes) of the virtual machine's storage-related traffic. Type: Counter.

Example storage-related traffic query

```
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 This query returns the top 3 VMs performing the most storage traffic at every given moment over a six-minute time period.

15.3.4.3.2. Storage snapshot data

kubevirt_vmsnapshot_disks_restored_from_source

Returns the total number of virtual machine disks restored from the source virtual machine. Type: Gauge.

kubevirt_vmsnapshot_disks_restored_from_source_bytes

Returns the amount of space in bytes restored from the source virtual machine. Type: Gauge.

Examples of storage snapshot data queries

```
kubevirt_vmsnapshot_disks_restored_from_source{vm_name="simple-vm", vm_namespace="default"} 1
```

1 This query returns the total number of virtual machine disks restored from the source virtual machine.

```
kubevirt_vmsnapshot_disks_restored_from_source_bytes{vm_name="simple-vm", vm_namespace="default"} 1
```

-

1 This query returns the amount of space in bytes restored from the source virtual machine.

15.3.4.3.3. I/O performance

The following queries can determine the I/O performance of storage devices:

kubevirt_vmi_storage_iops_read_total

Returns the amount of write I/O operations the virtual machine is performing per second. Type: Counter.

kubevirt_vmi_storage_iops_write_total

Returns the amount of read I/O operations the virtual machine is performing per second. Type: Counter.

Example I/O performance query

```
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 This query returns the top 3 VMs performing the most I/O operations per second at every given moment over a six-minute time period.

15.3.4.4. Guest memory swapping metrics

The following queries can identify which swap-enabled guests are performing the most memory swapping:

kubevirt_vmi_memory_swap_in_traffic_bytes

Returns the total amount (in bytes) of memory the virtual guest is swapping in. Type: Gauge.

kubevirt_vmi_memory_swap_out_traffic_bytes

Returns the total amount (in bytes) of memory the virtual guest is swapping out. Type: Gauge.

Example memory swapping query

```
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 This query returns the top 3 VMs where the guest is performing the most memory swapping at every given moment over a six-minute time period.

NOTE

Memory swapping indicates that the virtual machine is under memory pressure. Increasing the memory allocation of the virtual machine can mitigate this issue.

15.3.4.5. Monitoring AAQ operator metrics

The following metrics are exposed by the Application Aware Quota (AAQ) controller for monitoring resource quotas:

kube_application_aware_resourcequota

Returns the current quota usage and the CPU and memory limits enforced by the AAQ Operator resources. Type: Gauge.

kube_application_aware_resourcequota_creation_timestamp

Returns the time, in UNIX timestamp format, when the AAQ Operator resource is created. Type: Gauge.

15.3.4.6. Live migration metrics

The following metrics can be queried to show live migration status:

kubevirt_vmi_migration_data_processed_bytes

The amount of guest operating system data that has migrated to the new virtual machine (VM). Type: Gauge.

kubevirt_vmi_migration_data_remaining_bytes

The amount of guest operating system data that remains to be migrated. Type: Gauge.

kubevirt_vmi_migration_memory_transfer_rate_bytes

The rate at which memory is becoming dirty in the guest operating system. Dirty memory is data that has been changed but not yet written to disk. Type: Gauge.

kubevirt_vmi_migrations_in_pending_phase

The number of pending migrations. Type: Gauge.

kubevirt_vmi_migrations_in_scheduling_phase

The number of scheduling migrations. Type: Gauge.

kubevirt_vmi_migrations_in_running_phase

The number of running migrations. Type: Gauge.

kubevirt_vmi_migration_succeeded

The number of successfully completed migrations. Type: Gauge.

kubevirt_vmi_migration_failed

The number of failed migrations. Type: Gauge.

15.3.5. Additional resources

- About OpenShift Container Platform monitoring
- Querying Prometheus
- Prometheus query examples

15.4. EXPOSING CUSTOM METRICS FOR VIRTUAL MACHINES

OpenShift Container Platform includes a preconfigured, preinstalled, and self-updating monitoring stack that provides monitoring for core platform components. This monitoring stack is based on the Prometheus monitoring system. Prometheus is a time-series database and a rule evaluation engine for metrics.

In addition to using the OpenShift Container Platform monitoring stack, you can enable monitoring for user-defined projects by using the CLI and query custom metrics that are exposed for virtual machines through the **node-exporter** service.

15.4.1. Configuring the node exporter service

The node-exporter agent is deployed on every virtual machine in the cluster from which you want to collect metrics. Configure the node-exporter agent as a service to expose internal metrics and processes that are associated with virtual machines.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster as a user with **cluster-admin** privileges.
- Create the **cluster-monitoring-config ConfigMap** object in the **openshift-monitoring** project.
- Configure the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project by setting **enableUserWorkload** to **true**.

Procedure

1. Create the **Service** YAML file. In the following example, the file is called **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. The node-exporter service that exposes the metrics from the virtual machines.
2. The namespace where the service is created.
3. The label for the service. The **ServiceMonitor** uses this label to match this service.
4. The name given to the port that exposes metrics on port 9100 for the **ClusterIP** service.
5. The target port used by **node-exporter-service** to listen for requests.
6. The TCP port number of the virtual machine that is configured with the **monitor** label.

7 The label used to match the virtual machine's pods. In this example, any virtual machine's pod with the label **monitor** and a value of **metrics** will be matched.

2. Create the node-exporter service:

```
$ oc create -f node-exporter-service.yaml
```

15.4.2. Configuring a virtual machine with the node exporter service

Download the **node-exporter** file on to the virtual machine. Then, create a **systemd** service that runs the node-exporter service when the virtual machine boots.

Prerequisites

- The pods for the component are running in the **openshift-user-workload-monitoring** project.
- Grant the **monitoring-edit** role to users who need to monitor this user-defined project.

Procedure

1. Log on to the virtual machine.
2. Download the **node-exporter** file on to the virtual machine by using the directory path that applies to the version of **node-exporter** file.

```
$ wget
https://github.com/prometheus/node_exporter/releases/download/<version>/node_exporter-
<version>.linux-<architecture>.tar.gz
```

3. Extract the executable and place it in the **/usr/bin** directory.

```
$ sudo tar xvf node_exporter-<version>.linux-<architecture>.tar.gz \
    --directory /usr/bin --strip 1 "*/node_exporter"
```

4. Create a **node_exporter.service** file in this directory path: **/etc/systemd/system**. This **systemd** service file runs the node-exporter service when the virtual machine reboots.

```
[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. Enable and start the **systemd** service.

```
$ sudo systemctl enable node_exporter.service
$ sudo systemctl start node_exporter.service
```

Verification

- Verify that the node-exporter agent is reporting metrics from the virtual machine.

```
$ curl http://localhost:9100/metrics
```

Example output

```
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
```

15.4.3. Creating a custom monitoring label for virtual machines

To enable queries to multiple virtual machines from a single service, add a custom label in the virtual machine's YAML file.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Access to the web console for stop and restart a virtual machine.

Procedure

1. Edit the **template** spec of your virtual machine configuration file. In this example, the label **monitor** has the value **metrics**.

```
spec:
  template:
    metadata:
      labels:
        monitor: metrics
```

2. Stop and restart the virtual machine to create a new pod with the label name given to the **monitor** label.

15.4.3.1. Querying the node-exporter service for metrics

Metrics are exposed for virtual machines through an HTTP service endpoint under the /**metrics** canonical name. When you query for metrics, Prometheus directly scrapes the metrics from the metrics endpoint exposed by the virtual machines and presents these metrics for viewing.

Prerequisites

- You have access to the cluster as a user with **cluster-admin** privileges or the **monitoring-edit** role.
- You have enabled monitoring for the user-defined project by configuring the node-exporter service.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Obtain the HTTP service endpoint by specifying the namespace for the service:

```
$ oc get service -n <namespace> <node-exporter-service>
```

2. To list all available metrics for the node-exporter service, query the **metrics** resource.

```
$ curl http://<172.30.226.162:9100>/metrics | grep -vE "^#|^$"
```

Example output

```
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
```

15.4.4. Creating a ServiceMonitor resource for the node exporter service

You can use a Prometheus client library and scrape metrics from the /**metrics** endpoint to access and view the metrics exposed by the node-exporter service. Use a **ServiceMonitor** custom resource definition (CRD) to monitor the node exporter service.

Prerequisites

- You have access to the cluster as a user with **cluster-admin** privileges or the **monitoring-edit** role.
- You have enabled monitoring for the user-defined project by configuring the node-exporter service.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a YAML file for the **ServiceMonitor** resource configuration. In this example, the service monitor matches any service with the label **metrics** and queries the **exmet** port every 30 seconds.

```
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) The name of the **ServiceMonitor**.

(2) The namespace where the **ServiceMonitor** is created.

3 The interval at which the port will be queried.

4 The name of the port that is queried every 30 seconds

2. Create the **ServiceMonitor** configuration for the node-exporter service.

```
$ oc create -f node-exporter-metrics-monitor.yaml
```

15.4.4.1. Accessing the node exporter service outside the cluster

You can access the node-exporter service outside the cluster and view the exposed metrics.

Prerequisites

- You have access to the cluster as a user with **cluster-admin** privileges or the **monitoring-edit** role.
- You have enabled monitoring for the user-defined project by configuring the node-exporter service.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Expose the node-exporter service.

```
$ oc expose service -n <namespace> <node_exporter_service_name>
```

2. Obtain the FQDN (Fully Qualified Domain Name) for the route.

```
$ oc get route -o=custom-columns=NAME:.metadata.name,DNS:.spec.host
```

Example output

```
NAME                 DNS
node-exporter-service   node-exporter-service-dynamation.apps.cluster.example.org
```

3. Use the **curl** command to display metrics for the node-exporter service.

```
$ curl -s http://node-exporter-service-dynamation.apps.cluster.example.org/metrics
```

Example output

```
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
```

15.4.5. Additional resources

- Core platform monitoring first steps
- Enabling monitoring for user-defined projects
- Accessing metrics as a developer
- Reviewing monitoring dashboards as a developer
- Monitoring application health by using health checks
- Creating and using config maps
- Controlling virtual machine states

15.5. EXPOSING DOWNWARD METRICS FOR VIRTUAL MACHINES

As an administrator, you can expose a limited set of host and virtual machine (VM) metrics to a guest VM by first enabling a **downwardMetrics** feature gate and then configuring a **downwardMetrics** device.

Users can view the metrics results by using the command line or the **vm-dump-metrics tool**.

NOTE

On Red Hat Enterprise Linux (RHEL) 9, use the command line to view downward metrics. See Viewing downward metrics by using the command line .

The vm-dump-metrics tool is not supported on the Red Hat Enterprise Linux (RHEL) 9 platform.

15.5.1. Enabling or disabling the downwardMetrics feature gate

You can enable or disable the **downwardMetrics** feature gate by performing either of the following actions:

- Editing the HyperConverged custom resource (CR) in your default editor
- Using the command line

15.5.1.1. Enabling or disabling the downward metrics feature gate in a YAML file

To expose downward metrics for a host virtual machine, you can enable the **downwardMetrics** feature gate by editing a YAML file.

Prerequisites

- You must have administrator privileges to enable the feature gate.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Open the HyperConverged custom resource (CR) in your default editor by running the following command:

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

2. Choose to enable or disable the downwardMetrics feature gate as follows:

   - To enable the **downwardMetrics** feature gate, add and then set **spec.featureGates.downwardMetrics** to **true**. For example:

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

   - To disable the **downwardMetrics** feature gate, set **spec.featureGates.downwardMetrics** to **false**. For example:

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

15.5.1.2. Enabling or disabling the downward metrics feature gate from the CLI

To expose downward metrics for a host virtual machine, you can enable the **downwardMetrics** feature gate by using the command line.

Prerequisites

- You must have administrator privileges to enable the feature gate.
- You have installed the OpenShift CLI (**oc**).

Procedure

- Choose to enable or disable the **downwardMetrics** feature gate as follows:
  - Enable the **downwardMetrics** feature gate by running the command shown in the following example:

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

  - Disable the **downwardMetrics** feature gate by running the command shown in the following example:

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

15.5.2. Configuring a downward metrics device

You enable the capturing of downward metrics for a host VM by creating a configuration file that includes a **downwardMetrics** device. Adding this device establishes that the metrics are exposed through a **virtio-serial** port.

Prerequisites

- You must first enable the **downwardMetrics** feature gate.

Procedure

- Edit or create a YAML file that includes a **downwardMetrics** device, as shown in the following example:

  Example downwardMetrics configuration file

```
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: {}
  instancetype:
    name: u1.medium
  runStrategy: Always
  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 The **downwardMetrics** device.

2 The password for the **fedora** user.

15.5.3. Viewing downward metrics

You can view downward metrics by using either of the following options:

- The command-line interface (CLI)
- The **vm-dump-metrics** tool

NOTE

On Red Hat Enterprise Linux (RHEL) 9, use the command line to view downward metrics. The vm-dump-metrics tool is not supported on the Red Hat Enterprise Linux (RHEL) 9 platform.

15.5.3.1. Viewing downward metrics by using the CLI

You can view downward metrics by entering a command from inside a guest virtual machine (VM).

Procedure

- Run the following commands:

```
$ 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
```

15.5.3.2. Viewing downward metrics by using the vm-dump-metrics tool

To view downward metrics, install the **vm-dump-metrics** tool and then use the tool to expose the metrics results.

NOTE

On Red Hat Enterprise Linux (RHEL) 9, use the command line to view downward metrics. The vm-dump-metrics tool is not supported on the Red Hat Enterprise Linux (RHEL) 9 platform.

Procedure

1. Install the **vm-dump-metrics** tool by running the following command:

```
$ sudo dnf install -y vm-dump-metrics
```

2. Retrieve the metrics results by running the following command:

```
$ sudo vm-dump-metrics
```

Example output

```
<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>
```

15.6. VIRTUAL MACHINE HEALTH CHECKS

You can configure virtual machine (VM) health checks by defining readiness and liveness probes in the **VirtualMachine** resource.

15.6.1. About readiness and liveness probes

Use readiness and liveness probes to detect and handle unhealthy virtual machines (VMs). You can include one or more probes in the specification of the VM to ensure that traffic does not reach a VM that is not ready for it and that a new VM is created when a VM becomes unresponsive.

A *readiness probe* determines whether a VM is ready to accept service requests. If the probe fails, the VM is removed from the list of available endpoints until the VM is ready.

A *liveness probe* determines whether a VM is responsive. If the probe fails, the VM is deleted and a new VM is created to restore responsiveness.

You can configure readiness and liveness probes by setting the **spec.readinessProbe** and the **spec.livenessProbe** fields of the **VirtualMachine** object. These fields support the following tests:

HTTP GET

The probe determines the health of the VM by using a web hook. The test is successful if the HTTP response code is between 200 and 399. You can use an HTTP GET test with applications that return HTTP status codes when they are completely initialized.

TCP socket

The probe attempts to open a socket to the VM. The VM is only considered healthy if the probe can establish a connection. You can use a TCP socket test with applications that do not start listening until initialization is complete.

Guest agent ping

The probe uses the **guest-ping** command to determine if the QEMU guest agent is running on the virtual machine.

15.6.1.1. Defining an HTTP readiness probe

Define an HTTP readiness probe by setting the **spec.readinessProbe.httpGet** field of the virtual machine (VM) configuration.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Include details of the readiness probe in the VM configuration file.

Sample readiness probe with an HTTP GET test

```
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. The HTTP GET request to perform to connect to the VM.
2. The port of the VM that the probe queries. In the above example, the probe queries port 1500.
3. The path to access on the HTTP server. In the above example, if the handler for the server's /healthz path returns a success code, the VM is considered to be healthy. If the handler returns a failure code, the VM is removed from the list of available endpoints.
4. The time, in seconds, after the VM starts before the readiness probe is initiated.

5 The delay, in seconds, between performing probes. The default delay is 10 seconds. This value must be greater than **timeoutSeconds**.

6 The number of seconds of inactivity after which the probe times out and the VM is assumed to have failed. The default value is 1. This value must be lower than **periodSeconds**.

7 The number of times that the probe is allowed to fail. The default is 3. After the specified number of attempts, the pod is marked **Unready**.

8 The number of times that the probe must report success, after a failure, to be considered successful. The default is 1.

2. Create the VM by running the following command:

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

15.6.1.2. Defining a TCP readiness probe

Define a TCP readiness probe by setting the **spec.readinessProbe.tcpSocket** field of the virtual machine (VM) configuration.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Include details of the TCP readiness probe in the VM configuration file.

Sample readiness probe with a TCP socket test

```
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 The time, in seconds, after the VM starts before the readiness probe is initiated.

2 The delay, in seconds, between performing probes. The default delay is 10 seconds. This value must be greater than **timeoutSeconds**.

3 The TCP action to perform.

4 The port of the VM that the probe queries.

5 The number of seconds of inactivity after which the probe times out and the VM is assumed to have failed. The default value is 1. This value must be lower than **periodSeconds**.

2. Create the VM by running the following command:

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

15.6.1.3. Defining an HTTP liveness probe

Define an HTTP liveness probe by setting the **spec.livenessProbe.httpGet** field of the virtual machine (VM) configuration. You can define both HTTP and TCP tests for liveness probes in the same way as readiness probes. This procedure configures a sample liveness probe with an HTTP GET test.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Include details of the HTTP liveness probe in the VM configuration file.

Sample liveness probe with an HTTP GET test

```
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. The time, in seconds, after the VM starts before the liveness probe is initiated.
2. The delay, in seconds, between performing probes. The default delay is 10 seconds. This value must be greater than **timeoutSeconds**.
3. The HTTP GET request to perform to connect to the VM.
4. The port of the VM that the probe queries. In the above example, the probe queries port 1500. The VM installs and runs a minimal HTTP server on port 1500 via cloud-init.
5. The path to access on the HTTP server. In the above example, if the handler for the server's /**healthz** path returns a success code, the VM is considered to be healthy. If the handler returns a failure code, the VM is deleted and a new VM is created.
6. The number of seconds of inactivity after which the probe times out and the VM is assumed to have failed. The default value is 1. This value must be lower than **periodSeconds**.

2. Create the VM by running the following command:

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

15.6.2. Defining a watchdog

You can define a watchdog to monitor the health of the guest operating system by performing the following steps:

1. Configure a watchdog device for the virtual machine (VM).
2. Install the watchdog agent on the guest.

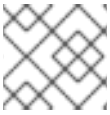
The watchdog device monitors the agent and performs one of the following actions if the guest operating system is unresponsive:

- **poweroff**: The VM powers down immediately. If **spec.runStrategy** is not set to **manual**, the VM reboots.
- **reset**: The VM reboots in place and the guest operating system cannot react.

NOTE

The reboot time might cause liveness probes to time out. If cluster-level protections detect a failed liveness probe, the VM might be forcibly rescheduled, increasing the reboot time.

- **shutdown**: The VM gracefully powers down by stopping all services.



NOTE

Watchdog is not available for Windows VMs.

15.6.2.1. Configuring a watchdog device for the virtual machine

You configure a watchdog device for the virtual machine (VM).

Prerequisites

- For **x86** systems, the VM must use a kernel that works with the **i6300esb** watchdog device. If you use **s390x** architecture, the kernel must be enabled for **diag288**. Red Hat Enterprise Linux (RHEL) images support **i6300esb** and **diag288**.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **YAML** file with the following contents:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: <vm-label>
  name: <vm-name>
spec:
  runStrategy: Halted
  template:
    metadata:
      labels:
        kubevirt.io/vm: <vm-label>
    spec:
      domain:
        devices:
          watchdog:
            name: <watchdog>
            <watchdog-device-model>: 1
              action: "poweroff" 2
# ...
```

1. The watchdog device model to use. For **x86** specify **i6300esb**. For **s390x** specify **diag288**.
2. Specify **poweroff**, **reset**, or **shutdown**. The **shutdown** action requires that the guest virtual machine is responsive to ACPI signals. Therefore, using **shutdown** is not recommended.

   The example above configures the watchdog device on a VM with the **poweroff** action and exposes the device as **/dev/watchdog**.

   This device can now be used by the watchdog binary.

2. Apply the YAML file to your cluster by running the following command:

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

Verification

IMPORTANT

This procedure is provided for testing watchdog functionality only and must not be run on production machines.

1. Run the following command to verify that the VM is connected to the watchdog device:

```
$ lspci | grep watchdog -i
```

2. Run one of the following commands to confirm the watchdog is active:

- Trigger a kernel panic:

```
# echo c > /proc/sysrq-trigger
```

- Stop the watchdog service:

```
# pkill -9 watchdog
```

15.6.2.2. Installing the watchdog agent on the guest

You install the watchdog agent on the guest and start the **watchdog** service.

Procedure

1. Log in to the virtual machine as root user.
2. This step is only required when installing on IBM Z® (**s390x**). Enable **watchdog** by running the following command:

```
# modprobe diag288_wdt
```

3. Verify that the **/dev/watchdog** file path is present in the VM by running the following command:

```
# ls /dev/watchdog
```

4. Install the **watchdog** package and its dependencies:

```
# yum install watchdog
```

5. Uncomment the following line in the **/etc/watchdog.conf** file and save the changes:

```
#watchdog-device = /dev/watchdog
```

6. Enable the **watchdog** service to start on boot:

```
# systemctl enable --now watchdog.service
```

15.6.3. Defining a guest agent ping probe

Define a guest agent ping probe by setting the **spec.readinessProbe.guestAgentPing** field of the virtual machine (VM) configuration.

Prerequisites

- The QEMU guest agent must be installed and enabled on the virtual machine.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Include details of the guest agent ping probe in the VM configuration file. For example:

Sample guest agent ping probe

```
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. The guest agent ping probe to connect to the VM.
2. Optional: The time, in seconds, after the VM starts before the guest agent probe is initiated.
3. Optional: The delay, in seconds, between performing probes. The default delay is 10 seconds. This value must be greater than **timeoutSeconds**.
4. Optional: The number of seconds of inactivity after which the probe times out and the VM is assumed to have failed. The default value is 1. This value must be lower than **periodSeconds**.
5. Optional: The number of times that the probe is allowed to fail. The default is 3. After the specified number of attempts, the pod is marked **Unready**.
6. Optional: The number of times that the probe must report success, after a failure, to be considered successful. The default is 1.

2. Create the VM by running the following command:

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

15.6.4. Additional resources

- Monitoring application health by using health checks

15.7. OPENSHIFT VIRTUALIZATION RUNBOOKS

To diagnose and resolve issues that trigger OpenShift Virtualization alerts, follow the procedures in the runbooks for the OpenShift Virtualization Operator. Triggered OpenShift Virtualization alerts can be viewed in the main **Observe → Alerts** tab in the web console, and also in the **Virtualization → Overview** tab.

Runbooks for the OpenShift Virtualization Operator are maintained in the openshift/runbooks Git repository, and you can view them on GitHub.

15.7.1. CDIDataImportCronOutdated

- View the runbook for the **CDIDataImportCronOutdated** alert.

15.7.2. CDIDataVolumeUnusualRestartCount

- View the runbook for the **CDIDataVolumeUnusualRestartCount** alert.

15.7.3. CDIDefaultStorageClassDegraded

- View the runbook for the **CDIDefaultStorageClassDegraded** alert.

15.7.4. CDIMultipleDefaultVirtStorageClasses

- View the runbook for the **CDIMultipleDefaultVirtStorageClasses** alert.

15.7.5. CDINoDefaultStorageClass

- View the runbook for the **CDINoDefaultStorageClass** alert.

15.7.6. CDINotReady

- View the runbook for the **CDINotReady** alert.

15.7.7. CDIOperatorDown

- View the runbook for the **CDIOperatorDown** alert.

15.7.8. CDIStorageProfilesIncomplete

- View the runbook for the **CDIStorageProfilesIncomplete** alert.

15.7.9. CnaoDown

- View the runbook for the **CnaoDown** alert.

15.7.10. CnaoNMstateMigration

- View the runbook for the **CnaoNMstateMigration** alert.

15.7.11. HAControlPlaneDown

- View the runbook for the **HAControlPlaneDown** alert.

15.7.12. HCOInstallationIncomplete

- View the runbook for the **HCOInstallationIncomplete** alert.

15.7.13. HCOMisconfiguredDescheduler

- View the runbook for the **HCOMisconfiguredDescheduler** alert.

15.7.14. HPPNotReady

- View the runbook for the **HPPNotReady** alert.

15.7.15. HPPOperatorDown

- View the runbook for the **HPPOperatorDown** alert.

15.7.16. HPPSharingPoolPathWithOS

- View the runbook for the **HPPSharingPoolPathWithOS** alert.

15.7.17. HighCPUWorkload

- View the runbook for the **HighCPUWorkload** alert.

15.7.18. KubemacpoolDown

- View the runbook for the **KubemacpoolDown** alert.

15.7.19. KubeMacPoolDuplicateMacsFound

- View the runbook for the **KubeMacPoolDuplicateMacsFound** alert.

15.7.20. KubeVirtComponentExceedsRequestedCPU

- The **KubeVirtComponentExceedsRequestedCPU** alert is deprecated.

15.7.21. KubeVirtComponentExceedsRequestedMemory

- The **KubeVirtComponentExceedsRequestedMemory** alert is deprecated.

15.7.22. KubeVirtCRModified

- View the runbook for the **KubeVirtCRModified** alert.

15.7.23. KubeVirtDeprecatedAPIRequested

- View the runbook for the **KubeVirtDeprecatedAPIRequested** alert.

15.7.24. KubeVirtNoAvailableNodesToRunVMs

- View the runbook for the **KubeVirtNoAvailableNodesToRunVMs** alert.

15.7.25. KubevirtVmHighMemoryUsage

- View the runbook for the **KubevirtVmHighMemoryUsage** alert.

15.7.26. KubeVirtVMIExcessiveMigrations

- View the runbook for the **KubeVirtVMIExcessiveMigrations** alert.

15.7.27. LowKVMNodesCount

- View the runbook for the **LowKVMNodesCount** alert.

15.7.28. LowReadyVirtControllersCount

- View the runbook for the **LowReadyVirtControllersCount** alert.

15.7.29. LowReadyVirtOperatorsCount

- View the runbook for the **LowReadyVirtOperatorsCount** alert.

15.7.30. LowVirtAPICount

- View the runbook for the **LowVirtAPICount** alert.

15.7.31. LowVirtControllersCount

- View the runbook for the **LowVirtControllersCount** alert.

15.7.32. LowVirtOperatorCount

- View the runbook for the **LowVirtOperatorCount** alert.

15.7.33. NetworkAddonsConfigNotReady

- View the runbook for the **NetworkAddonsConfigNotReady** alert.

15.7.34. NoLeadingVirtOperator

- View the runbook for the **NoLeadingVirtOperator** alert.

15.7.35. NoReadyVirtController

- View the runbook for the **NoReadyVirtController** alert.

15.7.36. NoReadyVirtOperator

- View the runbook for the **NoReadyVirtOperator** alert.

15.7.37. NodeNetworkInterfaceDown

- View the runbook for the **NodeNetworkInterfaceDown** alert.

15.7.38. OperatorConditionsUnhealthy

- The **OperatorConditionsUnhealthy** alert is deprecated.

15.7.39. OrphanedVirtualMachineInstances

- View the runbook for the **OrphanedVirtualMachineInstances** alert.

15.7.40. OutdatedVirtualMachineInstanceWorkloads

- View the runbook for the **OutdatedVirtualMachineInstanceWorkloads** alert.

15.7.41. SingleStackIPv6Unsupported

- View the runbook for the **SingleStackIPv6Unsupported** alert.

15.7.42. SSPCommonTemplatesModificationReverted

- View the runbook for the **SSPCommonTemplatesModificationReverted** alert.

15.7.43. SSPDown

- View the runbook for the **SSPDown** alert.

15.7.44. SSPFailingToReconcile

- View the runbook for the **SSPFailingToReconcile** alert.

15.7.45. SSPHighRateRejectedVms

- View the runbook for the **SSPHighRateRejectedVms** alert.

15.7.46. SSPOperatorDown

- View the runbook for the **SSPOperatorDown** alert.

15.7.47. SSPTemplateValidatorDown

- View the runbook for the **SSPTemplateValidatorDown** alert.

15.7.48. UnsupportedHCOModification

- View the runbook for the **UnsupportedHCOModification** alert.

15.7.49. VirtAPIDown

- View the runbook for the **VirtAPIDown** alert.

15.7.50. VirtApiRESTErrorsBurst

- View the runbook for the **VirtApiRESTErrorsBurst** alert.

15.7.51. VirtApiRESTErrorsHigh

- View the runbook for the **VirtApiRESTErrorsHigh** alert.

15.7.52. VirtControllerDown

- View the runbook for the **VirtControllerDown** alert.

15.7.53. VirtControllerRESTErrorsBurst

- View the runbook for the **VirtControllerRESTErrorsBurst** alert.

15.7.54. VirtControllerRESTErrorsHigh

- View the runbook for the **VirtControllerRESTErrorsHigh** alert.

15.7.55. VirtHandlerDaemonSetRolloutFailing

- View the runbook for the **VirtHandlerDaemonSetRolloutFailing** alert.

15.7.56. VirtHandlerRESTErrorsBurst

- View the runbook for the **VirtHandlerRESTErrorsBurst** alert.

15.7.57. VirtHandlerRESTErrorsHigh

- View the runbook for the **VirtHandlerRESTErrorsHigh** alert.

15.7.58. VirtOperatorDown

- View the runbook for the **VirtOperatorDown** alert.

15.7.59. VirtOperatorRESTErrorsBurst

- View the runbook for the **VirtOperatorRESTErrorsBurst** alert.

15.7.60. VirtOperatorRESTErrorsHigh

- View the runbook for the **VirtOperatorRESTErrorsHigh** alert.

15.7.61. VirtualMachineCRCErrors

- The **VirtualMachineCRCErrors** alert is deprecated.

The alert is now called **VMStorageClassWarning**.

15.7.62. VMCannotBeEvicted

- View the runbook for the **VMCannotBeEvicted** alert.

15.7.63. VMStorageClassWarning

- View the runbook for the **VMStorageClassWarning** alert.

CHAPTER 16. SUPPORT

16.1. SUPPORT OVERVIEW

You can request assistance from Red Hat Support, report bugs, collect data about your environment, and monitor the health of your cluster and virtual machines (VMs) with the following tools.

16.1.1. Opening support tickets

If you have encountered an issue that requires immediate assistance from Red Hat Support, you can submit a support case.

To report a bug, you can create a Jira issue directly.

16.1.1.1. Submitting a support case

To request support from Red Hat Support, follow the instructions for submitting a support case.

It is helpful to collect debugging data to include with your support request.

16.1.1.1.1. Collecting data for Red Hat Support

You can gather debugging information by performing the following steps:

Collecting data about your environment

Configure Prometheus and Alertmanager and collect **must-gather** data for OpenShift Container Platform and OpenShift Virtualization.

must-gather tool for OpenShift Virtualization

Configure and use the **must-gather** tool.

Collecting data about VMs

Collect **must-gather** data and memory dumps from VMs.

16.1.1.2. Creating a Jira issue

To report a bug, you can create a Jira issue directly by filling out the form on the **Create Issue** page.

16.1.2. Web console monitoring

You can monitor the health of your cluster and VMs by using the OpenShift Container Platform web console. The web console displays resource usage, alerts, events, and trends for your cluster and for OpenShift Virtualization components and resources.

Table 16.1. Web console pages for monitoring and troubleshooting

Page	Description
Overview page	Cluster details, status, alerts, inventory, and resource usage

Page	Description
Virtualization → Overview tab	OpenShift Virtualization resources, usage, alerts, and status
Virtualization → Top consumers tab	Top consumers of CPU, memory, and storage
Virtualization → Migrations tab	Progress of live migrations
Virtualization → VirtualMachines tab	CPU, memory, and storage usage summary
Virtualization → VirtualMachines → VirtualMachine details → Metrics tab	VM resource usage, storage, network, and migration
Virtualization → VirtualMachines → VirtualMachine details → Events tab	List of VM events
Virtualization → VirtualMachines → VirtualMachine details → Diagnostics tab	VM status conditions and volume snapshot status

16.2. COLLECTING DATA FOR RED HAT SUPPORT

When you submit a support case to Red Hat Support, it is helpful to provide debugging information for OpenShift Container Platform and OpenShift Virtualization by using the following tools:

must-gather tool

The **must-gather** tool collects diagnostic information, including resource definitions and service logs.

Prometheus

Prometheus is a time-series database and a rule evaluation engine for metrics. Prometheus sends alerts to Alertmanager for processing.

Alertmanager

The Alertmanager service handles alerts received from Prometheus. The Alertmanager is also responsible for sending the alerts to external notification systems. For information about the OpenShift Container Platform monitoring stack, see About OpenShift Container Platform monitoring.

16.2.1. Collecting data about your environment

Collecting data about your environment minimizes the time required to analyze and determine the root cause.

Prerequisites

- Set the retention time for Prometheus metrics data to a minimum of seven days.
- Configure the Alertmanager to capture relevant alerts and to send alert notifications to a dedicated mailbox so that they can be viewed and persisted outside the cluster.
- Record the exact number of affected nodes and virtual machines.

Procedure

1. Collect must-gather data for the cluster.
2. Collect must-gather data for Red Hat OpenShift Data Foundation , if necessary.
3. Collect must-gather data for OpenShift Virtualization.
4. Collect Prometheus metrics for the cluster.

16.2.2. Collecting data about virtual machines

Collecting data about malfunctioning virtual machines (VMs) minimizes the time required to analyze and determine the root cause.

Prerequisites

- Linux VMs: Install the latest QEMU guest agent .
- Windows VMs:
  - Record the Windows patch update details.
  - Install the latest VirtIO drivers.
  - Install the latest QEMU guest agent .
  - If Remote Desktop Protocol (RDP) is enabled, connect by using the desktop viewer to determine whether there is a problem with the connection software.

Procedure

1. Collect must-gather data for the VMs using the **/usr/bin/gather** script.
2. Collect screenshots of VMs that have crashed *before* you restart them.
3. Collect memory dumps from VMs *before* remediation attempts.
4. Record factors that the malfunctioning VMs have in common. For example, the VMs have the same host or network.

16.2.3. Using the must-gather tool for OpenShift Virtualization

You can collect data about OpenShift Virtualization resources by running the **must-gather** command with the OpenShift Virtualization image.

The default data collection includes information about the following resources:

- OpenShift Virtualization Operator namespaces, including child objects
- OpenShift Virtualization custom resource definitions
- Namespaces that contain virtual machines
- Basic virtual machine definitions

Instance types information is not currently collected by default; you can, however, run a command to optionally collect it.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

- Run the following command to collect data about OpenShift Virtualization:

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.20.0 \
  -- /usr/bin/gather
```

16.2.3.1. must-gather tool options

You can run the **oc adm must-gather** command to collect **must gather** images for all the Operators and products deployed on your cluster without the need to explicitly specify the required images. Alternatively, you can specify a combination of scripts and environment variables for the following options:

- Collecting detailed virtual machine (VM) information from a namespace
- Collecting detailed information about specified VMs
- Collecting image, image-stream, and image-stream-tags information
- Limiting the maximum number of parallel processes used by the **must-gather** tool

16.2.3.1.1. Parameters

Environment variables

You can specify environment variables for a compatible script.

NS=<namespace_name>

Collect virtual machine information, including **virt-launcher** pod details, from the namespace that you specify. The **VirtualMachine** and **VirtualMachineInstance** CR data is collected for all namespaces.

VM=<vm_name>

Collect details about a particular virtual machine. To use this option, you must also specify a namespace by using the **NS** environment variable.

PROS=<number_of_processes>

Modify the maximum number of parallel processes that the **must-gather** tool uses. The default value is **5**.

IMPORTANT

Using too many parallel processes can cause performance issues. Increasing the maximum number of parallel processes is not recommended.

Scripts

Each script is compatible only with certain environment variable combinations.

/usr/bin/gather

Use the default **must-gather** script, which collects cluster data from all namespaces and includes only basic VM information. This script is compatible only with the **PROS** variable.

/usr/bin/gather --vms_details

Collect VM log files, VM definitions, control-plane logs, and namespaces that belong to OpenShift Virtualization resources. Specifying namespaces includes their child objects. If you use this parameter without specifying a namespace or VM, the **must-gather** tool collects this data for all VMs in the cluster. This script is compatible with all environment variables, but you must specify a namespace if you use the **VM** variable.

/usr/bin/gather --images

Collect image, image-stream, and image-stream-tags custom resource information. This script is compatible only with the **PROS** variable.

/usr/bin/gather --instancetypes

Collect instance types information. This information is not currently collected by default; you can, however, optionally collect it.

16.2.3.1.2. Usage and examples

Environment variables are optional. You can run a script by itself or with one or more compatible environment variables.

Table 16.2. Compatible parameters

Script	Compatible environment variable
/usr/bin/gather	* **PROS=<number_of_processes>**
/usr/bin/gather --vms_details	* For a namespace: **NS=<namespace_name>** * For a VM: **VM=<vm_name> NS=<namespace_name>** * **PROS=<number_of_processes>**
/usr/bin/gather --images	* **PROS=<number_of_processes>**

Syntax

To collect **must-gather** logs for all Operators and products on your cluster in a single pass, run the following command:

```
$ oc adm must-gather --all-images
```

If you need to pass additional parameters to individual **must-gather** images, use the following command:

```
$ oc adm must-gather \
```

```
--image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.20.0 \
-- <environment_variable_1> <environment_variable_2> <script_name>
```

Default data collection parallel processes

By default, five processes run in parallel.

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.20.0 \
  -- PROS=5 /usr/bin/gather 1
```

1 You can modify the number of parallel processes by changing the default.

Detailed VM information

The following command collects detailed VM information for the **my-vm** VM in the **mynamespace** namespace:

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.20.0 \
  -- NS=mynamespace VM=my-vm /usr/bin/gather --vms_details 1
```

1 The **NS** environment variable is mandatory if you use the **VM** environment variable.

Image, image-stream, and image-stream-tags information

The following command collects image, image-stream, and image-stream-tags information from the cluster:

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.20.0 \
  /usr/bin/gather --images
```

Instance types information

The following command collects instance types information from the cluster:

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.20.0 \
  /usr/bin/gather --instancetypes
```

16.2.4. Generating a VM memory dump

When a virtual machine (VM) terminates unexpectedly, you can use the **virtctl memory-dump** to generate a memory dump command to output a VM memory dump and save it on a persistent volume claim (PVC). Afterwards, you can analyze the memory dump to diagnose and troubleshoot issues on the VM.

Prerequisites

- The hot plug feature gate is enabled in the **HyperConverged** custom resource. To do so, run the following command:

  -

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

- Optional: You have an existing PVC on which you want to save the memory dump.
  - The PVC volume mode must be **FileSystem**.
  - The PVC must be large enough to contain the memory dump.
    The formula for calculating the PVC size is **(VMMemorySize + 100Mi) * FileSystemOverhead**, where **100Mi** is the memory dump overhead, and **FileSystemOverhead** is defined in the **HCO** object.

Procedure

1. Create a memory dump of the required VM:
   - If you have an existing PVC selected on which you want to save the memory dump:

     ```
     $ virtctl memory-dump get <vm_name> --claim-name=<pvc_name>
     ```

   - If you want to create a new PVC for the memory dump:

     ```
     $ virtctl memory-dump get <vm_name> --claim-name=<new_pvc_name> --create-claim
     ```

2. Download the memory dump:

   ```
   $ virtctl memory-dump download <vm_name> --output=<output_file>
   ```

3. Attach the memory dump to a Red Hat Support case.
   Alternatively, you can inspect the memory dump, for example by using the volatility3 tool.

4. Optional: Remove the memory dump:

   ```
   $ virtctl memory-dump remove <vm_name>
   ```

16.2.5. Additional resources

- VM support overview
- How to provide log files to Red Hat Support (Red Hat Knowledgebase)

16.3. TROUBLESHOOTING

OpenShift Virtualization provides tools and logs for troubleshooting virtual machines (VMs) and virtualization components.

You can troubleshoot OpenShift Virtualization components by using the tools provided in the web console or by using the **oc** CLI tool.

16.3.1. Events

OpenShift Container Platform events are records of important life-cycle information and are useful for monitoring and troubleshooting virtual machine, namespace, and resource issues.

- VM events: Navigate to the **Events** tab of the **VirtualMachine details** page in the web console.

  Namespace events

  You can view namespace events by running the following command:

  ```
  $ oc get events -n <namespace>
  ```

  See the list of events for details about specific events.

  Resource events

  You can view resource events by running the following command:

  ```
  $ oc describe <resource> <resource_name>
  ```

16.3.2. Pod logs

You can view logs for OpenShift Virtualization pods by using the web console or the CLI. You can also view aggregated logs by using the LokiStack in the web console.

16.3.2.1. Configuring OpenShift Virtualization pod log verbosity

You can configure the verbosity level of OpenShift Virtualization pod logs by editing the **HyperConverged** custom resource (CR).

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. To set log verbosity for specific components, open the **HyperConverged** CR in your default text editor by running the following command:

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

2. Set the log level for one or more components by editing the **spec.logVerbosityConfig** stanza. For example:

   ```
   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. The log verbosity value must be an integer in the range **1–9**, where a higher number indicates a more detailed log. In this example, the **virtAPI** component logs are exposed if their priority level is **5** or higher.

3. Apply your changes by saving and exiting the editor.

16.3.2.2. Viewing virt-launcher pod logs with the web console

You can view the **virt-launcher** pod logs for a virtual machine by using the OpenShift Container Platform web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines**.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. On the **General** tile, click the pod name to open the **Pod details** page.
4. Click the **Logs** tab to view the logs.

16.3.2.3. Viewing OpenShift Virtualization pod logs with the CLI

You can view logs for the OpenShift Virtualization pods by using the **oc** CLI tool.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. View a list of pods in the OpenShift Virtualization namespace by running the following command:

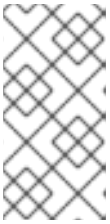
```
$ oc get pods -n openshift-cnv
```

Example 16.1. Example output

```
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. View the pod log by running the following command:

```
$ oc logs -n openshift-cnv <pod_name>
```



NOTE

If a pod fails to start, you can use the **--previous** option to view logs from the last attempt.

To monitor log output in real time, use the **-f** option.

Example 16.2. Example output


```
{"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"}
```


16.3.3. Guest system logs

Viewing the boot logs of VM guests can help diagnose issues. You can configure access to guests' logs and view them by using either the OpenShift Container Platform web console or the **oc** CLI.

This feature is disabled by default. If a VM does not explicitly have this setting enabled or disabled, it inherits the cluster-wide default setting.

IMPORTANT

If sensitive information such as credentials or other personally identifiable information (PII) is written to the serial console, it is logged with all other visible text. Red Hat recommends using SSH to send sensitive data instead of the serial console.

16.3.3.1. Enabling default access to VM guest system logs with the web console

You can enable default access to VM guest system logs by using the web console.

Procedure

1. From the side menu, click **Virtualization → Overview**.
2. Click the **Settings** tab.
3. Click **Cluster → Guest management**.
4. Set **Enable guest system log access** to on.

16.3.3.2. Enabling default access to VM guest system logs with the CLI

You can enable default access to VM guest system logs by editing the **HyperConverged** custom resource (CR).

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Open the **HyperConverged** CR in your default editor by running the following command:

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

2. Update the **disableSerialConsoleLog** value. For example:

```
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  virtualMachineOptions:
    disableSerialConsoleLog: true 1
#...
```

1. Set the value of **disableSerialConsoleLog** to **false** if you want serial console access to be enabled on VMs by default.

16.3.3.3. Setting guest system log access for a single VM with the web console

You can configure access to VM guest system logs for a single VM by using the web console. This setting takes precedence over the cluster-wide default configuration.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Configuration** tab.
4. Set **Guest system log access** to on or off.

16.3.3.4. Setting guest system log access for a single VM with the CLI

You can configure access to VM guest system logs for a single VM by editing the **VirtualMachine** CR. This setting takes precedence over the cluster-wide default configuration.

Prerequisites

- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the virtual machine manifest by running the following command:

```
$ oc edit vm <vm_name>
```

2. Update the value of the **logSerialConsole** field. For example:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          logSerialConsole: true 1
#...
```

1 To enable access to the guest's serial console log, set the **logSerialConsole** value to **true**.

3. Apply the new configuration to the VM by running the following command:

```
$ oc apply vm <vm_name>
```

4. Optional: If you edited a running VM, restart the VM to apply the new configuration. For example:

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

16.3.3.5. Viewing guest system logs with the web console

You can view the serial console logs of a virtual machine (VM) guest by using the web console.

Prerequisites

- Guest system log access is enabled.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.

2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Diagnostics** tab.
4. Click **Guest system logs** to load the serial console.

16.3.3.6. Viewing guest system logs with the CLI

You can view the serial console logs of a VM guest by running the **oc logs** command.

Prerequisites

- Guest system log access is enabled.
- You have installed the OpenShift CLI (**oc**).

Procedure

- View the logs by running the following command, substituting your own values for **<namespace>** and **<vm_name>**:

```
$ oc logs -n <namespace> -l kubevirt.io/domain=<vm_name> --tail=-1 -c guest-console-log
```

16.3.4. Log aggregation

You can facilitate troubleshooting by aggregating and filtering logs.

16.3.4.1. Viewing aggregated OpenShift Virtualization logs with the LokiStack

You can view aggregated logs for OpenShift Virtualization pods and containers by using the LokiStack in the web console.

Prerequisites

- You deployed the LokiStack.

Procedure

1. Navigate to **Observe → Logs** in the web console.
2. Select **application**, for **virt-launcher** pod logs, or **infrastructure**, for OpenShift Virtualization control plane pods and containers, from the log type list.
3. Click **Show Query** to display the query field.
4. Enter the LogQL query in the query field and click **Run Query** to display the filtered logs.

16.3.4.2. OpenShift Virtualization LogQL queries

You can view and filter aggregated logs for OpenShift Virtualization components by running Loki Query Language (LogQL) queries on the **Observe → Logs** page in the web console.

The default log type is *infrastructure*. The **virt-launcher** log type is *application*.

Optional: You can include or exclude strings or regular expressions by using line filter expressions.

NOTE

If the query matches a large number of logs, the query might time out.

Table 16.3. OpenShift Virtualization LogQL example queries

Component	LogQL query
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"`

Component	LogQL query
Container	{log_type=~".+",kubernetes_container_name=~"<container>|<container>"} (1) |json|kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" (1) Specify one or more containers separated by a pipe (|).
virt-launcher	You must select **application** from the log type list before running this query. {log_type=~".+", kubernetes_container_name="compute"}|json |!= "custom-ga-command" (1) (1) **|!= "custom-ga-command"** excludes libvirt logs that contain the string**custom-ga-command**. (BZ#2177684)

You can filter log lines to include or exclude strings or regular expressions by using line filter expressions.

Table 16.4. Line filter expressions

Line filter expression	Description
|= "<string>"	Log line contains string
!= "<string>"	Log line does not contain string
|~ "<regex>"	Log line contains regular expression
!~ "<regex>"	Log line does not contain regular expression

Example line filter expression

```
{log_type=~".+"}|json
|kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster"
|= "error" != "timeout"
```

Additional resources for LokiStack and LogQL

- LogQL log queries in the Grafana documentation

16.3.5. Common error messages

The following error messages might appear in OpenShift Virtualization logs:

ErrImagePull or **ImagePullBackOff**

Indicates an incorrect deployment configuration or problems with the images that are referenced.

16.3.6. Troubleshooting data volumes

You can check the **Conditions** and **Events** sections of the **DataVolume** object to analyze and resolve issues.

16.3.6.1. About data volume conditions and events

You can diagnose data volume issues by examining the output of the **Conditions** and **Events** sections generated by the command:

```
$ oc describe dv <DataVolume>
```

The **Conditions** section displays the following **Types**:

- **Bound**
- **Running**
- **Ready**

The **Events** section provides the following additional information:

- **Type** of event
- **Reason** for logging
- **Source** of the event
- **Message** containing additional diagnostic information.

The output from **oc describe** does not always contains **Events**.

An event is generated when the **Status**, **Reason**, or **Message** changes. Both conditions and events react to changes in the state of the data volume.

For example, if you misspell the URL during an import operation, the import generates a 404 message. That message change generates an event with a reason. The output in the **Conditions** section is updated as well.

16.3.6.2. Analyzing data volume conditions and events

By inspecting the **Conditions** and **Events** sections generated by the **describe** command, you determine the state of the data volume in relation to persistent volume claims (PVCs), and whether or not an operation is actively running or completed. You might also receive messages that offer specific details about the status of the data volume, and how it came to be in its current state.

There are many different combinations of conditions. Each must be evaluated in its unique context.

Examples of various combinations follow.

- **Bound** - A successfully bound PVC displays in this example.

Note that the **Type** is **Bound**, so the **Status** is **True**. If the PVC is not bound, the **Status** is **False**.

When the PVC is bound, an event is generated stating that the PVC is bound. In this case, the **Reason** is **Bound** and **Status** is **True**. The **Message** indicates which PVC owns the data volume.

Message, in the **Events** section, provides further details including how long the PVC has been bound (**Age**) and by what resource (**From**), in this case **datavolume-controller**:

Example output

```
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** - In this case, note that **Type** is **Running** and **Status** is **False**, indicating that an event has occurred that caused an attempted operation to fail, changing the Status from **True** to **False**.
  However, note that **Reason** is **Completed** and the **Message** field indicates **Import Complete**.

  In the **Events** section, the **Reason** and **Message** contain additional troubleshooting information about the failed operation. In this example, the **Message** displays an inability to connect due to a **404**, listed in the **Events** section's first **Warning**.

  From this information, you conclude that an import operation was running, creating contention for other operations that are attempting to access the data volume:

  Example output

```
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** – If **Type** is **Ready** and **Status** is **True**, then the data volume is ready to be used, as in the following example. If the data volume is not ready to be used, the **Status** is **False**:

  Example output

  ```
  Status:
    Conditions:
      Last Heart Beat Time: 2020-07-15T04:31:39Z
      Last Transition Time:  2020-07-15T04:31:39Z
      Status:               True
      Type:                 Ready
  ```

CHAPTER 17. BACKUP AND RESTORE

17.1. BACKUP AND RESTORE BY USING VM SNAPSHOTS

You can back up and restore virtual machines (VMs) by using snapshots. Snapshots are supported by the following storage providers:

- Red Hat OpenShift Data Foundation
- Any other cloud storage provider with the Container Storage Interface (CSI) driver that supports the Kubernetes Volume Snapshot API

To create snapshots of a VM in the **Running** state with the highest integrity, install the QEMU guest agent if it is not included with your operating system. The QEMU guest agent is included with the default Red Hat templates.

IMPORTANT

Online snapshots are supported for virtual machines that have hot plugged virtual disks. However, hot plugged disks that are not in the virtual machine specification are not included in the snapshot.

The QEMU guest agent takes a consistent snapshot by attempting to quiesce the VM file system. This ensures that in-flight I/O is written to the disk before the snapshot is taken. If the guest agent is not present, quiescing is not possible and a best-effort snapshot is taken.

The conditions under which a snapshot is taken are reflected in the snapshot indications that are displayed in the web console or CLI. If these conditions do not meet your requirements, try creating the snapshot again or use an offline snapshot

17.1.1. About snapshots

A *snapshot* represents the state and data of a virtual machine (VM) at a specific point in time. You can use a snapshot to restore an existing VM to a previous state (represented by the snapshot) for backup and disaster recovery or to rapidly roll back to a previous development version.

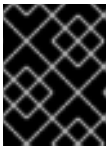
A VM snapshot is created from a VM that is powered off (Stopped state) or powered on (Running state).

When taking a snapshot of a running VM, the controller checks that the QEMU guest agent is installed and running. If so, it freezes the VM file system before taking the snapshot, and thaws the file system after the snapshot is taken.

The snapshot stores a copy of each Container Storage Interface (CSI) volume attached to the VM and a copy of the VM specification and metadata. Snapshots cannot be changed after creation.

You can perform the following snapshot actions:

- Create a new snapshot
- Create a clone of a virtual machine from a snapshot



IMPORTANT

Cloning a VM with a vTPM device attached to it or creating a new VM from its snapshot is not supported.

- List all snapshots attached to a specific VM
- Restore a VM from a snapshot
- Delete an existing VM snapshot

VM snapshot controller and custom resources

The VM snapshot feature introduces three new API objects defined as custom resource definitions (CRDs) for managing snapshots:

- **VirtualMachineSnapshot**: Represents a user request to create a snapshot. It contains information about the current state of the VM.
- **VirtualMachineSnapshotContent**: Represents a provisioned resource on the cluster (a snapshot). It is created by the VM snapshot controller and contains references to all resources required to restore the VM.
- **VirtualMachineRestore**: Represents a user request to restore a VM from a snapshot.

The VM snapshot controller binds a **VirtualMachineSnapshotContent** object with the **VirtualMachineSnapshot** object for which it was created, with a one-to-one mapping.

17.1.2. About application-consistent snapshots and backups

You can configure application-consistent snapshots and backups for Linux or Windows virtual machines (VMs) through a cycle of freezing and thawing. For any application, you can either configure a script on a Linux VM or register on a Windows VM to be notified when a snapshot or backup is due to begin.

On a Linux VM, freeze and thaw processes trigger automatically when a snapshot is taken or a backup is started by using, for example, a plugin from Velero or another backup vendor. The freeze process, performed by QEMU Guest Agent (QEMU GA) freeze hooks, ensures that before the snapshot or backup of a VM occurs, all of the VM's filesystems are frozen and each appropriately configured application is informed that a snapshot or backup is about to start. This notification affords each application the opportunity to quiesce its state. Depending on the application, quiescing might involve temporarily refusing new requests, finishing in-progress operations, and flushing data to disk. The operating system is then directed to quiesce the filesystems by flushing outstanding writes to disk and freezing new write activity. All new connection requests are refused. When all applications have become inactive, the QEMU GA freezes the filesystems, and a snapshot is taken or a backup initiated. After the taking of the snapshot or start of the backup, the thawing process begins. Filesystems writing is reactivated and applications receive notification to resume normal operations.

The same cycle of freezing and thawing is available on a Windows VM. Applications register with the Volume Shadow Copy Service (VSS) to receive notifications that they should flush out their data because a backup or snapshot is imminent. Thawing of the applications after the backup or snapshot is complete returns them to an active state. For more details, see the Windows Server documentation about the Volume Shadow Copy Service.

17.1.3. Creating snapshots

You can create snapshots of virtual machines (VMs) by using the OpenShift Container Platform web console or the command line.

17.1.3.1. Creating a snapshot by using the web console

You can create a snapshot of a virtual machine (VM) by using the OpenShift Container Platform web console.

Prerequisites

- The **snapshot** feature gate is enabled in the YAML configuration of the **kubevirt** CR.
- The VM snapshot includes disks that meet the following requirements:
  - The disks are data volumes or persistent volume claims.
  - The disks belong to a storage class that supports Container Storage Interface (CSI) volume snapshots.
  - The disks are *bound* to a persistent volume (PV) and *populated* with a datasource.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Select a VM to open the **VirtualMachine details** page.
3. Click the **Snapshots** tab and then click **Take Snapshot**.
   Alternatively, right-click the VM and select **Create snapshot** from the pop-up menu.
4. Enter the snapshot name.
5. Expand **Disks included in this Snapshot** to see the storage volumes to be included in the snapshot.
6. If your VM has disks that cannot be included in the snapshot and you wish to proceed, select **I am aware of this warning and wish to proceed**.
7. Click **Save**.

17.1.3.2. Creating a snapshot by using the CLI

You can create a virtual machine (VM) snapshot for an offline or online VM by creating a **VirtualMachineSnapshot** object.

Prerequisites

- Ensure the **Snapshot** feature gate is enabled for the **kubevirt** CR by using the following command:

```
$ oc get kubevirt kubevirt-hyperconverged -n openshift-cnv -o yaml
```

Truncated output

```
spec:
```

```
developerConfiguration:
  featureGates:
    - Snapshot
```

- Ensure that the VM snapshot includes disks that meet the following requirements:
  - The disks are data volumes or persistent volume claims.
  - The disks belong to a storage class that supports Container Storage Interface (CSI) volume snapshots.
  - The disks are *bound* to a persistent volume (PV) and *populated* with a datasource.
- Install the OpenShift CLI (**oc**).
- Optional: Power down the VM for which you want to create a snapshot.

Procedure

1. Create a YAML file to define a **VirtualMachineSnapshot** object that specifies the name of the new **VirtualMachineSnapshot** and the name of the source VM as in the following example:

```
apiVersion: snapshot.kubevirt.io/v1beta1
kind: VirtualMachineSnapshot
metadata:
  name: <snapshot_name>
spec:
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: <vm_name>
```

2. Create the **VirtualMachineSnapshot** object:

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

The snapshot controller creates a **VirtualMachineSnapshotContent** object, binds it to the **VirtualMachineSnapshot**, and updates the **status** and **readyToUse** fields of the **VirtualMachineSnapshot** object.

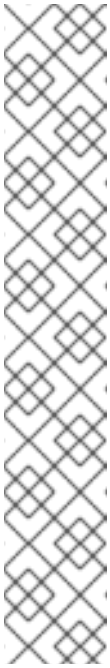
Verification

1. Optional: During the snapshot creation process, you can use the **wait** command to monitor the status of the snapshot and wait until it is ready for use:
   a. Enter the following command:

```
$ oc wait <vm_name> <snapshot_name> --for condition=Ready
```

   b. Verify the status of the snapshot:
      - **InProgress** - The snapshot operation is still in progress.
      - **Succeeded** - The snapshot operation completed successfully.

- **Failed** - The snapshot operaton failed.



NOTE

Online snapshots have a default time deadline of five minutes (**5m**). If the snapshot does not complete successfully in five minutes, the status is set to **failed**. Afterwards, the file system will be thawed and the VM unfrozen but the status remains **failed** until you delete the failed snapshot image.

To change the default time deadline, add the **FailureDeadline** attribute to the VM snapshot spec with the time designated in minutes (**m**) or in seconds (**s**) that you want to specify before the snapshot operation times out.

To set no deadline, you can specify **0**, though this is generally not recommended, as it can result in an unresponsive VM.

If you do not specify a unit of time such as **m** or **s**, the default is seconds (**s**).

2. Verify that the **VirtualMachineSnapshot** object is created and bound with **VirtualMachineSnapshotContent** and that the **readyToUse** flag is set to **true**:

```
$ oc describe vmsnapshot <snapshot_name>
```

Example output

```
apiVersion: snapshot.kubevirt.io/v1beta1
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/v1beta1/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
  indications: 5
    - Online
  includedVolumes: 6
    - name: rootdisk
      kind: PersistentVolumeClaim
      namespace: default
    - name: datadisk1
      kind: DataVolume
      namespace: default
```

1. The **status** field of the **Progressing** condition specifies if the snapshot is still being created.
2. The **status** field of the **Ready** condition specifies if the snapshot creation process is complete.
3. Specifies if the snapshot is ready to be used.
4. Specifies that the snapshot is bound to a **VirtualMachineSnapshotContent** object created by the snapshot controller.
5. Specifies additional information about the snapshot, such as whether it is an online snapshot, or whether it was created with QEMU guest agent running.
6. Lists the storage volumes that are part of the snapshot, as well as their parameters.

3. Check the **includedVolumes** section in the snapshot description to verify that the expected PVCs are included in the snapshot.

17.1.4. Verifying online snapshots by using snapshot indications

Snapshot indications are contextual information about online virtual machine (VM) snapshot operations. Indications are not available for offline virtual machine (VM) snapshot operations. Indications are helpful in describing details about the online snapshot creation.

Prerequisites

- You must have attempted to create an online VM snapshot.

Procedure

1. Display the output from the snapshot indications by performing one of the following actions:

- Use the command line to view indicator output in the **status** stanza of the **VirtualMachineSnapshot** object YAML.
- In the web console, click **VirtualMachineSnapshot → Status** in the **Snapshot details** screen.

2. Verify the status of your online VM snapshot by viewing the values of the **status.indications** parameter:
   - **Online** indicates that the VM was running during online snapshot creation.
   - **GuestAgent** indicates that the QEMU guest agent was active and successfully quiesced the guest file system for the online snapshot. This results in an application-consistent snapshot, preserving data integrity as if the applications had been gracefully shut down.
   - **NoGuestAgent** indicates that the QEMU guest agent was not installed, or not ready to quiesce the file system during the online snapshot. This results in a crash-consistent snapshot, which captures the VM's state like an abrupt power-off. As a result, application consistency is not guaranteed, which causes a risk of data issues for critical applications. For higher reliability, install and run the guest agent, or retry the snapshot.
   - **QuiesceFailed** indicates that an attempt to quiesce the file system failed during the online snapshot process. This means that the snapshot was created, but it is not necessarily application-consistent. To achieve proper consistency, retry the snapshot.

17.1.5. Restoring virtual machines from snapshots

You can restore virtual machines (VMs) from snapshots by using the OpenShift Container Platform web console or the command line.

17.1.5.1. Restoring a VM from a snapshot by using the web console

You can restore a virtual machine (VM) to a previous configuration represented by a snapshot in the OpenShift Container Platform web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Select a VM to open the **VirtualMachine details** page.
3. If the VM is running, click the Options menu and select **Stop** to power it down.
4. Click the **Snapshots** tab to view a list of snapshots associated with the VM.
5. Select a snapshot to open the **Snapshot Details** screen.
6. Click the Options menu and select **Restore VirtualMachine from snapshot**.
7. Click **Restore**.
8. Optional: You can also create a new VM based on the snapshot. To do so:

a. In the Options menu  of the snapshot, select **Create VirtualMachine from Snapshot**

b. Provide a name for the new VM.

c. Click **Create**

17.1.5.2. Restoring a VM from a snapshot by using the CLI

You can restore an existing virtual machine (VM) to a previous configuration by using the command line. You can only restore from an offline VM snapshot.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Power down the VM you want to restore.
- Optional: Adjust what happens if the target VM is not fully stopped (*ready*). To do so, set the **targetReadinessPolicy** parameter in the **vmrestore** YAML configuration to one of the following values:
  - **FailImmediate** - The restore process fails immediately if the VM is not ready.
  - **StopTarget** - If the VM is not ready, it gets stopped, and the restore process starts.
  - **WaitGracePeriod 5** - The restore process waits for a set amount of time, in minutes, for the VM to be ready. This is the default setting, with the default value set to 5 minutes.
  - **WaitEventually** - The restore process waits indefinitely for the VM to be ready.

Procedure

1. Create a YAML file to define a **VirtualMachineRestore** object that specifies the name of the VM you want to restore and the name of the snapshot to be used as the source as in the following example:

```
apiVersion: snapshot.kubevirt.io/v1beta1
kind: VirtualMachineRestore
metadata:
  name: <vm_restore>
spec:
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: <vm_name>
  virtualMachineSnapshotName: <snapshot_name>
```

2. Create the **VirtualMachineRestore** object:

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

The snapshot controller updates the status fields of the **VirtualMachineRestore** object and replaces the existing VM configuration with the snapshot content.

Verification

- Verify that the VM is restored to the previous state represented by the snapshot and that the **complete** flag is set to **true**:

```
$ oc get vmrestore <vm_restore>
```

Example output

```
apiVersion: snapshot.kubevirt.io/v1beta1
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/v1beta1/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. Specifies if the process of restoring the VM to the state represented by the snapshot is complete.
2. The **status** field of the **Progressing** condition specifies if the VM is still being restored.
3. The **status** field of the **Ready** condition specifies if the VM restoration process is complete.

17.1.6. Deleting snapshots

You can delete snapshots of virtual machines (VMs) by using the OpenShift Container Platform web console or the command line.

17.1.6.1. Deleting a snapshot by using the web console

You can delete an existing virtual machine (VM) snapshot by using the web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Select a VM to open the **VirtualMachine details** page.
3. Click the **Snapshots** tab to view a list of snapshots associated with the VM.
4. Click the Options menu  beside a snapshot and select **Delete snapshot**.
5. Click **Delete**.

17.1.6.2. Deleting a virtual machine snapshot in the CLI

You can delete an existing virtual machine (VM) snapshot by deleting the appropriate **VirtualMachineSnapshot** object.

Prerequisites

- Install the OpenShift CLI (**oc**).

Procedure

- Delete the **VirtualMachineSnapshot** object:

```
$ oc delete vmsnapshot <snapshot_name>
```

The snapshot controller deletes the **VirtualMachineSnapshot** along with the associated **VirtualMachineSnapshotContent** object.

Verification

- Verify that the snapshot is deleted and no longer attached to this VM:

```
$ oc get vmsnapshot
```

17.1.7. Additional resources

- CSI Volume Snapshots

17.2. BACKING UP AND RESTORING VIRTUAL MACHINES

IMPORTANT

Red Hat supports using OpenShift Virtualization 4.14 or later with OADP 1.3.x or later.

OADP versions earlier than 1.3.0 are not supported for back up and restore of OpenShift Virtualization.

Back up and restore virtual machines by using the OpenShift API for Data Protection.

You can install the OpenShift API for Data Protection (OADP) with OpenShift Virtualization by installing the OADP Operator and configuring a backup location. You can then install the Data Protection Application.

NOTE

OpenShift API for Data Protection with OpenShift Virtualization supports the following backup and restore storage options:

- Container Storage Interface (CSI) backups
- Container Storage Interface (CSI) backups with DataMover

The following storage options are excluded:

- File system backup and restore
- Volume snapshot backup and restore

For more information, see Backing up applications with File System Backup: Kopia or Restic.

To install the OADP Operator in a restricted network environment, you must first disable the default software catalog sources and mirror the Operator catalog.

See Using Operator Lifecycle Manager in disconnected environments for details.

17.2.1. Installing and configuring OADP with OpenShift Virtualization

As a cluster administrator, you install OADP by installing the OADP Operator.

The latest version of the OADP Operator installs Velero 1.16.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Install the OADP Operator according to the instructions for your storage provider.
2. Install the Data Protection Application (DPA) with the **kubevirt** and **openshift** OADP plugins.
3. Back up virtual machines by creating a **Backup** custom resource (CR).

WARNING

Red Hat support is limited to only the following options:

- CSI backups
- CSI backups with DataMover.

You restore the **Backup** CR by creating a **Restore** CR.

Additional resources

- OADP plugins
- **Backup** custom resource (CR)
- **Restore** CR
- Using Operator Lifecycle Manager in disconnected environments

17.2.2. Installing the Data Protection Application

You install the Data Protection Application (DPA) by creating an instance of the **DataProtectionApplication** API.

Prerequisites

- You must install the OADP Operator.
- You must configure object storage as a backup location.
- If you use snapshots to back up PVs, your cloud provider must support either a native snapshot API or Container Storage Interface (CSI) snapshots.
- If the backup and snapshot locations use the same credentials, you must create a **Secret** with the default name, **cloud-credentials**.

NOTE

If you do not want to specify backup or snapshot locations during the installation, you can create a default **Secret** with an empty **credentials-velero** file. If there is no default **Secret**, the installation will fail.

Procedure

1. Click **Ecosystem → Installed Operators** and select the OADP Operator.
2. Under **Provided APIs**, click **Create instance** in the **DataProtectionApplication** box.
3. Click **YAML View** and update the parameters of the **DataProtectionApplication** manifest:

```
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) The default namespace for OADP is **openshift-adp**. The namespace is a variable and is configurable.

(2) The **kubevirt** plugin is mandatory for OpenShift Virtualization.

(3) Specify the plugin for the backup provider, for example, **gcp**, if it exists.

(4) The **csi** plugin is mandatory for backing up PVs with CSI snapshots. The **csi** plugin uses the Velero CSI beta snapshot APIs. You do not need to configure a snapshot location.

5. The **openshift** plugin is mandatory.
6. Specify how many minutes to wait for several Velero resources before timeout occurs, such as Velero CRD availability, volumeSnapshot deletion, and backup repository availability. The default is 10m.
7. The administrative agent that routes the administrative requests to servers.
8. Set this value to **true** if you want to enable **nodeAgent** and perform File System Backup.
9. Enter **kopia** as your uploader to use the Built-in DataMover. The **nodeAgent** deploys a daemon set, which means that the **nodeAgent** pods run on each working node. You can configure File System Backup by adding **spec.defaultVolumesToFsBackup: true** to the **Backup** CR.
10. Specify the nodes on which Kopia are available. By default, Kopia runs on all nodes.
11. Specify the backup provider.
12. Specify the correct default name for the **Secret**, for example, **cloud-credentials-gcp**, if you use a default plugin for the backup provider. If specifying a custom name, then the custom name is used for the backup location. If you do not specify a **Secret** name, the default name is used.
13. Specify a bucket as the backup storage location. If the bucket is not a dedicated bucket for Velero backups, you must specify a prefix.
14. Specify a prefix for Velero backups, for example, **velero**, if the bucket is used for multiple purposes.

4. Click **Create**.

Verification

1. Verify the installation by viewing the OpenShift API for Data Protection (OADP) resources by running the following command:

```
$ oc get all -n openshift-adp
```

Example output

```
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. Verify that the **DataProtectionApplication** (DPA) is reconciled by running the following command:

```
$ oc get dpa dpa-sample -n openshift-adp -o jsonpath='{.status}'
```

Example output

```
{"conditions":[{"lastTransitionTime":"2023-10-27T01:23:57Z","message":"Reconcile
complete","reason":"Complete","status":"True","type":"Reconciled"}]}
```

3. Verify the **type** is set to **Reconciled**.

4. Verify the backup storage location and confirm that the **PHASE** is **Available** by running the following command:

```
$ oc get backupstoragelocations.velero.io -n openshift-adp
```

Example output

```
NAME           PHASE       LAST VALIDATED   AGE     DEFAULT
dpa-sample-1   Available   1s               3d16h   true
```

17.3. DISASTER RECOVERY

OpenShift Virtualization supports using disaster recovery (DR) solutions to ensure that your environment can recover after a site outage. To use these methods, you must plan your OpenShift Virtualization deployment in advance.

17.3.1. About disaster recovery methods

For an overview of disaster recovery (DR) concepts, architecture, and planning considerations, see the Red Hat OpenShift Virtualization disaster recovery guide in the Red Hat Knowledgebase.

The two primary DR methods for OpenShift Virtualization are Metropolitan Disaster Recovery (Metro-DR) and Regional-DR.

17.3.1.1. Metro-DR

Metro-DR uses synchronous replication. It writes to storage at both the primary and secondary sites so

that the data is always synchronized between sites. Because the storage provider is responsible for ensuring that the synchronization succeeds, the environment must meet the throughput and latency requirements of the storage provider.

17.3.1.2. Regional-DR

Regional-DR uses asynchronous replication. The data in the primary site is synchronized with the secondary site at regular intervals. For this type of replication, you can have a higher latency connection between the primary and secondary sites.

17.3.2. Defining applications for disaster recovery

Define applications for disaster recovery by using VMs that Red Hat Advanced Cluster Management (RHACM) manages or discovers.

17.3.2.1. Best practices when defining an RHACM-managed VM

When creating an RHACM-managed application that includes a VM, you must use a GitOps workflow and create an RHACM application or **ApplicationSet** resource.

You can take several actions to improve your experience and chance of success when defining an RHACM-managed VM.

Use a PVC and populator to define storage for the VM

Because data volumes create persistent volume claims (PVCs) implicitly, data volumes and VMs with data volume templates do not fit as neatly into the GitOps model.

Use the import method when choosing a population source for your VM disk

Select a RHEL image from the software catalog to use the import method. Red Hat recommends using a specific version of the image rather than a floating tag for consistent results. The KubeVirt community maintains container disks for other operating systems in a Quay repository.

Use pullMethod: node

Use the pod **pullMethod: node** when creating a data volume from a registry source to take advantage of the OpenShift Container Platform pull secret, which is required to pull container images from the Red Hat registry.

17.3.2.2. Best practices when defining an RHACM-discovered VM

You can configure any VM in the cluster that is not an RHACM-managed application as an RHACM-discovered application. This includes VMs imported by using the Migration Toolkit for Virtualization (MTV), VMs created by using the OpenShift Container Platform web console, or VMs created by any other means, such as the CLI.

You can take several actions to improve your experience and chance of success when defining an RHACM-discovered VM.

Protecting the VM when using MTV, the OpenShift Container Platform web console, or a custom VM

Because automatic labeling is not currently available, the application owner must manually label the components of the VM application when using MTV, the OpenShift Container Platform web console, or a custom VM.
After creating the VM, apply a common label to the following resources associated with the VM: **VirtualMachine**, **DataVolume**, **PersistentVolumeClaim**, **Service**, **Route**, **Secret** and **ConfigMap**. If the VM uses an instance type or preference, you must also label the **ControllerRevision** copy of

these objects referenced by the spec or status of the VM. Do not label virtual machine instances (VMIs) or pods; OpenShift Virtualization creates and manages these automatically.

IMPORTANT

You must apply the common label to everything in the namespace that you want to protect, including objects that you added to the VM that are not listed here.

Including more than the `VirtualMachine` object in the VM

Working VMs typically also contain data volumes, persistent volume claims (PVCs), services, routes, secrets, **`ConfigMap`** objects, and **`VirtualMachineSnapshot`** objects.

Including the VM as part of a larger logical application

This includes other pod-based workloads and VMs.

17.3.3. VM behavior during disaster recovery scenarios

VMs typically act similarly to pod-based workloads during both relocate and failover disaster recovery flows.

Relocate
Use relocate to move an application from the primary environment to the secondary environment when the primary environment is still accessible. During relocate, the VM is gracefully terminated, any unreplicated data is synchronized to the secondary environment, and the VM starts in the secondary environment.

Because the VM terminates gracefully, there is no data loss. Therefore, the VM operating system will not perform crash recovery.

Failover
Use failover when there is a critical failure in the primary environment that makes it impractical or impossible to use relocation to move the workload to a secondary environment. When failover is executed, the storage is fenced from the primary environment, the I/O to the VM disks is abruptly halted, and the VM restarts in the secondary environment using the replicated data.

You should expect data loss due to failover. The extent of loss depends on whether you use Metro-DR, which uses synchronous replication, or Regional-DR, which uses asynchronous replication. Because Regional-DR uses snapshot-based replication intervals, the window of data loss is proportional to the replication interval length. When the VM restarts, the operating system might perform crash recovery.

17.3.4. Disaster recovery solutions for Red Hat managed clusters

The following DR solutions combine Red Hat Advanced Cluster Management (RHACM), Red Hat Ceph Storage, and OpenShift Data Foundation components. You can use them to failover applications from the primary to the secondary site, and to relocate the applications back to the primary site after you restore the disaster site.

17.3.4.1. Metro-DR for Red Hat OpenShift Data Foundation

OpenShift Virtualization supports the Metro-DR solution for OpenShift Data Foundation, which provides two-way synchronous data replication between managed OpenShift Virtualization clusters installed on primary and secondary sites.

Metro-DR differences

- This synchronous solution is only available to metropolitan distance data centers with a network round-trip latency of 10 milliseconds or less.
- Multiple disk VMs are supported.
- To prevent data corruption, you must ensure that storage is fenced during failover.

  TIP

  Fencing means isolating a node so that workloads do not run on it.

For more information about using the Metro-DR solution for OpenShift Data Foundation with OpenShift Virtualization, see IBM's OpenShift Data Foundation Metro-DR documentation.

17.3.4.2. Regional-DR for Red Hat OpenShift Data Foundation

OpenShift Virtualization supports the Regional-DR solution for OpenShift Data Foundation , which provides asynchronous data replication at regular intervals between managed OpenShift Virtualization clusters installed on primary and secondary sites.

Regional-DR differences

- Regional-DR supports higher network latency between the primary and secondary sites.
- Regional-DR uses RBD snapshots to replicate data asynchronously. Currently, your applications must be resilient to small variances between VM disks. You can prevent these variances by using single disk VMs.
- Using the import method when selecting a population source for your VM disk is recommended. However, you can protect VMs that use cloned PVCs if you select a **VolumeReplicationClass** that enables image flattening. For more information, see the OpenShift Data Foundation documentation.

For more information about using the Regional-DR solution for OpenShift Data Foundation with OpenShift Virtualization, see IBM's OpenShift Data Foundation Regional-DR documentation.

17.3.5. Additional resources

- Configuring OpenShift Data Foundation Disaster Recovery for OpenShift Workloads
- Use OpenShift Data Foundation Disaster Recovery to Protect Virtual Machines in the Red Hat Knowledgebase
- Red Hat Advanced Cluster Management for Kubernetes 2.10