



OpenShift Container Platform 4.8

OpenShift Virtualization

OpenShift Virtualization installation, usage, and release notes

OpenShift Container Platform 4.8 OpenShift Virtualization

OpenShift Virtualization installation, usage, and release notes

Legal Notice

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

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

Table of Contents

CHAPTER 1. ABOUT OPENSIFT VIRTUALIZATION	14
1.1. WHAT YOU CAN DO WITH OPENSIFT VIRTUALIZATION	14
1.1.1. OpenShift Virtualization supported cluster version	14
CHAPTER 2. START HERE WITH OPENSIFT VIRTUALIZATION	15
2.1. CLUSTER ADMINISTRATOR	15
2.2. VIRTUALIZATION ADMINISTRATOR	15
2.3. VIRTUAL MACHINE ADMINISTRATOR / DEVELOPER	15
CHAPTER 3. OPENSIFT VIRTUALIZATION RELEASE NOTES	17
3.1. ABOUT RED HAT OPENSIFT VIRTUALIZATION	17
3.1.1. OpenShift Virtualization supported cluster version	17
3.1.2. Supported guest operating systems	17
3.2. MAKING OPEN SOURCE MORE INCLUSIVE	17
3.3. NEW AND CHANGED FEATURES	17
3.3.1. Quick starts	18
3.3.2. Networking	18
3.3.3. Storage	18
3.4. DEPRECATED AND REMOVED FEATURES	19
3.4.1. Deprecated features	19
3.5. NOTABLE TECHNICAL CHANGES	19
3.6. TECHNOLOGY PREVIEW FEATURES	19
3.7. KNOWN ISSUES	20
CHAPTER 4. INSTALLING OPENSIFT VIRTUALIZATION	23
4.1. PREPARING YOUR CLUSTER FOR OPENSIFT VIRTUALIZATION	23
4.1.1. Hardware and operating system requirements	23
4.1.2. Physical resource overhead requirements	24
4.1.2.1. Memory overhead	24
4.1.2.2. CPU overhead	25
4.1.2.3. Storage overhead	25
4.1.2.4. Example	25
4.1.3. Object maximums	25
4.1.4. Restricted network environments	26
4.1.5. Live migration	26
4.1.6. Snapshots and cloning	26
4.1.7. Cluster high-availability options	26
4.2. SPECIFYING NODES FOR OPENSIFT VIRTUALIZATION COMPONENTS	27
4.2.1. About node placement for virtualization components	27
4.2.1.1. How to apply node placement rules to virtualization components	27
4.2.1.2. Node placement in the OLM Subscription object	28
4.2.1.3. Node placement in the HyperConverged object	28
4.2.1.4. Node placement in the HostPathProvisioner object	29
4.2.1.5. Additional resources	29
4.2.2. Example manifests	29
4.2.2.1. Operator Lifecycle Manager Subscription object	29
4.2.2.1.1. Example: Node placement with nodeSelector in the OLM Subscription object	29
4.2.2.1.2. Example: Node placement with tolerations in the OLM Subscription object	30
4.2.2.2. HyperConverged object	30
4.2.2.2.1. Example: Node placement with nodeSelector in the HyperConverged Cluster CR	30
4.2.2.2.2. Example: Node placement with affinity in the HyperConverged Cluster CR	31
4.2.2.2.3. Example: Node placement with tolerations in the HyperConverged Cluster CR	31

4.2.2.3. HostPathProvisioner object	32
4.2.2.3.1. Example: Node placement with nodeSelector in the HostPathProvisioner object	32
4.3. INSTALLING OPENSIFT VIRTUALIZATION USING THE WEB CONSOLE	32
4.3.1. Installing the OpenShift Virtualization Operator	32
4.3.2. Next steps	34
4.3.3. Prerequisites	34
4.3.4. Subscribing to the OpenShift Virtualization catalog by using the CLI	34
4.3.5. Deploying the OpenShift Virtualization Operator by using the CLI	35
4.3.6. Next steps	36
4.4. INSTALLING THE VIRTCTL CLIENT	36
4.4.1. Installing the virtctl client from the web console	36
4.4.2. Enabling OpenShift Virtualization repositories	37
4.4.3. Installing the virtctl client	37
4.4.4. Additional resources	37
4.5. UNINSTALLING OPENSIFT VIRTUALIZATION USING THE WEB CONSOLE	37
4.5.1. Prerequisites	37
4.5.2. Deleting the OpenShift Virtualization Operator Deployment custom resource	38
4.5.3. Deleting the OpenShift Virtualization catalog subscription	38
4.5.4. Deleting a namespace using the web console	39
4.6. UNINSTALLING OPENSIFT VIRTUALIZATION USING THE CLI	39
4.6.1. Prerequisites	39
4.6.2. Deleting OpenShift Virtualization	39
CHAPTER 5. UPDATING OPENSIFT VIRTUALIZATION	41
5.1. ABOUT UPGRADING OPENSIFT VIRTUALIZATION	41
5.1.1. How OpenShift Virtualization upgrades work	41
5.1.2. How OpenShift Virtualization upgrades affect your cluster	41
5.2. MANUALLY APPROVING A PENDING OPERATOR UPDATE	42
5.3. MONITORING UPGRADE STATUS	42
5.4. ADDITIONAL RESOURCES	43
CHAPTER 6. ADDITIONAL SECURITY PRIVILEGES GRANTED FOR KUBEVIRT-CONTROLLER AND VIRT-LAUNCHER	44
6.1. EXTENDED SELINUX POLICIES FOR VIRT-LAUNCHER PODS	44
6.2. ADDITIONAL OPENSIFT CONTAINER PLATFORM SECURITY CONTEXT CONSTRAINTS AND LINUX CAPABILITIES FOR THE KUBEVIRT-CONTROLLER SERVICE ACCOUNT	44
6.2.1. Additional SCCs granted to the kubevirt-controller service account	44
6.2.2. Viewing the SCC and RBAC definitions for the kubevirt-controller	45
6.3. ADDITIONAL RESOURCES	45
CHAPTER 7. USING THE CLI TOOLS	46
7.1. PREREQUISITES	46
7.2. VIRTCTL CLIENT COMMANDS	46
7.3. OPENSIFT CONTAINER PLATFORM CLIENT COMMANDS	47
CHAPTER 8. VIRTUAL MACHINES	49
8.1. CREATING VIRTUAL MACHINES	49
8.1.1. Using a Quick Start to create a virtual machine	49
8.1.2. Running the virtual machine wizard to create a virtual machine	50
8.1.2.1. Virtual machine wizard fields	51
8.1.2.2. Networking fields	53
8.1.2.3. Storage fields	54
Advanced storage settings	55
8.1.2.4. Cloud-init fields	56

8.1.2.5. Pasting in a pre-configured YAML file to create a virtual machine	56
8.1.3. Using the CLI to create a virtual machine	57
8.1.4. Virtual machine storage volume types	58
8.1.5. About RunStrategies for virtual machines	60
8.1.6. Additional resources	61
8.2. EDITING VIRTUAL MACHINES	62
8.2.1. Editing a virtual machine in the web console	62
8.2.2. Editing a virtual machine YAML configuration using the web console	63
8.2.3. Editing a virtual machine YAML configuration using the CLI	63
8.2.4. Adding a virtual disk to a virtual machine	64
8.2.4.1. Storage fields	64
Advanced storage settings	66
8.2.5. Adding a network interface to a virtual machine	66
8.2.5.1. Networking fields	67
8.2.6. Editing CD-ROMs for Virtual Machines	67
8.3. EDITING BOOT ORDER	68
8.3.1. Adding items to a boot order list in the web console	68
8.3.2. Editing a boot order list in the web console	68
8.3.3. Editing a boot order list in the YAML configuration file	69
8.3.4. Removing items from a boot order list in the web console	70
8.4. DELETING VIRTUAL MACHINES	70
8.4.1. Deleting a virtual machine using the web console	70
8.4.2. Deleting a virtual machine by using the CLI	71
8.5. MANAGING VIRTUAL MACHINE INSTANCES	71
8.5.1. About virtual machine instances	71
8.5.2. Listing all virtual machine instances using the CLI	72
8.5.3. Listing standalone virtual machine instances using the web console	72
8.5.4. Editing a standalone virtual machine instance using the web console	72
8.5.5. Deleting a standalone virtual machine instance using the CLI	73
8.5.6. Deleting a standalone virtual machine instance using the web console	73
8.6. CONTROLLING VIRTUAL MACHINE STATES	74
8.6.1. Starting a virtual machine	74
8.6.2. Restarting a virtual machine	74
8.6.3. Stopping a virtual machine	75
8.6.4. Unpausing a virtual machine	75
8.7. ACCESSING VIRTUAL MACHINE CONSOLES	76
8.7.1. Accessing virtual machine consoles in the OpenShift Container Platform web console	76
8.7.1.1. Connecting to the serial console	76
8.7.1.2. Connecting to the VNC console	77
8.7.1.3. Connecting to a Windows virtual machine with RDP	77
8.7.1.4. Copying the SSH command from the web console	78
8.7.2. Accessing virtual machine consoles by using CLI commands	78
8.7.2.1. Accessing a virtual machine instance via SSH	78
8.7.2.2. Accessing a virtual machine via SSH with YAML configurations	79
8.7.2.3. Accessing the serial console of a virtual machine instance	82
8.7.2.4. Accessing the graphical console of a virtual machine instances with VNC	82
8.7.2.5. Connecting to a Windows virtual machine with an RDP console	83
8.8. TRIGGERING VIRTUAL MACHINE FAILOVER BY RESOLVING A FAILED NODE	84
8.8.1. Prerequisites	84
8.8.2. Deleting nodes from a bare metal cluster	84
8.8.3. Verifying virtual machine failover	85
8.8.3.1. Listing all virtual machine instances using the CLI	85
8.9. INSTALLING THE QEMU GUEST AGENT ON VIRTUAL MACHINES	85

8.9.1. Installing QEMU guest agent on a Linux virtual machine	86
8.9.2. Installing QEMU guest agent on a Windows virtual machine	86
8.9.2.1. Installing VirtIO drivers on an existing Windows virtual machine	86
8.9.2.2. Installing VirtIO drivers during Windows installation	87
8.10. VIEWING THE QEMU GUEST AGENT INFORMATION FOR VIRTUAL MACHINES	87
8.10.1. Prerequisites	87
8.10.2. About the QEMU guest agent information in the web console	87
8.10.3. Viewing the QEMU guest agent information in the web console	88
8.11. MANAGING CONFIG MAPS, SECRETS, AND SERVICE ACCOUNTS IN VIRTUAL MACHINES	88
8.11.1. Adding a secret, config map, or service account to a virtual machine	88
8.11.2. Removing a secret, config map, or service account from a virtual machine	89
8.11.3. Additional resources	90
8.12. INSTALLING VIRTIO DRIVER ON AN EXISTING WINDOWS VIRTUAL MACHINE	90
8.12.1. Understanding VirtIO drivers	90
8.12.2. Supported VirtIO drivers for Microsoft Windows virtual machines	91
8.12.3. Adding VirtIO drivers container disk to a virtual machine	91
8.12.4. Installing VirtIO drivers on an existing Windows virtual machine	92
8.12.5. Removing the VirtIO container disk from a virtual machine	93
8.13. INSTALLING VIRTIO DRIVER ON A NEW WINDOWS VIRTUAL MACHINE	93
8.13.1. Prerequisites	93
8.13.2. Understanding VirtIO drivers	93
8.13.3. Supported VirtIO drivers for Microsoft Windows virtual machines	94
8.13.4. Adding VirtIO drivers container disk to a virtual machine	94
8.13.5. Installing VirtIO drivers during Windows installation	95
8.13.6. Removing the VirtIO container disk from a virtual machine	95
8.14. ADVANCED VIRTUAL MACHINE MANAGEMENT	96
8.14.1. Working with resource quotas for virtual machines	96
8.14.1.1. Setting resource quota limits for virtual machines	96
8.14.1.2. Additional resources	96
8.14.2. Specifying nodes for virtual machines	97
8.14.2.1. About node placement for virtual machines	97
8.14.2.2. Node placement examples	97
8.14.2.2.1. Example: VM node placement with nodeSelector	98
8.14.2.2.2. Example: VM node placement with pod affinity and pod anti-affinity	98
8.14.2.2.3. Example: VM node placement with node affinity	99
8.14.2.2.4. Example: VM node placement with tolerations	100
8.14.2.3. Additional resources	100
8.14.3. Configuring certificate rotation	100
8.14.3.1. Configuring certificate rotation	100
8.14.3.2. Troubleshooting certificate rotation parameters	101
8.14.4. Automating management tasks	102
8.14.4.1. About Red Hat Ansible Automation	102
8.14.4.2. Automating virtual machine creation	102
8.14.4.3. Example: Ansible Playbook for creating virtual machines	104
8.14.5. Using EFI mode for virtual machines	104
8.14.5.1. About EFI mode for virtual machines	104
8.14.5.2. Booting virtual machines in EFI mode	105
8.14.6. Configuring PXE booting for virtual machines	105
8.14.6.1. Prerequisites	106
8.14.6.2. PXE booting with a specified MAC address	106
8.14.6.3. Template: Virtual machine instance configuration file for PXE booting	108
8.14.7. Managing guest memory	109
8.14.7.1. Configuring guest memory overcommitment	109

8.14.7.2. Disabling guest memory overhead accounting	110
8.14.8. Using huge pages with virtual machines	111
8.14.8.1. Prerequisites	111
8.14.8.2. What huge pages do	111
8.14.8.3. Configuring huge pages for virtual machines	112
8.14.9. Enabling dedicated resources for virtual machines	113
8.14.9.1. About dedicated resources	113
8.14.9.2. Prerequisites	113
8.14.9.3. Enabling dedicated resources for a virtual machine	113
8.14.10. Scheduling virtual machines	113
8.14.10.1. Understanding policy attributes	113
8.14.10.2. Setting a policy attribute and CPU feature	114
8.14.10.3. Scheduling virtual machines with the supported CPU model	114
8.14.10.4. Scheduling virtual machines with the host model	115
8.14.11. Configuring PCI passthrough	115
8.14.11.1. About preparing a host device for PCI passthrough	115
8.14.11.1.1. Adding kernel arguments to enable the IOMMU driver	116
8.14.11.1.2. Binding PCI devices to the VFIO driver	117
8.14.11.1.3. Exposing PCI host devices in the cluster using the CLI	118
8.14.11.1.4. Removing PCI host devices from the cluster using the CLI	120
8.14.11.2. Configuring virtual machines for PCI passthrough	122
8.14.11.2.1. Assigning a PCI device to a virtual machine	122
8.14.11.3. Additional resources	122
8.14.12. Configuring a watchdog	122
8.14.12.1. Prerequisites	123
8.14.12.2. Defining a watchdog device	123
8.14.12.3. Installing a watchdog device	124
8.14.12.4. Additional resources	124
8.15. IMPORTING VIRTUAL MACHINES	124
8.15.1. TLS certificates for data volume imports	124
8.15.1.1. Adding TLS certificates for authenticating data volume imports	124
8.15.1.2. Example: Config map created from a TLS certificate	125
8.15.2. Importing virtual machine images with data volumes	125
8.15.2.1. Prerequisites	125
8.15.2.2. CDI supported operations matrix	126
8.15.2.3. About data volumes	126
8.15.2.4. Importing a virtual machine image into storage by using a data volume	126
8.15.2.5. Additional resources	129
8.15.3. Importing virtual machine images into block storage with data volumes	129
8.15.3.1. Prerequisites	130
8.15.3.2. About data volumes	130
8.15.3.3. About block persistent volumes	130
8.15.3.4. Creating a local block persistent volume	130
8.15.3.5. Importing a virtual machine image into block storage by using a data volume	131
8.15.3.6. CDI supported operations matrix	133
8.15.3.7. Additional resources	133
8.15.4. Importing a single Red Hat Virtualization virtual machine	133
8.15.4.1. OpenShift Virtualization storage feature matrix	134
8.15.4.2. Prerequisites for importing a virtual machine	134
8.15.4.3. Importing a virtual machine with the VM Import wizard	135
Virtual machine wizard fields	136
Networking fields	138
Storage fields	139

Advanced storage settings	140
8.15.4.4. Importing a virtual machine with the CLI	141
8.15.4.4.1. Creating a config map for importing a VM	144
8.15.4.5. Troubleshooting a virtual machine import	148
8.15.4.5.1. Logs	148
8.15.4.5.2. Error messages	148
8.15.4.5.3. Known issues	149
8.15.5. Importing a single VMware virtual machine or template	149
8.15.5.1. OpenShift Virtualization storage feature matrix	149
8.15.5.2. Preparing a VDDK image	149
8.15.5.2.1. Configuring an internal image registry	150
Changing the image registry's management state	150
Configuring registry storage for bare metal and other manual installations	150
Accessing registry directly from the cluster	152
Exposing a secure registry manually	153
8.15.5.2.2. Configuring an external image registry	154
Adding certificate authorities to the cluster	155
Allowing pods to reference images from other secured registries	155
8.15.5.2.3. Creating and using a VDDK image	156
8.15.5.3. Importing a virtual machine with the VM Import wizard	157
Virtual machine wizard fields	160
Cloud-init fields	162
Networking fields	162
Storage fields	162
Advanced storage settings	164
8.15.5.3.1. Updating the NIC name of an imported virtual machine	164
8.15.5.4. Troubleshooting a virtual machine import	165
8.15.5.4.1. Logs	165
8.15.5.4.2. Error messages	165
8.16. CLONING VIRTUAL MACHINES	166
8.16.1. Enabling user permissions to clone data volumes across namespaces	166
8.16.1.1. Prerequisites	166
8.16.1.2. About data volumes	166
8.16.1.3. Creating RBAC resources for cloning data volumes	166
8.16.2. Cloning a virtual machine disk into a new data volume	167
8.16.2.1. Prerequisites	168
8.16.2.2. About data volumes	168
8.16.2.3. Cloning the persistent volume claim of a virtual machine disk into a new data volume	168
8.16.2.4. Template: Data volume clone configuration file	169
8.16.2.5. CDI supported operations matrix	170
8.16.3. Cloning a virtual machine by using a data volume template	170
8.16.3.1. Prerequisites	171
8.16.3.2. About data volumes	171
8.16.3.3. Creating a new virtual machine from a cloned persistent volume claim by using a data volume template	171
8.16.3.4. Template: Data volume virtual machine configuration file	172
8.16.3.5. CDI supported operations matrix	173
8.16.4. Cloning a virtual machine disk into a new block storage data volume	174
8.16.4.1. Prerequisites	174
8.16.4.2. About data volumes	174
8.16.4.3. About block persistent volumes	174
8.16.4.4. Creating a local block persistent volume	175
8.16.4.5. Cloning the persistent volume claim of a virtual machine disk into a new data volume	176

8.16.4.6. CDI supported operations matrix	177
8.17. VIRTUAL MACHINE NETWORKING	178
8.17.1. Configuring the virtual machine for the default pod network	178
8.17.1.1. Configuring masquerade mode from the command line	178
8.17.1.2. Configuring masquerade mode with dual-stack (IPv4 and IPv6)	179
8.17.2. Creating a service to expose a virtual machine	180
8.17.2.1. About services	180
8.17.2.1.1. Dual-stack support	180
8.17.2.2. Exposing a virtual machine as a service	181
8.17.2.3. Additional resources	183
8.17.3. Attaching a virtual machine to a Linux bridge network	183
8.17.3.1. Connecting to the network through the network attachment definition	184
8.17.3.1.1. Creating a Linux bridge node network configuration policy	184
8.17.3.2. Creating a Linux bridge network attachment definition	184
8.17.3.2.1. Prerequisites	185
8.17.3.2.2. Creating a Linux bridge network attachment definition in the web console	185
8.17.3.2.3. Creating a Linux bridge network attachment definition in the CLI	185
8.17.3.3. Configuring the virtual machine for a Linux bridge network	187
8.17.3.3.1. Creating a NIC for a virtual machine in the web console	187
8.17.3.3.2. Networking fields	187
8.17.3.3.3. Attaching a virtual machine to an additional network in the CLI	188
8.17.4. Configuring IP addresses for virtual machines	189
8.17.4.1. Configuring an IP address for a new virtual machine using cloud-init	189
8.17.5. Configuring an SR-IOV network device for virtual machines	190
8.17.5.1. Prerequisites	190
8.17.5.2. Automated discovery of SR-IOV network devices	191
8.17.5.2.1. Example SrioVNetworkNodeState object	191
8.17.5.3. Configuring SR-IOV network devices	192
8.17.5.4. Next steps	194
8.17.6. Defining an SR-IOV network	194
8.17.6.1. Prerequisites	194
8.17.6.2. Configuring SR-IOV additional network	194
8.17.6.3. Next steps	196
8.17.7. Attaching a virtual machine to an SR-IOV network	196
8.17.7.1. Prerequisites	197
8.17.7.2. Attaching a virtual machine to an SR-IOV network	197
8.17.8. Viewing the IP address of NICs on a virtual machine	198
8.17.8.1. Viewing the IP address of a virtual machine interface in the CLI	198
8.17.8.2. Viewing the IP address of a virtual machine interface in the web console	198
8.17.9. Using a MAC address pool for virtual machines	199
8.17.9.1. About KubeMacPool	199
8.17.9.2. Disabling a MAC address pool for a namespace in the CLI	199
8.17.9.3. Re-enabling a MAC address pool for a namespace in the CLI	199
8.18. VIRTUAL MACHINE DISKS	200
8.18.1. Storage features	200
8.18.1.1. OpenShift Virtualization storage feature matrix	200
8.18.2. Configuring local storage for virtual machines	201
8.18.2.1. About the hostpath provisioner	201
8.18.2.2. Configuring SELinux for the hostpath provisioner on Red Hat Enterprise Linux CoreOS (RHCOS)	201
8.18.2.3. Using the hostpath provisioner to enable local storage	203
8.18.2.4. Creating a storage class	204
8.18.3. Creating data volumes using profiles	205

8.18.3.1. Creating data volumes using the storage API	206
8.18.3.2. Creating data volumes using the PVC API	207
8.18.3.3. Customizing the storage profile	208
8.18.3.4. Additional resources	210
8.18.4. Reserving PVC space for file system overhead	210
8.18.4.1. How file system overhead affects space for virtual machine disks	210
8.18.4.2. Overriding the default file system overhead value	210
8.18.5. Configuring CDI to work with namespaces that have a compute resource quota	211
8.18.5.1. About CPU and memory quotas in a namespace	211
8.18.5.2. Overriding CPU and memory defaults	211
8.18.5.3. Additional resources	212
8.18.6. Managing data volume annotations	212
8.18.6.1. Example: Data volume annotations	212
8.18.7. Using preallocation for data volumes	213
8.18.7.1. About preallocation	213
8.18.7.2. Enabling preallocation for a data volume	213
8.18.8. Uploading local disk images by using the web console	214
8.18.8.1. Prerequisites	214
8.18.8.2. CDI supported operations matrix	214
8.18.8.3. Uploading an image file using the web console	214
8.18.8.4. Additional resources	215
8.18.9. Uploading local disk images by using the virtctl tool	216
8.18.9.1. Prerequisites	216
8.18.9.2. About data volumes	216
8.18.9.3. Creating an upload data volume	216
8.18.9.4. Uploading a local disk image to a data volume	217
8.18.9.5. CDI supported operations matrix	218
8.18.9.6. Additional resources	218
8.18.10. Uploading a local disk image to a block storage data volume	219
8.18.10.1. Prerequisites	219
8.18.10.2. About data volumes	219
8.18.10.3. About block persistent volumes	219
8.18.10.4. Creating a local block persistent volume	219
8.18.10.5. Creating an upload data volume	220
8.18.10.6. Uploading a local disk image to a data volume	221
8.18.10.7. CDI supported operations matrix	222
8.18.10.8. Additional resources	223
8.18.11. Managing offline virtual machine snapshots	223
8.18.11.1. About virtual machine snapshots	223
8.18.11.1.1. Virtual machine snapshot controller and custom resource definitions (CRDs)	224
8.18.11.2. Creating an offline virtual machine snapshot in the web console	224
8.18.11.3. Creating an offline virtual machine snapshot in the CLI	224
8.18.11.4. Restoring a virtual machine from a snapshot in the web console	226
8.18.11.5. Restoring a virtual machine from a snapshot in the CLI	227
8.18.11.6. Deleting a virtual machine snapshot in the web console	229
8.18.11.7. Deleting a virtual machine snapshot in the CLI	229
8.18.11.8. Additional resources	230
8.18.12. Moving a local virtual machine disk to a different node	230
8.18.12.1. Cloning a local volume to another node	230
8.18.13. Expanding virtual storage by adding blank disk images	233
8.18.13.1. About data volumes	233
8.18.13.2. Creating a blank disk image with data volumes	233
8.18.13.3. Template: Data volume configuration file for blank disk images	234

8.18.13.4. Additional resources	234
8.18.14. Cloning a data volume using smart-cloning	234
8.18.14.1. Understanding smart-cloning	234
8.18.14.2. Cloning a data volume	234
8.18.14.3. Additional resources	236
8.18.15. Creating and using boot sources	236
8.18.15.1. About virtual machines and boot sources	236
8.18.15.2. Importing a Red Hat Enterprise Linux image as a boot source	236
8.18.15.3. Adding a boot source for a virtual machine template	237
8.18.15.4. Creating a virtual machine from a template with an attached boot source	239
8.18.15.5. Creating a custom boot source	239
8.18.15.6. Additional resources	240
8.18.16. Hot-plugging virtual disks	240
8.18.16.1. Hot-plugging a virtual disk using the CLI	241
8.18.16.2. Hot-unplugging a virtual disk using the CLI	242
8.18.17. Using container disks with virtual machines	242
8.18.17.1. About container disks	242
8.18.17.1.1. Importing a container disk into a PVC by using a data volume	243
8.18.17.1.2. Attaching a container disk to a virtual machine as a containerDisk volume	243
8.18.17.2. Preparing a container disk for virtual machines	243
8.18.17.3. Disabling TLS for a container registry to use as insecure registry	244
8.18.17.4. Next steps	244
8.18.18. Preparing CDI scratch space	245
8.18.18.1. About data volumes	245
8.18.18.2. Understanding scratch space	245
Manual provisioning	245
8.18.18.3. CDI operations that require scratch space	245
8.18.18.4. Defining a storage class	246
8.18.18.5. CDI supported operations matrix	246
8.18.18.6. Additional resources	247
8.18.19. Re-using persistent volumes	247
8.18.19.1. About reclaiming statically provisioned persistent volumes	247
8.18.19.2. Reclaiming statically provisioned persistent volumes	247
8.18.20. Deleting data volumes	249
8.18.20.1. About data volumes	249
8.18.20.2. Listing all data volumes	249
8.18.20.3. Deleting a data volume	249
CHAPTER 9. VIRTUAL MACHINE TEMPLATES	250
9.1. CREATING VIRTUAL MACHINE TEMPLATES	250
9.1.1. About virtual machine templates	250
9.1.2. About virtual machines and boot sources	250
9.1.3. Adding a boot source for a virtual machine template	251
9.1.3.1. Virtual machine template fields for adding a boot source	252
9.1.4. Marking virtual machine templates as favorites	254
9.1.5. Filtering the list of virtual machine templates by providers	254
9.1.6. Creating a virtual machine template with the wizard in the web console	254
9.1.7. Virtual machine template wizard fields	256
9.1.7.1. Virtual machine template wizard fields	256
9.1.7.2. Networking fields	258
9.1.7.3. Storage fields	258
Advanced storage settings	260
9.1.7.4. Cloud-init fields	260

9.1.8. Additional resources	261
9.2. EDITING VIRTUAL MACHINE TEMPLATES	261
9.2.1. Editing a virtual machine template in the web console	261
9.2.2. Editing virtual machine template YAML configuration in the web console	261
9.2.3. Adding a virtual disk to a virtual machine template	262
9.2.4. Adding a network interface to a virtual machine template	262
9.2.5. Editing CD-ROMs for Templates	263
9.3. ENABLING DEDICATED RESOURCES FOR VIRTUAL MACHINE TEMPLATES	263
9.3.1. About dedicated resources	263
9.3.2. Prerequisites	263
9.3.3. Enabling dedicated resources for a virtual machine template	263
9.4. DELETING A VIRTUAL MACHINE TEMPLATE	264
9.4.1. Deleting a virtual machine template in the web console	264
CHAPTER 10. LIVE MIGRATION	265
10.1. VIRTUAL MACHINE LIVE MIGRATION	265
10.1.1. Understanding live migration	265
10.1.2. Updating access mode for live migration	265
10.2. LIVE MIGRATION LIMITS AND TIMEOUTS	265
10.2.1. Configuring live migration limits and timeouts	265
10.2.2. Cluster-wide live migration limits and timeouts	266
10.3. MIGRATING A VIRTUAL MACHINE INSTANCE TO ANOTHER NODE	267
10.3.1. Initiating live migration of a virtual machine instance in the web console	267
10.3.2. Initiating live migration of a virtual machine instance in the CLI	267
10.4. MONITORING LIVE MIGRATION OF A VIRTUAL MACHINE INSTANCE	268
10.4.1. Monitoring live migration of a virtual machine instance in the web console	268
10.4.2. Monitoring live migration of a virtual machine instance in the CLI	268
10.5. CANCELLING THE LIVE MIGRATION OF A VIRTUAL MACHINE INSTANCE	269
10.5.1. Cancelling live migration of a virtual machine instance in the web console	269
10.5.2. Cancelling live migration of a virtual machine instance in the CLI	269
10.6. CONFIGURING VIRTUAL MACHINE EVICTION STRATEGY	270
10.6.1. Configuring custom virtual machines with the LiveMigration eviction strategy	270
CHAPTER 11. NODE MAINTENANCE	271
11.1. ABOUT NODE MAINTENANCE	271
11.1.1. Understanding node maintenance mode	271
11.1.2. Maintaining bare metal nodes	271
11.2. SETTING A NODE TO MAINTENANCE MODE	272
11.2.1. Setting a node to maintenance mode in the web console	272
11.2.2. Setting a node to maintenance mode in the CLI	272
11.2.3. Setting a node to maintenance mode with a NodeMaintenance custom resource	273
11.2.3.1. Checking status of current NodeMaintenance CR tasks	274
11.3. RESUMING A NODE FROM MAINTENANCE MODE	275
11.3.1. Resuming a node from maintenance mode in the web console	275
11.3.2. Resuming a node from maintenance mode in the CLI	275
11.3.3. Resuming a node from maintenance mode that was initiated with a NodeMaintenance CR	275
11.4. AUTOMATIC RENEWAL OF TLS CERTIFICATES	276
11.4.1. TLS certificates automatic renewal schedules	276
11.5. MANAGING NODE LABELING FOR OBSOLETE CPU MODELS	276
11.5.1. About node labeling for obsolete CPU models	276
11.5.2. About node labeling for CPU features	277
11.5.3. Configuring obsolete CPU models	279
11.6. PREVENTING NODE RECONCILIATION	280

11.6.1. Using skip-node annotation	280
11.6.2. Additional resources	280
CHAPTER 12. NODE NETWORKING	281
12.1. OBSERVING NODE NETWORK STATE	281
12.1.1. About nmstate	281
12.1.2. Viewing the network state of a node	281
12.2. UPDATING NODE NETWORK CONFIGURATION	282
12.2.1. About nmstate	282
12.2.2. Creating an interface on nodes	283
Additional resources	284
12.2.3. Confirming node network policy updates on nodes	284
12.2.4. Removing an interface from nodes	285
12.2.5. Example policy configurations for different interfaces	286
12.2.5.1. Example: Linux bridge interface node network configuration policy	286
12.2.5.2. Example: VLAN interface node network configuration policy	287
12.2.5.3. Example: Bond interface node network configuration policy	288
12.2.5.4. Example: Ethernet interface node network configuration policy	289
12.2.5.5. Example: Multiple interfaces in the same node network configuration policy	290
12.2.6. Examples: IP management	291
12.2.6.1. Static	291
12.2.6.2. No IP address	291
12.2.6.3. Dynamic host configuration	291
12.2.6.4. DNS	292
12.2.6.5. Static routing	292
12.3. TROUBLESHOOTING NODE NETWORK CONFIGURATION	293
12.3.1. Troubleshooting an incorrect node network configuration policy configuration	293
CHAPTER 13. LOGGING, EVENTS, AND MONITORING	298
13.1. VIEWING VIRTUAL MACHINE LOGS	298
13.1.1. Understanding virtual machine logs	298
13.1.2. Viewing virtual machine logs in the CLI	298
13.1.3. Viewing virtual machine logs in the web console	298
13.2. VIEWING EVENTS	298
13.2.1. Understanding virtual machine events	299
13.2.2. Viewing the events for a virtual machine in the web console	299
13.2.3. Viewing namespace events in the CLI	299
13.2.4. Viewing resource events in the CLI	299
13.3. DIAGNOSING DATA VOLUMES USING EVENTS AND CONDITIONS	300
13.3.1. About conditions and events	300
13.3.2. Analyzing data volumes using conditions and events	300
13.4. VIEWING INFORMATION ABOUT VIRTUAL MACHINE WORKLOADS	302
13.4.1. About the Virtual Machines dashboard	302
13.5. MONITORING VIRTUAL MACHINE HEALTH	303
13.5.1. About readiness and liveness probes	303
13.5.2. Defining an HTTP readiness probe	303
13.5.3. Defining a TCP readiness probe	304
13.5.4. Defining an HTTP liveness probe	305
13.5.5. Template: Virtual machine instance configuration file for defining health checks	306
13.5.6. Additional resources	307
13.6. USING THE OPENSIFT CONTAINER PLATFORM DASHBOARD TO GET CLUSTER INFORMATION	307
13.6.1. About the OpenShift Container Platform dashboards page	307
13.7. OPENSIFT CONTAINER PLATFORM CLUSTER MONITORING, LOGGING, AND TELEMETRY	308

13.7.1. About OpenShift Container Platform monitoring	308
13.7.2. About OpenShift Logging components	309
13.7.3. About Telemetry	309
13.7.3.1. Information collected by Telemetry	309
13.7.4. CLI troubleshooting and debugging commands	310
13.8. PROMETHEUS QUERIES FOR VIRTUAL RESOURCES	310
13.8.1. Prerequisites	310
13.8.2. Querying metrics	311
13.8.2.1. Querying metrics for all projects as a cluster administrator	311
13.8.2.2. Querying metrics for user-defined projects as a developer	312
13.8.3. Virtualization metrics	313
13.8.3.1. vCPU metrics	313
13.8.3.2. Network metrics	313
13.8.3.3. Storage metrics	314
13.8.3.3.1. Storage-related traffic	314
13.8.3.3.2. I/O performance	314
13.8.3.4. Guest memory swapping metrics	314
13.8.4. Additional resources	315
13.9. COLLECTING DATA FOR RED HAT SUPPORT	315
13.9.1. Collecting data about your environment	315
13.9.1.1. Additional resources	316
13.9.2. Collecting data about virtual machines	316
13.9.2.1. Additional resources	316
13.9.3. Using the must-gather tool for OpenShift Virtualization	317
13.9.3.1. must-gather tool options	317
13.9.3.1.1. Parameters	317
13.9.3.1.2. Usage and examples	318
13.9.3.2. Additional resources	319

CHAPTER 1. ABOUT OPENSIFT VIRTUALIZATION

Learn about OpenShift Virtualization's capabilities and support scope.

1.1. WHAT YOU CAN DO WITH OPENSIFT VIRTUALIZATION

OpenShift Virtualization is an add-on to OpenShift Container Platform that allows you to run and manage virtual machine workloads alongside container workloads.

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

- Creating and managing Linux and Windows virtual machines
- Connecting to virtual machines through a variety of consoles and CLI tools
- Importing and cloning existing virtual machines
- Managing network interface controllers and storage disks attached to virtual machines
- Live migrating virtual machines between nodes

An enhanced web console provides a graphical portal to manage these virtualized resources alongside the OpenShift Container Platform cluster containers and infrastructure.

OpenShift Virtualization is tested with OpenShift Container Storage (OCS) and designed to use with OCS features for the best experience.

You can use OpenShift Virtualization with the [OVN-Kubernetes](#), [OpenShift SDN](#), or one of the other certified default Container Network Interface (CNI) network providers listed in [Certified OpenShift CNI Plugins](#).

1.1.1. OpenShift Virtualization supported cluster version

OpenShift Virtualization 4.8 is supported for use on OpenShift Container Platform 4.8 clusters. To use the latest z-stream release of OpenShift Virtualization, you must first upgrade to the latest version of OpenShift Container Platform.

CHAPTER 2. START HERE WITH OPENSIFT VIRTUALIZATION

Use the following tables to find content to help you learn about and use OpenShift Virtualization.

2.1. CLUSTER ADMINISTRATOR

Learn	Plan	Deploy	Additional resources
Learn about OpenShift Virtualization	Configuring your cluster for OpenShift Virtualization	Updating your node network configuration	Getting Support
Learn more about OpenShift Container Platform	Plan storage for virtual machine disks	Configuring CSI volumes	
Learn about virtual machine live migration		Installing OpenShift Virtualization using the OpenShift Virtualization console or CLI	
Learn about node maintenance			

2.2. VIRTUALIZATION ADMINISTRATOR

Learn	Deploy	Manage	Use
Learn about OpenShift Virtualization	Connecting virtual machines to the default pod network for virtual machines and external networks	Installing the virtctl client	Importing virtual machines with the Migration Toolkit for containers
Learn about storage features for virtual machine disks	Customizing the storage profile	Using the CLI tools	Using live migration
	Creating boot sources and attaching them to templates	Viewing logs and events	
	Updating boot source templates	Monitoring virtual machine health	

2.3. VIRTUAL MACHINE ADMINISTRATOR / DEVELOPER

Learn	Use	Manage	Additional resources
Learn about OpenShift Virtualization	Installing the virtctl client	Viewing logs and events	Getting Support
	Creating virtual machines	Monitoring virtual machine health	
	Managing virtual machines instances	Creating and managing virtual machine snapshots	
	Controlling virtual machine states		
	Accessing the virtual machine consoles		
	Pass configuration data to virtual machines using secrets, configuration maps, and service accounts		

CHAPTER 3. OPENSIFT VIRTUALIZATION RELEASE NOTES

3.1. ABOUT RED HAT OPENSIFT VIRTUALIZATION

Red Hat OpenShift Virtualization enables you to bring traditional virtual machines (VMs) into OpenShift Container Platform where they run alongside containers, and are managed as native Kubernetes objects.



OpenShift Virtualization is represented by the  logo.

You can use OpenShift Virtualization with either the [OVN-Kubernetes](#) or the [OpenShiftSDN](#) default Container Network Interface (CNI) network provider.

Learn more about [what you can do with OpenShift Virtualization](#).

3.1.1. OpenShift Virtualization supported cluster version

OpenShift Virtualization 4.8 is supported for use on OpenShift Container Platform 4.8 clusters. To use the latest z-stream release of OpenShift Virtualization, you must first upgrade to the latest version of OpenShift Container Platform.

3.1.2. Supported guest operating systems

OpenShift Virtualization guests can use the following operating systems:

- Red Hat Enterprise Linux 6, 7, and 8.
- Microsoft Windows Server 2012 R2, 2016, and 2019.
- Microsoft Windows 10.

Other operating system templates shipped with OpenShift Virtualization are not supported.

3.2. MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

3.3. NEW AND CHANGED FEATURES

- 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 on RHEL CoreOS*.
 - Intel and AMD CPUs.
- The Containerized Data Importer (CDI) now uses the OpenShift Container Platform [cluster-wide proxy configuration](#).

- OpenShift Virtualization now supports third-party Container Network Interface (CNI) plug-ins that are [certified by Red Hat](#) for use with OpenShift Container Platform.
- OpenShift Virtualization now provides metrics for monitoring how infrastructure resources are consumed in the cluster. You can use the OpenShift Container Platform monitoring dashboard to [query metrics](#) for the following resources:
 - vCPU
 - Network
 - Storage
 - Guest memory swapping
- OpenShift Virtualization now provides a unified API to [configure certificate rotation](#).
- If a Windows virtual machine is created from a template or has predefined Hyper-V capabilities, it can now only be scheduled to Hyper-V capable nodes.
- The **--proxy-only** option for the **virtctl vnc** command allows you to [manually connect to a virtual machine instance](#) through a Virtual Network Client (VNC) connection by using any VNC viewer.

3.3.1. 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 Virtualization console and then select **Quick Starts**. You can filter the available tours by entering the **virtualization** keyword in the **Filter** field.

3.3.2. Networking

- You can use the Kubernetes NMstate Operator to [configure and manage IP addresses](#) on your cluster nodes.
- OpenShift Virtualization now supports [live migration of virtual machines](#) that are attached to an SR-IOV network interface if the **sriovLiveMigration** feature gate is enabled in the **HyperConverged** custom resource (CR).

3.3.3. Storage

- Cloning a data volume into a different namespace is now faster and more efficient when using storage that supports Container Storage Interface (CSI) snapshots. The Containerized Data Importer (CDI) uses CSI snapshots, when they are available, to improve performance when you create a virtual machine from a template.
- When the **fstrim** or **blkdiscard** commands are run on a virtual disk, the discard requests are passed to the underlying storage device. If the storage provider supports the Pass Discard feature, the discard requests free up storage capacity.
- You can now specify data volumes by using the storage API. The storage API, unlike the PVC API, allows the system to optimize **accessModes**, **volumeMode**, and storage capacity when allocating storage.

- You can now [clone virtual machine disks between different data volume modes](#) if they have the content type **kubevirt**. For example, you can clone a persistent volume (PV) with **volumeMode: Block** to a PV with **volumeMode: Filesystem**.
- You can [create a custom disk image as a boot source](#) for any template that has a defined source by running a wizard in the OpenShift Virtualization console.

3.4. DEPRECATED AND REMOVED FEATURES

3.4.1. 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.

- Importing a single virtual machine from Red Hat Virtualization (RHV) or VMware is deprecated in the current release and will be removed in OpenShift Virtualization 4.9. This feature is replaced by the [Migration Toolkit for Virtualization](#).

3.5. NOTABLE TECHNICAL CHANGES

- OpenShift Virtualization now configures IPv6 addresses when running on clusters that have dual-stack networking enabled. You can [create a service](#) that uses IPv4, IPv6, or both IP address families, if dual-stack networking is enabled for the underlying OpenShift Container Platform cluster.
- KubeMacPool is now enabled by default when you install OpenShift Virtualization. You can [disable a MAC address pool](#) for a namespace by adding the **mutatevirtualmachines.kubemacpool.io=ignore** label to the namespace. Re-enable KubeMacPool for the namespace by removing the label.
- The **HyperConverged** custom resource (CR) is now the central point of configuration for OpenShift Virtualization. By editing the **HyperConverged** CR, you can:
 - [Configure live migration limits and timeouts](#)
 - [Define the location of a VMware Virtual Disk Development Kit \(VDDK\) image](#)
 - [Configure obsolete CPU models](#)
 - [Disable TLS for container registries](#)
 - [Configure a storage class for scratch space](#)
 - [Configure resource requirements for storage workloads](#)

3.6. 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 now [hot-plug and hot-unplug virtual disks](#) when you want to add or remove them from your virtual machine without stopping the virtual machine instance.

3.7. KNOWN ISSUES

- Updating to OpenShift Virtualization 4.8.7 causes some virtual machines (VMs) to get stuck in a live migration loop. This occurs if the **spec.volumes.containerDisk.path** field in the VM manifest is set to a relative path.
 - As a workaround, delete and recreate the VM manifest, setting the value of the **spec.volumes.containerDisk.path** field to an absolute path. You can then update OpenShift Virtualization.
- If you initially deployed OpenShift Virtualization version 2.4.z or earlier, upgrading to version 4.8 fails with the following message:

```
risk of data loss updating hyperconvergeds.hco.kubevirt.io: new CRD removes
version v1alpha1 that is listed as a stored version on the existing CRD
```

This bug does not affect clusters where OpenShift Virtualization was initially deployed at version 2.5.0 or later. ([BZ#1986989](#))

- As a workaround, remove the **v1alpha1** version from the **HyperConverged** custom resource definition (CRD) and resume the upgrade process:

1. Open a proxy connection to the cluster by running the following command:

```
$ oc proxy &
```

2. Remove the **v1alpha1** version from **.status.storedVersions** on the **HyperConverged** CRD by running the following command:

```
$ curl --header "Content-Type: application/json-patch+json" --request PATCH
http://localhost:8001/apis/apiextensions.k8s.io/v1/customresourcedefinitions/hyperconvergeds.hco.kubevirt.io/status --data [{"op": "replace", "path": "/status/storedVersions", "value": ["v1beta1"]}]"
```

3. Resume the upgrade process by running the following command:

```
$ curl --header "Content-Type: application/json-patch+json" --request PATCH
http://localhost:8001/apis/operators.coreos.com/v1alpha1/namespaces/openshift-cnv/installplans/(oc get installplan -n openshift-cnv | grep kubevirt-hyperconverged-operator.v4.8.0 | cut -d' ' -f1)/status --data [{"op": "remove", "path": "/status/conditions"}, {"op": "remove", "path": "/status/message"}, {"op": "replace", "path": "/status/phase", "value": "Installing"}]"
```

4. Kill the **oc proxy** process by running the following command:

```
$ kill $(ps -C "oc proxy" -o pid=)
```

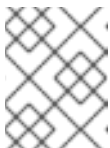
5. Optional: Monitor the upgrade status by running the following command:

```
$ oc get csv
```

- If you delete OpenShift Virtualization-provided templates in version 4.8 or later, the templates are automatically recreated by the OpenShift Virtualization Operator. However, if you delete OpenShift Virtualization-provided templates created before version 4.8, those earlier

templates are not automatically recreated after deletion. As a result, any edit or update to a virtual machine referencing a deleted earlier template will fail.

- If a cloning operation is initiated before the source is available to be cloned, the operation stalls indefinitely. This is because the clone authorization expires before the cloning operation starts. ([BZ#1855182](#))
 - As a workaround, delete the **DataVolume** object that is requesting the clone. When the source is available, recreate the **DataVolume** object that you deleted so that the cloning operation can complete successfully.
- If your OpenShift Container Platform cluster uses OVN-Kubernetes as the default Container Network Interface (CNI) provider, you cannot attach a Linux bridge or bonding to the default interface of a host because of a change in the host network topology of OVN-Kubernetes. ([BZ#1885605](#))
 - As a workaround, you can use a secondary network interface connected to your host, or switch to the OpenShift SDN default CNI provider.
- Running virtual machines that cannot be live migrated might block an OpenShift Container Platform cluster upgrade. This includes virtual machines that use hostpath-provisioner storage or SR-IOV network interfaces. ([BZ#1858777](#))
 - As a workaround, you can reconfigure the virtual machines so that they can be powered off during a cluster upgrade. In the **spec** section of the virtual machine configuration file:
 1. Remove the **evictionStrategy: LiveMigrate** field. See [Configuring virtual machine eviction strategy](#) for more information on how to configure eviction strategy.
 2. Set the **runStrategy** field to **Always**.
- Live migration fails when nodes have different CPU models. Even in cases where nodes have the same physical CPU model, differences introduced by microcode updates have the same effect. This is because the default settings trigger host CPU passthrough behavior, which is incompatible with live migration. ([BZ#1760028](#))
 - As a workaround, set the default CPU model by running the following command:



NOTE

You must make this change before starting the virtual machines that support live migration.

```
$ oc annotate --overwrite -n openshift-cnv hyperconverged kubevirt-hyperconverged
kubevirt.kubevirt.io/jsonpatch='[
  {
    "op": "add",
    "path": "/spec/configuration/cpuModel",
    "value": "<cpu_model>" 1
  }
]'
```

- 1 Replace **<cpu_model>** with the actual CPU model value. You can determine this value by running **oc describe node <node>** for all nodes and looking at the **cpu-model-
<name>** labels. Select the CPU model that is present on all of your nodes.

- If you enter the wrong credentials for the RHV Manager while importing a RHV VM, the Manager might lock the admin user account because the **vm-import-operator** tries repeatedly to connect to the RHV API. ([BZ#1887140](#))
 - To unlock the account, log in to the Manager and enter the following command:

```
$ ovirt-aaa-jdbc-tool user unlock admin
```

- If you run OpenShift Virtualization 2.6.5 with OpenShift Container Platform 4.8, various issues occur. You can avoid these issues by upgrading OpenShift Virtualization to version 4.8.
 - In the web console, if you navigate to the **Virtualization** page and select **Create → With YAML** the following error message is displayed:

```
The server doesn't have a resource type "kind: VirtualMachine, apiVersion:
kubevirt.io/v1"
```

- As a workaround, edit the **VirtualMachine** manifest so the **apiVersion** is **kubevirt.io/v1alpha3**. For example:

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  annotations:
  ...
```

([BZ#1979114](#))

- If you use the **Customize** wizard to create a VM, the following error message is displayed:

```
Error creating virtual machine
```

 - As a workaround, copy the manifest and [create the virtual machine from the CLI](#). ([BZ#1979116](#))
- When connecting to the VNC console by using the OpenShift Virtualization web console, the VNC console always fails to respond.
 - As a workaround, create the virtual machine from the CLI or upgrade to OpenShift Virtualization 4.8. ([BZ#1977037](#))

CHAPTER 4. INSTALLING OPENSIFT VIRTUALIZATION

4.1. PREPARING YOUR CLUSTER FOR OPENSIFT VIRTUALIZATION

Review this section before you install OpenShift Virtualization to ensure that your cluster meets the requirements.



IMPORTANT

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.

FIPS mode

If you install your cluster in [FIPS mode](#), no additional setup is required for OpenShift Virtualization.

4.1.1. Hardware and operating system requirements

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

Supported platforms

- On-premise bare metal servers
- Amazon Web Services bare metal instances



IMPORTANT

Installing OpenShift Virtualization on an AWS bare metal instance 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 <https://access.redhat.com/support/offerings/techpreview/>.

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

CPU requirements

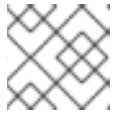
- Supported by Red Hat Enterprise Linux (RHEL) 8
- Support for Intel 64 or AMD64 CPU extensions
- Intel VT or AMD-V hardware virtualization extensions enabled
- NX (no execute) flag enabled

Storage requirements

- Supported by OpenShift Container Platform

Operating system requirements

- Red Hat Enterprise Linux CoreOS (RHCOS) installed on worker nodes



NOTE

RHEL worker nodes are not supported.

Additional resources

- [About RHCOS](#)
- [Red Hat Ecosystem Catalog](#) for supported CPUs
- [Supported storage](#)

4.1.2. 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.

4.1.2.1. Memory overhead

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

Cluster memory overhead

Memory overhead per infrastructure node \approx 150 MiB

Memory overhead per worker node \approx 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 \approx $(1.002 * \text{requested memory}) + 146 \text{ MiB} \setminus$
 $+ 8 \text{ MiB} * (\text{number of vCPUs}) \setminus$ **1**
 $+ 16 \text{ MiB} * (\text{number of graphics devices})$ **2**

1 Number of virtual CPUs requested by the virtual machine

2 Number of virtual graphics cards requested by the virtual machine

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.

4.1.2.2. 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 \approx 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 \approx 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.

4.1.2.3. Storage overhead

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

Cluster storage overhead

Aggregated storage overhead per node \approx 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.

4.1.2.4. 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.

4.1.3. Object maximums

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

- [OpenShift Container Platform object maximums](#)
- [OpenShift Virtualization object maximums](#)

4.1.4. Restricted network environments

If you install OpenShift Virtualization in a restricted environment with no internet connectivity, you must [configure Operator Lifecycle Manager for restricted networks](#).

If you have limited internet connectivity, you can [configure proxy support in Operator Lifecycle Manager](#) to access the Red Hat-provided OperatorHub.

4.1.5. Live migration

Live migration has the following requirements:

- Shared storage with **ReadWriteMany** (RWX) access mode
- Sufficient RAM and network bandwidth
- Appropriate CPUs with sufficient capacity on the worker nodes. If the CPUs have different capacities, live migration might be very slow or fail.

4.1.6. Snapshots and cloning

See [OpenShift Virtualization storage features](#) for snapshot and cloning requirements.

4.1.7. Cluster high-availability options

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

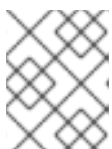
- Automatic high availability for [installer-provisioned infrastructure](#) (IPI) is available by deploying [machine health checks](#).



NOTE

In OpenShift Container Platform clusters installed using installer-provisioned infrastructure and with MachineHealthCheck properly configured, if a node fails the MachineHealthCheck 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 [About RunStrategies for virtual machines](#) for more detailed information about the potential outcomes and how RunStrategies affect those outcomes.

- 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. SPECIFYING NODES FOR OPENSIFT VIRTUALIZATION COMPONENTS

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 for some components after installing OpenShift Virtualization, but there must not be virtual machines present if you want to configure node placement for workloads.

4.2.1. About node placement for virtualization components

You might want to customize where OpenShift Virtualization deploys its components to ensure that:

- Virtual machines only deploy on nodes that are intended for virtualization workloads.
- Operators only deploy on infrastructure nodes.
- Certain nodes are unaffected by OpenShift Virtualization. For example, you have workloads unrelated to virtualization running on your cluster, and you want those workloads to be isolated from OpenShift Virtualization.

4.2.1.1. How to apply node placement rules to virtualization components

You can specify node placement rules for a component by editing the corresponding object directly or by using the web console.

- For the OpenShift Virtualization Operators that Operator Lifecycle Manager (OLM) deploys, edit the OLM **Subscription** object directly. Currently, you cannot configure node placement rules for the **Subscription** object by using the web console.
- For components that the OpenShift Virtualization Operators deploy, edit the **HyperConverged** object directly or configure it by using the web console during OpenShift Virtualization installation.
- For the hostpath provisioner, edit the **HostPathProvisioner** object directly or configure it by using the web console.



WARNING

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

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, rather than a hard requirement, so that pods are still scheduled if 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.

4.2.1.2. Node placement in the OLM Subscription object

To specify the nodes where OLM deploys the OpenShift Virtualization Operators, edit the **Subscription** object during OpenShift Virtualization installation. You can include node placement rules in the **spec.config** field, as shown in the following example:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cn
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.8.7
  channel: "stable"
  config: 1
```

1 The **config** field supports **nodeSelector** and **tolerations**, but it does not support **affinity**.

4.2.1.3. Node placement in the HyperConverged object

To specify the nodes where OpenShift Virtualization deploys its components, you can include the **nodePlacement** object in the HyperConverged Cluster custom resource (CR) file that you create during OpenShift Virtualization installation. You can include **nodePlacement** under the **spec.infra** and **spec.workloads** fields, as shown in the following example:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cn
spec:
  infra:
    nodePlacement: 1
    ...
  workloads:
    nodePlacement:
    ...
```

1 The **nodePlacement** fields support **nodeSelector**, **affinity**, and **tolerations** fields.

4.2.1.4. Node placement in the HostPathProvisioner object

You can configure node placement rules in the **spec.workload** field of the **HostPathProvisioner** object that you create when you install the hostpath provisioner.

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

❶ The **workload** field supports **nodeSelector**, **affinity**, and **tolerations** fields.

4.2.1.5. 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](#)
- [Installing OpenShift Virtualization using the CLI](#)
- [Installing OpenShift Virtualization using the web console](#)
- [Configuring local storage for virtual machines](#)

4.2.2. Example manifests

The following example YAML files use **nodePlacement**, **affinity**, and **tolerations** objects to customize node placement for OpenShift Virtualization components.

4.2.2.1. Operator Lifecycle Manager Subscription object

4.2.2.1.1. Example: Node placement with nodeSelector in the OLM Subscription object

In this example, **nodeSelector** is configured so that OLM places the OpenShift Virtualization Operators on nodes that are labeled with **example.io/example-infra-key = example-infra-value**.

```
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.v4.8.7
channel: "stable"
config:
  nodeSelector:
    example.io/example-infra-key: example-infra-value

```

4.2.2.1.2. Example: Node placement with tolerations in the OLM Subscription object

In this example, nodes that are reserved for OLM to deploy OpenShift Virtualization Operators are labeled with the **key=virtualization:NoSchedule** taint. Only pods with the matching tolerations are scheduled to these nodes.

```

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.v4.8.7
  channel: "stable"
  config:
    tolerations:
      - key: "key"
        operator: "Equal"
        value: "virtualization"
        effect: "NoSchedule"

```

4.2.2.2. HyperConverged object

4.2.2.2.1. Example: Node placement with nodeSelector in the HyperConverged Cluster CR

In this example, **nodeSelector** is configured so that infrastructure resources are placed on nodes that are labeled with **example.io/example-infra-key = example-infra-value** and workloads are placed on nodes labeled with **example.io/example-workloads-key = example-workloads-value**.

```

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
  workloads:
    nodePlacement:
      nodeSelector:
        example.io/example-workloads-key: example-workloads-value

```

4.2.2.2.2. Example: Node placement with affinity in the HyperConverged Cluster CR

In this example, **affinity** is configured so that infrastructure resources are placed on nodes that are labeled with **example.io/example-infra-key = example-value** and workloads are placed on nodes labeled with **example.io/example-workloads-key = example-workloads-value**. Nodes that have more than eight CPUs are preferred for workloads, but if they are not available, pods are still scheduled.

```

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
  workloads:
    nodePlacement:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: example.io/example-workloads-key
                    operator: In
                    values:
                      - example-workloads-value
            preferredDuringSchedulingIgnoredDuringExecution:
              - weight: 1
                preference:
                  matchExpressions:
                    - key: example.io/num-cpus
                      operator: Gt
                      values:
                        - 8

```

4.2.2.2.3. Example: Node placement with tolerations in the HyperConverged Cluster CR

In this example, nodes that are reserved for OpenShift Virtualization components are labeled with the **key=virtualization:NoSchedule** taint. Only pods with the matching tolerations are scheduled to these nodes.

```

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

```

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

4.2.2.3. HostPathProvisioner object

4.2.2.3.1. Example: Node placement with nodeSelector in the HostPathProvisioner object

In this example, **nodeSelector** is configured so that workloads are placed on nodes labeled with **example.io/example-workloads-key = example-workloads-value**.

```
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
```

4.3. INSTALLING OPENSIFT VIRTUALIZATION USING THE WEB CONSOLE

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

You can use the OpenShift Container Platform 4.8 [web console](#) to subscribe to and deploy the OpenShift Virtualization Operators.

4.3.1. Installing the OpenShift Virtualization Operator

You can install the OpenShift Virtualization Operator from the OpenShift Container Platform web console.

Prerequisites

- Install OpenShift Container Platform 4.8 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 **Operators** → **OperatorHub**.

2. In the **Filter by keyword** field, type **OpenShift Virtualization**.
3. Select the **OpenShift Virtualization** tile.
4. Read the information about the Operator and click **Install**.
5. On the **Install Operator** page:
 - a. Select **stable** from the list of available **Update Channel** options. 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.
While it is possible to select the **Manual** approval strategy, this is inadvisable because of the high risk that it presents to the supportability and functionality of your cluster. 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.

4.3.2. Next steps

You might want to additionally configure the following components:

- 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.

Install OpenShift Virtualization to add virtualization functionality to your OpenShift Container Platform cluster. You can subscribe to and deploy the OpenShift Virtualization Operators by using the command line to apply manifests to your cluster.



NOTE

To specify the nodes where you want OpenShift Virtualization to install its components, [configure node placement rules](#).

4.3.3. Prerequisites

- Install OpenShift Container Platform 4.8 on your cluster.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

4.3.4. 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.

Procedure

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

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-cnv
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: kubevirt-hyperconverged-group
  namespace: openshift-cnv
spec:
  targetNamespaces:
    - openshift-cnv
---
```

```

apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cn
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubvirt-hyperconverged
  startingCSV: kubvirt-hyperconverged-operator.v4.8.7
  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 <file name>.yaml
```



NOTE

You can [configure certificate rotation](#) parameters in the YAML file.

4.3.5. Deploying the OpenShift Virtualization Operator by using the CLI

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

Prerequisites

- An active subscription to the OpenShift Virtualization catalog in the **openshift-cn** namespace.

Procedure

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

```

apiVersion: hco.kubvirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubvirt-hyperconverged
  namespace: openshift-cn
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-cn** 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.v4.8.7  OpenShift Virtualization  4.8.7
Succeeded
```

4.3.6. Next steps

You might want to additionally configure the following components:

- 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.

4.4. INSTALLING THE VIRTCTL CLIENT

The **virtctl** client is a command-line utility for managing OpenShift Virtualization resources. It is available for Linux, macOS, and Windows distributions.

You can install the **virtctl** client from the OpenShift Virtualization web console or by enabling the OpenShift Virtualization repository and installing the **kubevirt-virtctl** package.


4.4.1. Installing the virtctl client from the web console

You can download the **virtctl** client from the Red Hat Customer Portal, which is linked to in your OpenShift Virtualization web console in the **Command Line Tools** page.

Prerequisites

- You must have an activated OpenShift Container Platform subscription to access the download page on the Customer Portal.

Procedure

1. Access the Customer Portal by clicking the  icon, which is in the upper-right corner of the web console, and selecting **Command Line Tools**.
2. Ensure you have the appropriate version for your cluster selected from the **Version:** list.
3. Download the **virtctl** client for your distribution. All downloads are in **tar.gz** format.
4. Extract the tarball. The following CLI command extracts it into the same directory as the tarball and is applicable for all distributions:

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

5. For Linux and macOS:
 - a. Navigate the extracted folder hierarchy and make the **virtctl** binary executable:


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

b. Move the **virtctl** binary to a directory on your PATH.

i. To check your path, run:

```
$ echo $PATH
```

6. For Windows users:

a. Navigate the extracted folder hierarchy and double-click the **virtctl** executable file to install the client.

4.4.2. Enabling OpenShift Virtualization repositories

Red Hat offers OpenShift Virtualization repositories for both Red Hat Enterprise Linux 8 and Red Hat Enterprise Linux 7:

- Red Hat Enterprise Linux 8 repository: **cnv-4.8-for-rhel-8-x86_64-rpms**
- Red Hat Enterprise Linux 7 repository: **rhel-7-server-cnv-4.8-rpms**

The process for enabling the repository in **subscription-manager** is the same in both platforms.

Procedure

- Enable the appropriate OpenShift Virtualization repository for your system by running the following command:

```
# subscription-manager repos --enable <repository>
```

4.4.3. Installing the virtctl client

Install the **virtctl** client from the **kubevirt-virtctl** package.

Procedure

- Install the **kubevirt-virtctl** package:

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

4.4.4. Additional resources

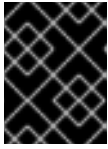
- [Using the CLI tools](#) for OpenShift Virtualization.

4.5. UNINSTALLING OPENSIFT VIRTUALIZATION USING THE WEB CONSOLE

You can uninstall OpenShift Virtualization by using the OpenShift Container Platform [web console](#).

4.5.1. Prerequisites

- You must have OpenShift Virtualization 4.8 installed.
- You must delete all [virtual machines](#), [virtual machine instances](#), and [data volumes](#).



IMPORTANT

Attempting to uninstall OpenShift Virtualization without deleting these objects results in failure.


4.5.2. Deleting the OpenShift Virtualization Operator Deployment custom resource

To uninstall OpenShift Virtualization, you must first delete the **OpenShift Virtualization Operator Deployment** custom resource.

Prerequisites

- Create the **OpenShift Virtualization Operator Deployment** custom resource.

Procedure

1. From the OpenShift Container Platform web console, select **openshift-cnv** from the **Projects** list.
2. Navigate to the **Operators** → **Installed Operators** page.
3. Click **OpenShift Virtualization**.
4. Click the **OpenShift Virtualization Operator Deployment** tab.
5. Click the Options menu  in the row containing the **kubevirt-hyperconverged** custom resource. In the expanded menu, click **Delete HyperConverged Cluster**.
6. Click **Delete** in the confirmation window.
7. Navigate to the **Workloads** → **Pods** page to verify that only the Operator pods are running.
8. Open a terminal window and clean up the remaining resources by running the following command:

```
$ oc delete apiservices v1alpha3.subresources.kubevirt.io -n openshift-cnv
```

4.5.3. Deleting the OpenShift Virtualization catalog subscription

To finish uninstalling OpenShift Virtualization, delete the **OpenShift Virtualization** catalog subscription.

Prerequisites

- An active subscription to the **OpenShift Virtualization** catalog

Procedure

1. Navigate to the **Operators** → **OperatorHub** page.

2. Search for **OpenShift Virtualization** and then select it.
3. Click **Uninstall**.

**NOTE**

You can now delete the **openshift-cnv** namespace.

4.5.4. Deleting a namespace using the web console

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

**NOTE**

If you do not have permissions to delete the namespace, the **Delete Namespace** option is not available.

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



4. When the **Delete Namespace** pane opens, enter the name of the namespace that you want to delete in the field.
5. Click **Delete**.

4.6. UNINSTALLING OPENSIFT VIRTUALIZATION USING THE CLI

You can uninstall OpenShift Virtualization by using the OpenShift Container Platform [CLI](#).

4.6.1. Prerequisites

- You must have OpenShift Virtualization 4.8 installed.
- You must delete all [virtual machines](#), [virtual machine instances](#), and [data volumes](#).

**IMPORTANT**

Attempting to uninstall OpenShift Virtualization without deleting these objects results in failure.

4.6.2. Deleting OpenShift Virtualization

You can delete OpenShift Virtualization by using the CLI.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Access to a OpenShift Virtualization cluster using an account with **cluster-admin** permissions.



NOTE

When you delete the subscription of the OpenShift Virtualization operator in the OLM by using the CLI, the **ClusterServiceVersion** (CSV) object is not deleted from the cluster. To completely uninstall OpenShift Virtualization, you must explicitly delete the CSV.

Procedure

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

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

2. Delete the subscription of the OpenShift Virtualization operator in the Operator Lifecycle Manager (OLM):

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

3. Set the cluster service version (CSV) name for OpenShift Virtualization as an environment variable:

```
$ CSV_NAME=$(oc get csv -n openshift-cnv -o=jsonpath="{.items[0].metadata.name}")
```

4. Delete the CSV from the OpenShift Virtualization cluster by specifying the CSV name from the previous step:

```
$ oc delete csv ${CSV_NAME} -n openshift-cnv
```

OpenShift Virtualization is uninstalled when a confirmation message indicates that the CSV was deleted successfully:

Example output

```
clusterserviceversion.operators.coreos.com "kubevirt-hyperconverged-operator.v4.8.7"  
deleted
```

CHAPTER 5. UPDATING OPENSIFT VIRTUALIZATION

Learn how Operator Lifecycle Manager (OLM) delivers z-stream and minor version updates for OpenShift Virtualization.

5.1. ABOUT UPGRADING OPENSIFT VIRTUALIZATION

5.1.1. How OpenShift Virtualization upgrades 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 upgrade OpenShift Container Platform to the next minor version. You cannot upgrade OpenShift Virtualization to the next minor version without first upgrading OpenShift Container Platform.
- OpenShift Virtualization subscriptions use a single update channel that is named **stable**. The **stable** channel ensures that your OpenShift Virtualization and OpenShift Container Platform versions are compatible.
- If your subscription's approval strategy is set to **Automatic**, the upgrade process starts as soon as a new version of the Operator is available in the **stable** channel. It is highly recommended to use the **Automatic** approval strategy to maintain a supportable environment. Each minor version of OpenShift Virtualization is only supported if you run the corresponding OpenShift Container Platform version. For example, you must run OpenShift Virtualization 4.8 on OpenShift Container Platform 4.8.
 - Though it is possible to select the **Manual** approval strategy, this is not recommended because it risks the supportability and functionality of your cluster. With 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.
- The amount of time an update takes to complete depends on your network connection. Most automatic updates complete within fifteen minutes.

5.1.2. How OpenShift Virtualization upgrades affect your cluster

- Upgrading does not interrupt virtual machine workloads.
 - Virtual machine pods are not restarted or migrated during an upgrade. If you need to update the **virt-launcher** pod, you must restart or live migrate the virtual machine.



NOTE

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

- Upgrading does not interrupt network connections.
- Data volumes and their associated persistent volume claims are preserved during upgrade.



IMPORTANT

If you have virtual machines running that cannot be live migrated, they might block an OpenShift Container Platform cluster upgrade. This includes virtual machines that use hostpath provisioner storage or SR-IOV network interfaces that have the **sriovLiveMigration** feature gate disabled.

As a workaround, you can reconfigure the virtual machines so that they can be powered off automatically during a cluster upgrade. Remove the **evictionStrategy: LiveMigrate** field and set the **runStrategy** field to **Always**.

5.2. 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 **Administrator** perspective of the OpenShift Container Platform web console, navigate to **Operators → 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 update 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 **Operators → Installed Operators** page to monitor the progress of the update. When complete, the status changes to **Succeeded** and **Up to date**.

5.3. MONITORING UPGRADE STATUS

The best way to monitor OpenShift Virtualization upgrade status is to 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
4.8.0	kubevirt-hyperconverged-operator.v2.6.5	Installing
4.8.1	kubevirt-hyperconverged-operator.v4.8.0	Replacing

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

```
$ oc get hco -n openshift-cnv kubevirt-hyperconverged \
-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
```

5.4. ADDITIONAL RESOURCES

- [What are Operators?](#)
- [Operator Lifecycle Manager concepts and resources](#)
- [Cluster service versions \(CSVs\)](#)
- [Configuring virtual machine eviction strategy](#)

CHAPTER 6. ADDITIONAL SECURITY PRIVILEGES GRANTED FOR KUBEVIRT-CONTROLLER AND VIRT-LAUNCHER

The **kubevirt-controller** and virt-launcher pods are granted some SELinux policies and Security Context Constraints privileges that are in addition to typical pod owners. These privileges enable virtual machines to use OpenShift Virtualization features.

6.1. EXTENDED SELINUX POLICIES FOR VIRT-LAUNCHER PODS

The **container_t** SELinux policy for virt-launcher pods is extended with the following rules:

- **allow process self (tun_socket (relabelfrom relabelto attach_queue))**
- **allow process sysfs_t (file (write))**
- **allow process hugetlbfs_t (dir (add_name create write remove_name rmdir setattr))**
- **allow process hugetlbfs_t (file (create unlink))**

These rules enable the following virtualization features:

- Relabel and attach queues to its own TUN sockets, which is required to support network multi-queue. Multi-queue enables network performance to scale as the number of available vCPUs increases.
- Allows virt-launcher pods to write information to sysfs (**/sys**) files, which is required to enable Single Root I/O Virtualization (SR-IOV).
- Read/write **hugetlbfs** entries, which is required to support huge pages. Huge pages are a method of managing large amounts of memory by increasing the memory page size.

6.2. ADDITIONAL OPENSIFT CONTAINER PLATFORM SECURITY CONTEXT CONSTRAINTS AND LINUX CAPABILITIES 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 **kubevirt-controller** is a cluster controller that creates the virt-launcher pods for virtual machines in the cluster. These virt-launcher pods are granted permissions by the **kubevirt-controller** service account.

6.2.1. Additional SCCs granted to the kubevirt-controller service account

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 take advantage of 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{"NET_ADMIN", "NET_RAW", "SYS_NICE"}**
This provides the following additional Linux capabilities **NET_ADMIN**, **NET_RAW**, and **SYS_NICE**.

6.2.2. 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
```

6.3. ADDITIONAL RESOURCES

- The Red Hat Enterprise Linux Virtualization Tuning and Optimization Guide has more information on [network multi-queue](#) and [huge pages](#).
- The **capabilities** man page has more information on the Linux capabilities.
- The **sysfs(5)** man page has more information on sysfs.
- The OpenShift Container Platform Authentication guide has more information on [Security Context Constraints](#).

CHAPTER 7. USING THE CLI TOOLS

The two primary CLI tools used for managing resources in the cluster are:

- The OpenShift Virtualization **virtctl** client
- The OpenShift Container Platform **oc** client

7.1. PREREQUISITES

- You must [install the virtctl client](#).

7.2. VIRTCTL CLIENT COMMANDS

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

To view a list of **virtctl** commands, run the following command:

```
$ virtctl help
```

To view a list of options that you can use with a specific command, run it with the **-h** or **--help** flag. For example:

```
$ virtctl image-upload -h
```

To view a list of global command options that you can use with any **virtctl** command, run the following command:

```
$ virtctl options
```

The following table contains the **virtctl** commands used throughout the OpenShift Virtualization documentation.

Table 7.1. virtctl client commands

Command	Description
virtctl start <vm_name>	Start a virtual machine.
virtctl stop <vm_name>	Stop a virtual machine.
virtctl pause vm vmi <object_name>	Pause a virtual machine or virtual machine instance. The machine state is kept in memory.
virtctl unpause vm vmi <object_name>	Unpause a virtual machine or virtual machine instance.
virtctl migrate <vm_name>	Migrate a virtual machine.
virtctl restart <vm_name>	Restart a virtual machine.

Command	Description
virtctl expose <vm_name>	Create a service that forwards a designated port of a virtual machine or virtual machine instance and expose the service on the specified port of the node.
virtctl console <vmi_name>	Connect to a serial console of a virtual machine instance.
virtctl vnc -- kubeconfig=\$KUBECONFIG <vmi_name>	Open a VNC (Virtual Network Client) connection to a virtual machine instance. Access the graphical console of a virtual machine instance through a VNC which requires a remote viewer on your local machine.
virtctl vnc -- kubeconfig=\$KUBECONFIG --proxy- only=true <vmi-name>	Display the port number and connect manually to the virtual machine instance by using any viewer through the VNC connection.
virtctl vnc -- kubeconfig=\$KUBECONFIG --port= <port-number> <vmi-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.
virtctl image-upload dv <datavolume_name> --image-path= </path/to/image> --no-create	Upload a virtual machine image to a data volume that already exists.
virtctl image-upload dv <datavolume_name> --size= <datavolume_size> --image-path= </path/to/image>	Upload a virtual machine image to a new data volume.
virtctl version	Display the client and server version information.
virtctl help	Display a descriptive list of virtctl commands.
virtctl fslist <vmi_name>	Return a full list of file systems available on the guest machine.
virtctl guestosinfo <vmi_name>	Return guest agent information about the operating system.
virtctl userlist <vmi_name>	Return a full list of logged-in users on the guest machine.

7.3. OPENSIFT CONTAINER PLATFORM CLIENT COMMANDS

The OpenShift Container Platform **oc** client is a command-line utility for managing OpenShift Container Platform resources, including the **VirtualMachine (vm)** and **VirtualMachineInstance (vmi)** object types.

**NOTE**

You can use the **-n <namespace>** flag to specify a different project.

Table 7.2. **oc** commands

Command	Description
oc login -u <user_name>	Log in to the OpenShift Container Platform cluster as <user_name> .
oc get <object_type>	Display a list of objects for the specified object type in the current project.
oc describe <object_type> <resource_name>	Display details of the specific resource in the current project.
oc create -f <object_config>	Create a resource in the current project from a file name or from stdin.
oc edit <object_type> <resource_name>	Edit a resource in the current project.
oc delete <object_type> <resource_name>	Delete a resource in the current project.

For more comprehensive information on **oc** client commands, see the [OpenShift Container Platform CLI tools](#) documentation.

CHAPTER 8. VIRTUAL MACHINES

8.1. CREATING VIRTUAL MACHINES

Use one of these procedures to create a virtual machine:

- Quick Start guided tour
- Running the wizard
- Pasting a pre-configured YAML file with the virtual machine wizard
- Using the CLI
- [Importing a VMware virtual machine or template with the virtual machine wizard](#)



WARNING

Do not create virtual machines in **openshift-*** namespaces. Instead, create a new namespace or use an existing namespace without the **openshift** prefix.

When you create virtual machines from the web console, select a virtual machine template that is configured with a boot source. Virtual machine templates with a boot source are labeled as **Available boot source** or they display a customized label text. Using templates with an available boot source expedites the process of creating virtual machines.

Templates without a boot source are labeled as **Boot source required**. You can use these templates if you complete the steps for [adding a boot source to the virtual machine](#).

8.1.1. Using a Quick Start to create a virtual machine

The web console provides Quick Starts with instructional guided tours for creating virtual machines. You can access the Quick Starts catalog by selecting the Help menu in the **Administrator** perspective to view the Quick Starts catalog. When you click on a Quick Start tile and begin the tour, the system guides you through the process.

Tasks in a Quick Start begin with selecting a Red Hat template. Then, you can add a boot source and import the operating system image. Finally, you can save the custom template and use it to create a virtual machine.

Prerequisites

- Access to the website where you can download the URL link for the operating system image.

Procedure

1. In the web console, select **Quick Starts** from the Help menu.

2. Click on a tile in the Quick Starts catalog. For example: **Creating a Red Hat Linux Enterprise Linux virtual machine**.
3. Follow the instructions in the guided tour and complete the tasks for importing an operating system image and creating a virtual machine. The **Virtual Machines** tab displays the virtual machine.



8.1.2. Running the virtual machine wizard to create a virtual machine

The web console features a wizard that guides you through the process of selecting a virtual machine template and creating a virtual machine. Red Hat virtual machine templates are preconfigured with an operating system image, default settings for the operating system, flavor (CPU and memory), and workload type (server). When templates are configured with a boot source, they are labeled with a customized label text or the default label text **Available boot source**. These templates are then ready to be used for creating virtual machines.

You can select a template from the list of preconfigured templates, review the settings, and create a virtual machine in the **Create virtual machine from template** wizard. If you choose to customize your virtual machine, the wizard guides you through the **General, Networking, Storage, Advanced, and Review** steps. All required fields displayed by the wizard are marked by a *.

Create network interface controllers (NICs) and storage disks later and attach them to virtual machines.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. From the **Virtual Machines** tab or the **Templates** tab, click **Create** and select **Virtual Machine with Wizard**.
3. Select a template that is configured with a boot source.
4. Click **Next** to go to the **Review and create** step.
5. Clear the **Start this virtual machine after creation** checkbox if you do not want to start the virtual machine now.
6. Click **Create virtual machine** and exit the wizard or continue with the wizard to customize the virtual machine.
7. Click **Customize virtual machine** to go to the **General** step.
 - a. Optional: Edit the **Name** field to specify a custom name for the virtual machine.
 - b. Optional: In the **Description** field, add a description.
8. Click **Next** to go to the **Networking** step. A **nic0** NIC is attached by default.
 - a. Optional: Click **Add Network Interface** to create additional NICs.
 - b. Optional: You can remove any or all NICs by clicking the Options menu  and selecting **Delete**. A virtual machine does not need a NIC attached to be created. You can create NICs after the virtual machine has been created.
9. Click **Next** to go to the **Storage** step.

- a. Optional: Click **Add Disk** to create additional disks. These disks can be removed by clicking

the Options menu  and selecting **Delete**.

- b. Optional: Click the Options menu  to edit the disk and save your changes.

10. Click **Next** to go to the **Advanced** step to review the details for **Cloud-init** and configure SSH access.



NOTE

Statically inject an SSH key by using the custom script in cloud-init or in the wizard. This allows you to securely and remotely manage virtual machines and manage and transfer information. This step is strongly recommended to secure your VM.

11. Click **Next** to go to the **Review** step and review the settings for the virtual machine.
12. Click **Create Virtual Machine**
13. Click **See virtual machine details** to view the **Overview** for this virtual machine. The virtual machine is listed in the **Virtual Machines** tab.

Refer to the virtual machine wizard fields section when running the web console wizard.

8.1.2.1. Virtual machine wizard fields

Name	Parameter	Description
Name		The name can contain lowercase letters (a-z), numbers (0-9), and hyphens (-), up to a maximum of 253 characters. The first and last characters must be alphanumeric. The name must not contain uppercase letters, spaces, periods (.), or special characters.
Description		Optional description field.
Operating System		The operating system that is selected for the virtual machine in the template. You cannot edit this field when creating a virtual machine from a template.

Name	Parameter	Description
Boot Source	Import via URL (creates PVC)	Import content from an image available from an HTTP or HTTPS endpoint. Example: Obtaining a URL link from the web page with the operating system image.
	Clone existing PVC (creates PVC)	Select an existent persistent volume claim available on the cluster and clone it.
	Import via Registry (creates PVC)	Provision virtual machine from a bootable operating system container located in a registry accessible from the cluster. Example: kubevirt/cirros-registry-disk-demo .
	PXE (network boot - adds network interface)	Boot an operating system from a server on the network. Requires a PXE bootable network attachment definition.
Persistent Volume Claim project		Project name that you want to use for cloning the PVC.
Persistent Volume Claim name		PVC name that should apply to this virtual machine template if you are cloning an existing PVC.
Mount this as a CD-ROM boot source		A CD-ROM requires an additional disk for installing the operating system. Select the checkbox to add a disk and customize it later.

Name	Description
Model	Indicates the model of the network interface controller. Supported values are e1000e and virtio .
Network	List of available network attachment definitions.
Type	List of available binding methods. For the default pod network, masquerade is the only recommended binding method. For secondary networks, use the bridge binding method. The masquerade method is not supported for non-default networks. Select SR-IOV if you configured an SR-IOV network device and defined that network in the namespace.
MAC Address	MAC address for the network interface controller. If a MAC address is not specified, one is assigned automatically.

8.1.2.3. Storage fields


Name	Selection	Description
Source	Blank (creates PVC)	Create an empty disk.
	Import via URL (creates PVC)	Import content via URL (HTTP or HTTPS endpoint).
	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	Selection	Description
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 → Volume Mode		Defines whether the persistent volume uses a formatted file system or raw block state. Default is Filesystem .
Advanced → Access Mode		Access mode of the persistent volume. Supported access modes are Single User (RWO) , Shared Access (RWX) , and Read Only (ROX) .

Advanced storage settings

The following advanced storage settings are available for **Blank**, **Import via URL**, and **Clone existing PVC** disks. These parameters are optional. If you do not specify these parameters, the system uses the default values from the **kubevirt-storage-class-defaults** config map.

Name	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	Single User (RWO)	The disk can be mounted as read/write by a single node.

Name	Parameter	Description
	Shared Access (RWX)	<p>The disk can be mounted as read/write by many nodes.</p>  <p>NOTE</p> <p>This is required for some features, such as live migration of virtual machines between nodes.</p>
	Read Only (ROX)	The disk can be mounted as read-only by many nodes.

8.1.2.4. Cloud-init fields

Name	Description
Hostname	Sets a specific hostname for the virtual machine.
Authorized SSH Keys	The user's public key that is copied to <code>~/.ssh/authorized_keys</code> on the virtual machine.
Custom script	Replaces other options with a field in which you paste a custom cloud-init script.

To configure storage class defaults, use storage profiles. For more information, see [Customizing the storage profile](#).

8.1.2.5. Pasting in a pre-configured YAML file to create a virtual machine

Create a virtual machine by writing or pasting a YAML configuration file. A valid **example** virtual machine configuration is provided by default whenever you open the YAML edit screen.

If your YAML configuration is invalid when you click **Create**, an error message indicates the parameter in which the error occurs. Only one error is shown at a time.



NOTE

Navigating away from the YAML screen while editing cancels any changes to the configuration you have made.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Click **Create** and select **Virtual Machine With YAML**

4. Write or paste your virtual machine configuration in the editable window.
 - a. Alternatively, use the **example** virtual machine provided by default in the YAML screen.
5. Optional: Click **Download** to download the YAML configuration file in its present state.
6. Click **Create** to create the virtual machine.

The virtual machine is listed in the **Virtual Machines** tab.

8.1.3. Using the CLI to create a virtual machine

You can create a virtual machine from a **virtualMachine** manifest.

Procedure

1. Edit the **VirtualMachine** manifest for your VM. For example, the following manifest configures a Red Hat Enterprise Linux (RHEL) VM:

Example 8.1. Example manifest for a RHEL VM

```

apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    app: <vm_name> 1
  name: <vm_name>
spec:
  dataVolumeTemplates:
  - apiVersion: cdi.kubevirt.io/v1beta1
    kind: DataVolume
    metadata:
      name: <vm_name>
    spec:
      sourceRef:
        kind: DataSource
        name: rhel9
        namespace: openshift-virtualization-os-images
      storage:
        resources:
          requests:
            storage: 30Gi
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/domain: <vm_name>
    spec:
      domain:
        cpu:
          cores: 1
          sockets: 2
          threads: 1
      devices:
        disks:
          - disk:

```

```

    bus: virtio
    name: rootdisk
  - disk:
    bus: virtio
    name: cloudinitdisk
  interfaces:
  - masquerade: {}
    name: default
  rng: {}
  features:
  smm:
    enabled: true
  firmware:
  bootloader:
    efi: {}
  resources:
  requests:
    memory: 8Gi
  evictionStrategy: LiveMigrate
  networks:
  - name: default
    pod: {}
  volumes:
  - dataVolume:
    name: <vm_name>
    name: rootdisk
  - cloudInitNoCloud:
    userData: |-
      #cloud-config
      user: cloud-user
      password: '<password>' 2
      chpasswd: { expire: False }
    name: cloudinitdisk

```

- 1 Specify the name of the virtual machine.
- 2 Specify the password for cloud-user.

2. Create a virtual machine by using the manifest file:

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


3. Optional: Start the virtual machine:

```
$ virtctl start <vm_name>
```

8.1.4. Virtual machine storage volume types

Storage volume type	Description
---------------------	-------------

Storage volume type	Description
ephemeral	<p>A local copy-on-write (COW) image that uses a network volume as a read-only backing store. The backing volume must be a PersistentVolumeClaim. The ephemeral image is created when the virtual machine starts and stores all writes locally. The ephemeral image is discarded when the virtual machine is stopped, restarted, or deleted. The backing volume (PVC) is not mutated in any way.</p>
persistentVolumeClaim	<p>Attaches an available PV to a virtual machine. Attaching a PV allows for the virtual machine data to persist between sessions.</p> <p>Importing an existing virtual machine disk into a PVC by using CDI and attaching the PVC to a virtual machine instance is the recommended method for importing existing virtual machines into OpenShift Container Platform. There are some requirements for the disk to be used within a PVC.</p>
dataVolume	<p>Data volumes build on the persistentVolumeClaim disk type by managing the process of preparing the virtual machine disk via an import, clone, or upload operation. VMs that use this volume type are guaranteed not to start until the volume is ready.</p> <p>Specify type: dataVolume or type: "". If you specify any other value for type, such as persistentVolumeClaim, a warning is displayed, and the virtual machine does not start.</p>
cloudInitNoCloud	<p>Attaches a disk that contains the referenced cloud-init NoCloud data source, providing user data and metadata to the virtual machine. A cloud-init installation is required inside the virtual machine disk.</p>

Storage volume type	Description
containerDisk	<p>References an image, such as a virtual machine disk, that is stored in the container image registry. The image is pulled from the registry and attached to the virtual machine as a disk when the virtual machine is launched.</p> <p>A containerDisk volume is not limited to a single virtual machine and is useful for creating large numbers of virtual machine clones that do not require persistent storage.</p> <p>Only RAW and QCOW2 formats are supported disk types for the container image registry. QCOW2 is recommended for reduced image size.</p> <div data-bbox="815 752 922 1043" style="display: inline-block; vertical-align: top;">  </div> <p>NOTE</p> <p>A containerDisk volume is ephemeral. It is discarded when the virtual machine is stopped, restarted, or deleted. A containerDisk volume is useful for read-only file systems such as CD-ROMs or for disposable virtual machines.</p>
emptyDisk	<p>Creates an additional sparse QCOW2 disk that is tied to the life-cycle of the virtual machine interface. The data survives guest-initiated reboots in the virtual machine but is discarded when the virtual machine stops or is restarted from the web console. The empty disk is used to store application dependencies and data that otherwise exceeds the limited temporary file system of an ephemeral disk.</p> <p>The disk capacity size must also be provided.</p>

8.1.5. About RunStrategies for virtual machines

A **RunStrategy** for virtual machines determines a virtual machine instance's (VMI) behavior, depending on a series of conditions. The **spec.runStrategy** setting exists in the virtual machine configuration process as an alternative to the **spec.running** setting. The **spec.runStrategy** setting allows greater flexibility for how VMIs are created and managed, in contrast to the **spec.running** setting with only **true** or **false** responses. However, the two settings are mutually exclusive. Only either **spec.running** or **spec.runStrategy** can be used. An error occurs if both are used.

There are four defined RunStrategies.

Always

A VMI is always present when a virtual machine is created. A new VMI is created if the original stops for any reason, which is the same behavior as **spec.running: true**.

RerunOnFailure

A VMI is re-created if the previous instance fails due to an error. The instance is not re-created if the virtual machine stops successfully, such as when it shuts down.

Manual

The **start**, **stop**, and **restart** virtctl client commands can be used to control the VMI's state and existence.

Halted

No VMI is present when a virtual machine is created, which is the same behavior as **spec.running: false**.

Different combinations of the **start**, **stop** and **restart** virtctl commands affect which **RunStrategy** is used.

The following table follows a VM's transition from different states. The first column shows the VM's initial **RunStrategy**. Each additional column shows a virtctl command and the new **RunStrategy** after that command is run.

Initial RunStrategy	start	stop	restart
Always	-	Halted	Always
RerunOnFailure	-	Halted	RerunOnFailure
Manual	Manual	Manual	Manual
Halted	Always	-	-



NOTE

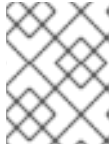
In OpenShift Virtualization clusters installed using installer-provisioned infrastructure, when a node fails the MachineHealthCheck and becomes unavailable to the cluster, VMs with a RunStrategy of **Always** or **RerunOnFailure** are rescheduled on a new node.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  RunStrategy: Always 1
  template:
  ...
```

1 The VMI's current **RunStrategy** setting.

8.1.6. Additional resources

- The **VirtualMachineSpec** definition in the [KubeVirt v0.41.0 API Reference](#) provides broader context for the parameters and hierarchy of the virtual machine specification.



NOTE

The KubeVirt API Reference is the upstream project reference and might contain parameters that are not supported in OpenShift Virtualization.

- [Prepare a container disk](#) before adding it to a virtual machine as a **containerDisk** volume.
- See [Deploying machine health checks](#) for further details on deploying and enabling machine health checks.
- See [Installer-provisioned infrastructure overview](#) for further details on installer-provisioned infrastructure.
- See [Configuring the SR-IOV Network Operator](#) for further details on the SR-IOV Network Operator.

8.2. EDITING VIRTUAL MACHINES

You can update a virtual machine configuration using either the YAML editor in the web console or the OpenShift CLI on the command line. You can also update a subset of the parameters in the **Virtual Machine Details** screen.

8.2.1. Editing a virtual machine in the web console

Edit select values of a virtual machine in the web console by clicking the pencil icon next to the relevant field. Other values can be edited using the CLI.

Labels and annotations are editable for both preconfigured Red Hat templates and your custom virtual machine templates. All other values are editable only for custom virtual machine templates that users have created using the Red Hat templates or the **Create Virtual Machine Template**wizard.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine.
4. Click the **Details** tab.
5. Click the pencil icon to make a field editable.
6. Make the relevant changes and click **Save**.

**NOTE**

If the virtual machine is running, changes to **Boot Order** or **Flavor** 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 relevant 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.

8.2.2. Editing a virtual machine YAML configuration using the web console

You can edit the YAML configuration of a virtual machine in the web console. Some parameters cannot be modified. If you click **Save** with an invalid configuration, an error message indicates the parameter that cannot be changed.

If you edit the YAML configuration while the virtual machine is running, changes will not take effect until you restart the virtual machine.

**NOTE**

Navigating away from the YAML screen while editing cancels any changes to the configuration you have made.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Select a virtual machine.
3. Click the **YAML** tab to display the editable configuration.
4. Optional: You can click **Download** to download the YAML file locally in its current state.
5. Edit the file and click **Save**.

A confirmation message shows that the modification has been successful and includes the updated version number for the object.

8.2.3. Editing a virtual machine YAML configuration using the CLI

Use this procedure to edit a virtual machine YAML configuration using the CLI.

Prerequisites

- You configured a virtual machine with a YAML object configuration file.
- You installed the **oc** CLI.

Procedure

1. Run the following command to update the virtual machine configuration:

```
$ oc edit <object_type> <object_ID>
```

2. Open the object configuration.

3. Edit the YAML.
4. 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 <object_type> <object_ID>
```

8.2.4. Adding a virtual disk to a virtual machine

Use this procedure to add a virtual disk to a virtual machine.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Disks** tab.
5. In the **Add Disk** window, specify the **Source**, **Name**, **Size**, **Type**, **Interface**, and **Storage Class**.
 - a. Optional: In the **Advanced** list, specify 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.
6. Click **Add**.



NOTE

If the virtual machine is running, the new disk is in the **pending restart** state and will not be attached until you restart the virtual machine.

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.

To configure storage class defaults, use storage profiles. For more information, see [Customizing the storage profile](#).


8.2.4.1. Storage fields

Name	Selection	Description
Source	Blank (creates PVC)	Create an empty disk.
	Import via URL (creates PVC)	Import content via URL (HTTP or HTTPS endpoint).

Name	Selection	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 → Volume Mode		Defines whether the persistent volume uses a formatted file system or raw block state. Default is Filesystem .
Advanced → Access Mode		Access mode of the persistent volume. Supported access modes are Single User (RWO) , Shared Access (RWX) , and Read Only (ROX) .

Advanced storage settings

The following advanced storage settings are available for **Blank**, **Import via URL**, and **Clone existing PVC** disks. These parameters are optional. If you do not specify these parameters, the system uses the default values from the **kubevirt-storage-class-defaults** config map.

Name	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	Single User (RWO)	The disk can be mounted as read/write by a single node.
	Shared Access (RWX)	The disk can be mounted as read/write by many nodes. <div style="display: flex; align-items: center;">  <div> <p>NOTE</p> <p>This is required for some features, such as live migration of virtual machines between nodes.</p> </div> </div>
	Read Only (ROX)	The disk can be mounted as read-only by many nodes.

8.2.5. Adding a network interface to a virtual machine

Use this procedure to add a network interface to a virtual machine.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Network Interfaces** tab.
5. Click **Add Network Interface**.
6. In the **Add Network Interface** window, specify the **Name**, **Model**, **Network**, **Type**, and **MAC Address** of the network interface.
7. Click **Add**.

**NOTE**

If the virtual machine is running, the new network interface is in the **pending restart** state and changes will not take effect until you restart the virtual machine.

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.


8.2.5.1. 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 .
Network	List of available network attachment definitions.
Type	List of available binding methods. For the default pod network, masquerade is the only recommended binding method. For secondary networks, use the bridge binding method. The masquerade method is not supported for non-default networks. Select SR-IOV if you configured an SR-IOV network device and defined that network in the namespace.
MAC Address	MAC address for the network interface controller. If a MAC address is not specified, one is assigned automatically.

8.2.6. Editing CD-ROMs for Virtual Machines

Use the following procedure to edit CD-ROMs for virtual machines.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Disks** tab.
5. Click the Options menu  for the CD-ROM that you want to edit and select **Edit**.
6. In the **Edit CD-ROM** window, edit the fields: **Source**, **Persistent Volume Claim**, **Name**, **Type**, and **Interface**.

7. Click **Save**.

8.3. 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.

8.3.1. 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 **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Details** tab.
5. 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.**
6. Click **Add Source** and select a bootable disk or network interface controller (NIC) for the virtual machine.
7. Add any additional disks or NICs to the boot order list.
8. 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.

8.3.2. Editing a boot order list in the web console

Edit the boot order list in the web console.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Details** tab.
5. Click the pencil icon that is located on the right side of **Boot Order**.
6. 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.
7. 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.

8.3.3. Editing a boot order list in the YAML configuration file

Edit the boot order list in a YAML configuration file by using the CLI.

Procedure

1. Open the YAML configuration file for the virtual machine by running the following command:

```
$ oc edit vm example
```

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.
 4. Click **reload the content** to apply the updated boot order values from the YAML file to the boot order list in the web console.

8.3.4. 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 **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Details** tab.
5. Click the pencil icon that is located on the right side of **Boot Order**.
6. 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.

8.4. DELETING VIRTUAL MACHINES

You can delete a virtual machine from the web console or by using the **oc** command line interface.


8.4.1. Deleting a virtual machine using the web console

Deleting a virtual machine permanently removes it from the cluster.

**NOTE**

When you delete a virtual machine, the data volume it uses is automatically deleted.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Click the Options menu  of the virtual machine that you want to delete and select **Delete Virtual Machine**.
 - Alternatively, click the virtual machine name to open the **Virtual Machine Overview** screen and click **Actions** → **Delete Virtual Machine**
4. In the confirmation pop-up window, click **Delete** to permanently delete the virtual machine.

8.4.2. Deleting a virtual machine by using the CLI

You can delete a virtual machine by using the **oc** command line interface (CLI). The **oc** client enables you to perform actions on multiple virtual machines.

**NOTE**

When you delete a virtual machine, the data volume it uses is automatically deleted.

Prerequisites

- Identify the name of the virtual machine that you want to delete.

Procedure

- Delete the virtual machine by running the following command:

```
$ oc delete vm <vm_name>
```

**NOTE**

This command only deletes objects that exist in the current project. Specify the **-n <project_name>** option if the object you want to delete is in a different project or namespace.

8.5. 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 the command-line interface (CLI).

8.5.1. 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.

8.5.2. 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).

Procedure

- List all VMIs by running the following command:

```
$ oc get vmis
```

8.5.3. 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

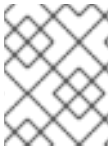
- Click **Workloads** → **Virtualization** from the side menu. A list of VMs and standalone VMIs displays. You can identify standalone VMIs by the dark colored badges that display next to the virtual machine instance names.

8.5.4. Editing a standalone virtual machine instance using the web console

You can edit annotations and labels for a standalone virtual machine instance (VMI) using the web console. Other items displayed in the **Details** page for a standalone VMI are not editable.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu. A list of virtual machines (VMs) and standalone VMIs displays.
2. Click the name of a standalone VMI to open the **Virtual Machine Instance Overview** screen.
3. Click the **Details** tab.
4. Click the pencil icon that is located on the right side of **Annotations**.
5. Make the relevant changes and click **Save**.



NOTE

To edit labels for a standalone VMI, click **Actions** and select **Edit Labels**. Make the relevant changes and click **Save**.

8.5.5. 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.

Procedure

- Delete the VMI by running the following command:

```
$ oc delete vmi <vmi_name>
```

8.5.6. 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 **Workloads** → **Virtualization** from the side menu.
2. Click the **:** button of the standalone virtual machine instance (VMI) that you want to delete and select **Delete Virtual Machine Instance**
 - Alternatively, click the name of the standalone VMI. The **Virtual Machine Instance Overview** page displays.
3. Select **Actions** → **Delete Virtual Machine Instance**
4. In the confirmation pop-up window, click **Delete** to permanently delete the standalone VMI.

8.6. CONTROLLING VIRTUAL MACHINE STATES

You can stop, start, restart, and unpause virtual machines from the web console.




NOTE

To control virtual machines from the command-line interface (CLI), use the [virtctl](#) client.

8.6.1. Starting a virtual machine

You can start a virtual machine from the web console.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Find the row that contains the virtual machine that you want to start.
4. Navigate to the appropriate menu for your use case:
 - To stay on this page, where you can perform actions on multiple virtual machines:
 - a. Click the Options menu  located at the far right end of the row.
 - To view comprehensive information about the selected virtual machine before you start it:
 - a. Access the **Virtual Machine Overview** screen by clicking the name of the virtual machine.
 - b. Click **Actions**.
5. Select **Start Virtual Machine**
6. In the confirmation window, click **Start** to start the virtual machine.

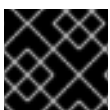


NOTE

When you start virtual machine that is provisioned from a **URL** source for the first time, the virtual machine 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.

8.6.2. Restarting a virtual machine


You can restart a running virtual machine from the web console.



IMPORTANT

To avoid errors, do not restart a virtual machine while it has a status of **Importing**.


Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Find the row that contains the virtual machine that you want to restart.
4. Navigate to the appropriate menu for your use case:
 - To stay on this page, where you can perform actions on multiple virtual machines:
 - a. Click the Options menu  located at the far right end of the row.
 - To view comprehensive information about the selected virtual machine before you restart it:
 - a. Access the **Virtual Machine Overview** screen by clicking the name of the virtual machine.
 - b. Click **Actions**.
5. Select **Restart Virtual Machine**
6. In the confirmation window, click **Restart** to restart the virtual machine.

8.6.3. Stopping a virtual machine

You can stop a virtual machine from the web console.

Procedure

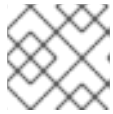
1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Find the row that contains the virtual machine that you want to stop.
4. Navigate to the appropriate menu for your use case:
 - To stay on this page, where you can perform actions on multiple virtual machines:
 - a. Click the Options menu  located at the far right end of the row.
 - To view comprehensive information about the selected virtual machine before you stop it:
 - a. Access the **Virtual Machine Overview** screen by clicking the name of the virtual machine.
 - b. Click **Actions**.
5. Select **Stop Virtual Machine**
6. In the confirmation window, click **Stop** to stop the virtual machine.

8.6.4. Unpausing a virtual machine

You can unpause a paused virtual machine from the web console.

Prerequisites

- At least one of your virtual machines must have a status of **Paused**.



NOTE

You can pause virtual machines by using the **virtctl** client.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Find the row that contains the virtual machine that you want to unpause.
4. Navigate to the appropriate menu for your use case:
 - To stay on this page, where you can perform actions on multiple virtual machines:
 - a. In the **Status** column, click **Paused**.
 - To view comprehensive information about the selected virtual machine before you unpause it:
 - a. Access the **Virtual Machine Overview** screen by clicking the name of the virtual machine.
 - b. Click the pencil icon that is located on the right side of **Status**.
5. In the confirmation window, click **Unpause** to unpause the virtual machine.

8.7. ACCESSING VIRTUAL MACHINE CONSOLES

OpenShift Virtualization provides different virtual machine consoles that you can use to accomplish different product tasks. You can access these consoles through the OpenShift Container Platform web console and by using CLI commands.

8.7.1. Accessing virtual machine consoles in the OpenShift Container Platform web console

You can connect to virtual machines by using the serial console or the VNC console in the OpenShift Container Platform web console.

You can connect to Windows virtual machines by using the desktop viewer console, which uses RDP (remote desktop protocol), in the OpenShift Container Platform web console.

8.7.1.1. Connecting to the serial console

Connect to the serial console of a running virtual machine from the **Console** tab in the **Virtual Machine Overview** screen of the web console.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** page.
4. Click **Console**. The VNC console opens by default.
5. Select **Disconnect before switching** to ensure that only one console session is open at a time. Otherwise, the VNC console session remains active in the background.
6. Click the **VNC Console** drop-down list and select **Serial Console**.
7. Click **Disconnect** to end the console session.
8. Optional: Open the serial console in a separate window by clicking **Open Console in New Window**.

8.7.1.2. Connecting to the VNC console

Connect to the VNC console of a running virtual machine from the **Console** tab in the **Virtual Machine Overview** screen of the web console.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** page.
4. Click the **Console** tab. The VNC console opens by default.
5. Optional: Open the VNC console in a separate window by clicking **Open Console in New Window**.
6. Optional: Send key combinations to the virtual machine by clicking **Send Key**.
7. Click outside the console window and then click **Disconnect** to end the session.

8.7.1.3. Connecting to a Windows virtual machine with RDP

The desktop viewer console, which utilizes the Remote Desktop Protocol (RDP), provides a better console experience for connecting to Windows virtual machines.

To connect to a Windows virtual machine with RDP, download the **console.rdp** file for the virtual machine from the **Consoles** tab in the **Virtual Machine Details** screen of the web console and supply it to your preferred RDP client.

Prerequisites

- A running Windows virtual machine with the QEMU guest agent installed. The **qemu-guest-agent** is included in the VirtIO drivers.
- A layer-2 NIC attached to the virtual machine.

- An RDP client installed on a machine on the same network as the Windows virtual machine.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a Windows virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Console** tab.
5. In the **Console** list, select **Desktop Viewer**.
6. In the **Network Interface** list, select the layer-2 NIC.
7. Click **Launch Remote Desktop** to download the **console.rdp** file.
8. Open an RDP client and reference the **console.rdp** file. For example, using **remmina**:

```
$ remmina --connect /path/to/console.rdp
```

9. Enter the **Administrator** user name and password to connect to the Windows virtual machine.

8.7.1.4. Copying the SSH command from the web console

Copy the command to access a running virtual machine (VM) via SSH from the **Actions** list in the web console.

Procedure

1. In the OpenShift Container Platform console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** page.
4. From the **Actions** list, select **Copy SSH Command**. You can now paste this command onto the OpenShift CLI (**oc**).

8.7.2. Accessing virtual machine consoles by using CLI commands

8.7.2.1. Accessing a virtual machine instance via SSH

You can use SSH to access a virtual machine (VM) after you expose port 22 on it.

The **virtctl expose** command forwards a virtual machine instance (VMI) port to a node port and creates a service for enabled access. The following example creates the **fedora-vm-ssh** service that forwards traffic from a specific port of cluster nodes to port 22 of the **<fedora-vm>** virtual machine.

Prerequisites

- You must be in the same project as the VMI.

- The VMI you want to access must be connected to the default pod network by using the **masquerade** binding method.
- The VMI you want to access must be running.
- Install the OpenShift CLI (**oc**).

Procedure

1. Run the following command to create the **fedora-vm-ssh** service:

```
$ virtctl expose vm <fedora-vm> --port=22 --name=fedora-vm-ssh --type=NodePort 1
```

1 **<fedora-vm>** is the name of the VM that you run the **fedora-vm-ssh** service on.

2. Check the service to find out which port the service acquired:

```
$ oc get svc
```

Example output

```
NAME          TYPE      CLUSTER-IP  EXTERNAL-IP  PORT(S)        AGE
fedora-vm-ssh NodePort  127.0.0.1   <none>       22:32551/TCP  6s
```

+ In this example, the service acquired the **32551** port.

1. Log in to the VMI via SSH. Use the **ipAddress** of any of the cluster nodes and the port that you found in the previous step:

```
$ ssh username@<node_IP_address> -p 32551
```

8.7.2.2. Accessing a virtual machine via SSH with YAML configurations

You can enable an SSH connection to a virtual machine (VM) without the need to run the **virtctl expose** command. When the YAML file for the VM and the YAML file for the service are configured and applied, the service forwards the SSH traffic to the VM.

The following examples show the configurations for the VM's YAML file and the service YAML file.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Create a namespace for the VM's YAML file by using the **oc create namespace** command and specifying a name for the namespace.

Procedure

1. In the YAML file for the VM, add the label and a value for exposing the service for SSH connections. Enable the **masquerade** feature for the interface:

Example VirtualMachine definition

■

```

...
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  namespace: ssh-ns 1
  name: vm-ssh
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm-ssh
        special: vm-ssh 2
    spec:
      domain:
        devices:
          disks:
            - disk:
                bus: virtio
                name: containerdisk
            - disk:
                bus: virtio
                name: cloudinitdisk
          interfaces:
            - masquerade: {} 3
              name: testmasquerade 4
            rng: {}
          machine:
            type: ""
          resources:
            requests:
              memory: 1024M
          networks:
            - name: testmasquerade
              pod: {}
          volumes:
            - name: containerdisk
              containerDisk:
                image: kubevirt/fedora-cloud-container-disk-demo
            - name: cloudinitdisk
              cloudInitNoCloud:
                userData: |
                  #!/bin/bash
                  echo "fedora" | passwd fedora --stdin
...

```

- 1** Name of the namespace created by the **oc create namespace** command.
- 2** Label used by the service to identify the virtual machine instances that are enabled for SSH traffic connections. The label can be any **key:value** pair that is added as a **label** to this YAML file and as a **selector** in the service YAML file.
- 3** The interface type is **masquerade**.
- 4** The name of this interface is **testmasquerade**.

2. Create the VM:

```
$ oc create -f <path_for_the_VM_YAML_file>
```

3. Start the VM:

```
$ virtctl start vm-ssh
```

4. In the YAML file for the service, specify the service name, port number, and the target port.

Example Service definition

```
...
apiVersion: v1
kind: Service
metadata:
  name: svc-ssh 1
  namespace: ssh-ns 2
spec:
  ports:
    - targetPort: 22 3
      protocol: TCP
      port: 27017
  selector:
    special: vm-ssh 4
  type: NodePort
...
```

1 Name of the SSH service.

2 Name of the namespace created by the **oc create namespace** command.

3 The target port number for the SSH connection.

4 The selector name and value must match the label specified in the YAML file for the VM.

5. Create the service:

```
$ oc create -f <path_for_the_service_YAML_file>
```

6. Verify that the VM is running:

```
$ oc get vmi
```

Example output

```
NAME AGE PHASE IP NODENAME
vm-ssh 6s Running 10.244.196.152 node01
```

7. Check the service to find out which port the service acquired:

```
$ oc get svc
```

-

Example output

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
svc-ssh	NodePort	10.106.236.208	<none>	27017:30093/TCP	22s

In this example, the service acquired the port number 30093.

- Run the following command to obtain the IP address for the node:

```
$ oc get node <node_name> -o wide
```

Example output

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP
node01	Ready	worker	6d22h	v1.20.0+5f82cdb	192.168.55.101	<none>

- Log in to the VM via SSH by specifying the IP address of the node where the VM is running and the port number. Use the port number displayed by the **oc get svc** command and the IP address of the node displayed by the **oc get node** command. The following example shows the **ssh** command with the username, node's IP address, and the port number:

```
$ ssh fedora@192.168.55.101 -p 30093
```

8.7.2.3. Accessing the serial console of a virtual machine instance

The **virtctl console** command opens a serial console to the specified virtual machine instance.

Prerequisites

- The **virt-viewer** package must be installed.
- The virtual machine instance you want to access must be running.

Procedure

- Connect to the serial console with **virtctl**:

```
$ virtctl console <VMI>
```

8.7.2.4. Accessing the graphical console of a virtual machine instances with VNC

The **virtctl** client utility can use the **remote-viewer** function to open a graphical console to a running virtual machine instance. This capability is included in the **virt-viewer** package.

Prerequisites

- The **virt-viewer** package must be installed.
- The virtual machine instance you want to access must be running.

**NOTE**

If you use **virtctl** via SSH on a remote machine, you must forward the X session to your machine.

Procedure

1. Connect to the graphical interface with the **virtctl** utility:

```
$ virtctl vnc <VMI>
```

2. If the command failed, try using the **-v** flag to collect troubleshooting information:

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

8.7.2.5. Connecting to a Windows virtual machine with an RDP console

The Remote Desktop Protocol (RDP) provides a better console experience for connecting to Windows virtual machines.

To connect to a Windows virtual machine with RDP, specify the IP address of the attached L2 NIC to your RDP client.

Prerequisites

- A running Windows virtual machine with the QEMU guest agent installed. The **qemu-guest-agent** is included in the VirtIO drivers.
- A layer 2 NIC attached to the virtual machine.
- An RDP client installed on a machine on the same network as the Windows virtual machine.

Procedure

1. Log in to the OpenShift Virtualization cluster through the **oc** CLI tool as a user with an access token.

```
$ oc login -u <user> https://<cluster.example.com>:8443
```

2. Use **oc describe vmi** to display the configuration of the running Windows virtual machine.

```
$ oc describe vmi <windows-vmi-name>
```

Example output

```
...
spec:
  networks:
  - name: default
    pod: {}
  - multus:
    networkName: cnv-bridge
    name: bridge-net
```

```

...
status:
  interfaces:
  - interfaceName: eth0
    ipAddress: 198.51.100.0/24
    ipAddresses:
      198.51.100.0/24
    mac: a0:36:9f:0f:b1:70
    name: default
  - interfaceName: eth1
    ipAddress: 192.0.2.0/24
    ipAddresses:
      192.0.2.0/24
      2001:db8::/32
    mac: 00:17:a4:77:77:25
    name: bridge-net
...

```

3. Identify and copy the IP address of the layer 2 network interface. This is **192.0.2.0** in the above example, or **2001:db8::** if you prefer IPv6.
4. Open an RDP client and use the IP address copied in the previous step for the connection.
5. Enter the **Administrator** user name and password to connect to the Windows virtual machine.

8.8. TRIGGERING VIRTUAL MACHINE FAILOVER BY RESOLVING A FAILED NODE

If a node fails and [machine health checks](#) are not deployed on your cluster, virtual machines (VMs) with **RunStrategy: Always** configured are not automatically relocated to healthy nodes. To trigger VM failover, you must manually delete the **Node** object.



NOTE

If you installed your cluster by using [installer-provisioned infrastructure](#) and you properly configured machine health checks:

- Failed nodes are automatically recycled.
- Virtual machines with **RunStrategy** set to **Always** or **RerunOnFailure** are automatically scheduled on healthy nodes.

8.8.1. 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**).

8.8.2. 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.

8.8.3. 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.

8.8.3.1. 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).

Procedure

- List all VMIs by running the following command:

```
$ oc get vmis
```

8.9. INSTALLING THE QEMU GUEST AGENT ON VIRTUAL MACHINES

The [QEMU guest agent](#) is a daemon that runs on the virtual machine and passes information to the host about the virtual machine, users, file systems, and secondary networks.

8.9.1. Installing QEMU guest agent on a Linux virtual machine

The **qemu-guest-agent** is widely available and available by default in Red Hat virtual machines. Install the agent and start the service

Procedure

1. Access the virtual machine command line through one of the consoles or by SSH.
2. Install the QEMU guest agent on the virtual machine:

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

3. Ensure the service is persistent and start it:

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

You can also install and start the QEMU guest agent by using the **custom script** field in the **cloud-init** section of the wizard when creating either virtual machines or virtual machines templates in the web console.

8.9.2. Installing QEMU guest agent on a Windows virtual machine

For Windows virtual machines, the QEMU guest agent is included in the VirtIO drivers, which can be installed using one of the following procedures:

8.9.2.1. Installing VirtIO drivers on an existing Windows virtual machine

Install the VirtIO drivers from the attached SATA CD drive to an existing Windows virtual machine.



NOTE

This procedure uses a generic approach to adding drivers to Windows. The process might differ slightly between versions of Windows. See the installation documentation for your version of Windows for specific installation steps.

Procedure

1. Start the virtual machine 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. Right-click the device and select **Properties**.
 - b. Click the **Details** tab and select **Hardware Ids** in the **Property** list.
 - c. 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 virtual machine to complete the driver installation.

8.9.2.2. Installing VirtIO drivers during Windows installation

Install the VirtIO drivers from the attached SATA CD driver during Windows installation.



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.

Procedure

1. Start the virtual machine and connect to a graphical console.
2. Begin the Windows installation process.
3. Select the **Advanced** installation.
4. The storage destination will not be recognized until the driver is loaded. Click **Load driver**.
5. The drivers are attached as a SATA CD drive. Click **OK** and browse the CD drive for the storage driver to load. The drivers are arranged hierarchically according to their driver type, operating system, and CPU architecture.
6. Repeat the previous two steps for all required drivers.
7. Complete the Windows installation.

8.10. VIEWING THE QEMU GUEST AGENT INFORMATION FOR VIRTUAL MACHINES

When the QEMU guest agent runs on the virtual machine, you can use the web console to view information about the virtual machine, users, file systems, and secondary networks.

8.10.1. Prerequisites

- Install the [QEMU guest agent](#) on the virtual machine.

8.10.2. About the QEMU guest agent information in the web console

When the QEMU guest agent is installed, the **Details** pane within the **Virtual Machine Overview** tab and the **Details** tab display information about the hostname, operating system, time zone, and logged in users.

The **Virtual Machine Overview** shows information about the guest operating system installed on the virtual machine. The **Details** tab displays a table with information for logged in users. The **Disks** tab displays a table with information for file systems.



NOTE

If the QEMU guest agent is not installed, the **Virtual Machine Overview** tab and the **Details** tab display information about the operating system that was specified when the virtual machine was created.

8.10.3. Viewing the QEMU guest agent information in the web console

You can use the web console to view information for virtual machines that is passed by the QEMU guest agent to the host.

Procedure

1. Click **Workloads** → **Virtual Machines** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine name to open the **Virtual Machine Overview** screen and view the **Details** pane.
4. Click **Logged in users** to view the **Details** tab that shows information for users.
5. Click the **Disks** tab to view information about the file systems.

8.11. MANAGING CONFIG MAPS, SECRETS, AND SERVICE ACCOUNTS IN VIRTUAL MACHINES

You can use secrets, config maps, and service accounts to pass configuration data to virtual machines. For example, you can:

- Give a virtual machine access to a service that requires credentials by adding a secret to the virtual machine.
- Store non-confidential configuration data in a config map so that a pod or another object can consume the data.
- Allow a component to access the API server by associating a service account with that component.



NOTE

OpenShift Virtualization exposes secrets, config maps, and service accounts as virtual machine disks so that you can use them across platforms without additional overhead.

8.11.1. Adding a secret, config map, or service account to a virtual machine

Add a secret, config map, or service account to a virtual machine by using the OpenShift Container Platform web console.

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 **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Environment** tab.
5. Click **Select a resource** and select a secret, config map, or service account from the list. A six character serial number is automatically generated for the selected resource.
6. Click **Save**.
7. Optional. Add another object by clicking **Add Config Map, Secret or Service Account**



NOTE

- a. You can reset the form to the last saved state by clicking **Reload**.
- b. The **Environment** resources are added to the virtual machine as disks. You can mount the secret, config map, or service account as you would mount any other disk.
- c. If the virtual machine is running, changes will not take effect until you restart the virtual machine. The newly added resources are marked as pending changes for both the **Environment** and **Disks** tab in the **Pending Changes** banner at the top of the page.

Verification

1. From the **Virtual Machine Overview** page, click the **Disks** tab.
2. Check to ensure that the secret, config map, or service account is included in the list of disks.
3. Optional. Choose the appropriate method to apply your changes:
 - a. If the virtual machine is running, restart the virtual machine by clicking **Actions** → **Restart Virtual Machine**.
 - b. If the virtual machine is stopped, start the virtual machine by clicking **Actions** → **Start Virtual Machine**.

You can now mount the secret, config map, or service account as you would mount any other disk.


8.11.2. Removing a secret, config map, or service account from a virtual machine

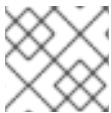
Remove a secret, config map, or service account from a virtual machine by using the OpenShift Container Platform web console.

Prerequisites

- You must have at least one secret, config map, or service account that is attached to a virtual machine.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Environment** tab.
5. Find the item that you want to delete in the list, and click **Remove**  on the right side of the item.
6. Click **Save**.



NOTE

You can reset the form to the last saved state by clicking **Reload**.

Verification

1. From the **Virtual Machine Overview** page, click the **Disks** tab.
2. Check to ensure that the secret, config map, or service account that you removed is no longer included in the list of disks.

8.11.3. Additional resources

- [Providing sensitive data to pods](#)
- [Understanding and creating service accounts](#)
- [Understanding config maps](#)

8.12. INSTALLING VIRTIO DRIVER ON AN EXISTING WINDOWS VIRTUAL MACHINE

8.12.1. Understanding VirtIO drivers

VirtIO drivers are paravirtualized device drivers required for Microsoft Windows virtual machines to run in OpenShift Virtualization. The supported drivers are available in the **container-native-virtualization/virtio-win** container disk of the [Red Hat Ecosystem Catalog](#).

The **container-native-virtualization/virtio-win** container disk must be attached to the virtual machine as a SATA CD drive to enable driver installation. You can install VirtIO drivers during Windows installation on the virtual machine 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 virtual machine.

See also: [Installing Virtio drivers on a new Windows virtual machine](#) .

8.12.2. Supported VirtIO drivers for Microsoft Windows virtual machines

Table 8.1. Supported drivers

Driver name	Hardware ID	Description
viostor	VEN_1AF4&DEV_1001 VEN_1AF4&DEV_1042	The block driver. Sometimes displays as an SCSI Controller in the Other devices group.
viorng	VEN_1AF4&DEV_1005 VEN_1AF4&DEV_1044	The entropy source driver. Sometimes displays as a PCI Device in the Other devices group.
NetKVM	VEN_1AF4&DEV_1000 VEN_1AF4&DEV_1041	The network driver. Sometimes displays as an Ethernet Controller in the Other devices group. Available only if a VirtIO NIC is configured.

8.12.3. Adding VirtIO drivers container disk to a virtual machine

OpenShift Virtualization distributes VirtIO drivers for Microsoft Windows as a container disk, which is available from the [Red Hat Ecosystem Catalog](#) . To install these drivers to a Windows virtual machine, attach the **container-native-virtualization/virtio-win** container disk to the virtual machine as a SATA CD drive in the virtual machine configuration file.

Prerequisites

- Download the **container-native-virtualization/virtio-win** container disk from the [Red Hat Ecosystem Catalog](#). This is not mandatory, because the container disk will be downloaded from the Red Hat registry if it not already present in the cluster, but it can reduce installation time.

Procedure

1. Add the **container-native-virtualization/virtio-win** container disk as a **cdrom** disk in the Windows virtual machine configuration file. The container disk will be downloaded from the registry if it is not already present in the cluster.

```
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 virtual machine disks in the order defined in the **VirtualMachine** configuration file. You can either define other disks for the virtual machine before the **container-native-virtualization/virtio-win** container disk or use the optional **bootOrder** parameter to ensure the virtual machine boots from the correct disk. If you specify the **bootOrder** for a disk, it must be specified for all disks in the configuration.

2. The disk is available once the virtual machine has started:
 - If you add the container disk to a running virtual machine, use **oc apply -f <vm.yaml>** in the CLI or reboot the virtual machine for the changes to take effect.
 - If the virtual machine is not running, use **virtctl start <vm>**.

After the virtual machine has started, the VirtIO drivers can be installed from the attached SATA CD drive.

8.12.4. Installing VirtIO drivers on an existing Windows virtual machine

Install the VirtIO drivers from the attached SATA CD drive to an existing Windows virtual machine.



NOTE

This procedure uses a generic approach to adding drivers to Windows. The process might differ slightly between versions of Windows. See the installation documentation for your version of Windows for specific installation steps.

Procedure

1. Start the virtual machine 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. Right-click the device and select **Properties**.
 - b. Click the **Details** tab and select **Hardware Ids** in the **Property** list.
 - c. 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 virtual machine to complete the driver installation.

8.12.5. Removing the VirtIO container disk from a virtual machine

After installing all required VirtIO drivers to the virtual machine, the **container-native-virtualization/virtio-win** container disk no longer needs to be attached to the virtual machine. Remove the **container-native-virtualization/virtio-win** container disk from the virtual machine configuration file.

Procedure

1. Edit the configuration file and remove the **disk** and the **volume**.

```
$ oc edit vm <vm-name>

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

2. Reboot the virtual machine for the changes to take effect.

8.13. INSTALLING VIRTIO DRIVER ON A NEW WINDOWS VIRTUAL MACHINE

8.13.1. Prerequisites

- Windows installation media accessible by the virtual machine, such as [importing an ISO into a data volume](#) and attaching it to the virtual machine.

8.13.2. Understanding VirtIO drivers

VirtIO drivers are paravirtualized device drivers required for Microsoft Windows virtual machines to run in OpenShift Virtualization. The supported drivers are available in the **container-native-virtualization/virtio-win** container disk of the [Red Hat Ecosystem Catalog](#).

The **container-native-virtualization/virtio-win** container disk must be attached to the virtual machine as a SATA CD drive to enable driver installation. You can install VirtIO drivers during Windows installation on the virtual machine 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 virtual machine.

See also: [Installing VirtIO driver on an existing Windows virtual machine](#).

8.13.3. Supported VirtIO drivers for Microsoft Windows virtual machines

Table 8.2. Supported drivers

Driver name	Hardware ID	Description
viostor	VEN_1AF4&DEV_1001 VEN_1AF4&DEV_1042	The block driver. Sometimes displays as an SCSI Controller in the Other devices group.
viornng	VEN_1AF4&DEV_1005 VEN_1AF4&DEV_1044	The entropy source driver. Sometimes displays as a PCI Device in the Other devices group.
NetKVM	VEN_1AF4&DEV_1000 VEN_1AF4&DEV_1041	The network driver. Sometimes displays as an Ethernet Controller in the Other devices group. Available only if a VirtIO NIC is configured.

8.13.4. Adding VirtIO drivers container disk to a virtual machine

OpenShift Virtualization distributes VirtIO drivers for Microsoft Windows as a container disk, which is available from the [Red Hat Ecosystem Catalog](#). To install these drivers to a Windows virtual machine, attach the **container-native-virtualization/virtio-win** container disk to the virtual machine as a SATA CD drive in the virtual machine configuration file.

Prerequisites

- Download the **container-native-virtualization/virtio-win** container disk from the [Red Hat Ecosystem Catalog](#). This is not mandatory, because the container disk will be downloaded from the Red Hat registry if it not already present in the cluster, but it can reduce installation time.

Procedure

1. Add the **container-native-virtualization/virtio-win** container disk as a **cdrom** disk in the Windows virtual machine configuration file. The container disk will be downloaded from the registry if it is not already present in the cluster.

```
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 virtual machine disks in the order defined in the **VirtualMachine** configuration file. You can either define other disks for the virtual machine
2. The disk is available once the virtual machine has started:
 - If you add the container disk to a running virtual machine, use **oc apply -f <vm.yaml>** in the CLI or reboot the virtual machine for the changes to take effect.
 - If the virtual machine is not running, use **virtctl start <vm>**.

After the virtual machine has started, the VirtIO drivers can be installed from the attached SATA CD drive.

8.13.5. Installing VirtIO drivers during Windows installation

Install the VirtIO drivers from the attached SATA CD driver during Windows installation.



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.

Procedure

1. Start the virtual machine and connect to a graphical console.
2. Begin the Windows installation process.
3. Select the **Advanced** installation.
4. The storage destination will not be recognized until the driver is loaded. Click **Load driver**.
5. The drivers are attached as a SATA CD drive. Click **OK** and browse the CD drive for the storage driver to load. The drivers are arranged hierarchically according to their driver type, operating system, and CPU architecture.
6. Repeat the previous two steps for all required drivers.
7. Complete the Windows installation.

8.13.6. Removing the VirtIO container disk from a virtual machine

After installing all required VirtIO drivers to the virtual machine, the **container-native-virtualization/virtio-win** container disk no longer needs to be attached to the virtual machine. Remove the **container-native-virtualization/virtio-win** container disk from the virtual machine configuration file.

Procedure

1. Edit the configuration file and remove the **disk** and the **volume**.

```
$ oc edit vm <vm-name>
```

```
spec:
```

```

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

```

2. Reboot the virtual machine for the changes to take effect.

8.14. ADVANCED VIRTUAL MACHINE MANAGEMENT

8.14.1. Working with resource quotas for virtual machines

Create and manage resource quotas for virtual machines.

8.14.1.1. Setting resource quota limits for virtual machines

Resource quotas that only use requests automatically work with virtual machines (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:
  running: false
  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.

8.14.1.2. Additional resources

- [Resource quotas per project](#)
- [Resource quotas across multiple projects](#)

8.14.2. Specifying nodes for virtual machines

You can place virtual machines (VMs) on specific nodes by using node placement rules.

8.14.2.1. 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.

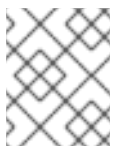
You can use the following rule types in the **spec** field of a **VirtualMachine** manifest:

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.



NOTE

Affinity rules only apply during scheduling. OpenShift Container Platform does not reschedule running workloads if the constraints are no longer met.

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.

8.14.2.2. Node placement examples

The following example YAML file snippets use **nodePlacement**, **affinity**, and **tolerations** fields to customize node placement for virtual machines.

8.14.2.2.1. 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
    ...

```

8.14.2.2.2. 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:
    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.

8.14.2.2.3. 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:
    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

```

```

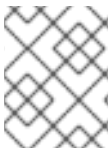
values:
  - example-node-label-value
...

```

- 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.

8.14.2.2.4. 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"
...

```

8.14.2.3. 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](#)

8.14.3. Configuring certificate rotation

Configure certificate rotation parameters to replace existing certificates.

8.14.3.1. Configuring certificate rotation

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

Procedure

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

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

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.

8.14.3.2. 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.

8.14.4. Automating management tasks

You can automate OpenShift Virtualization management tasks by using Red Hat Ansible Automation Platform. Learn the basics by using an Ansible Playbook to create a new virtual machine.

8.14.4.1. About Red Hat Ansible Automation

[Ansible](#) is an automation tool used to configure systems, deploy software, and perform rolling updates. Ansible includes support for OpenShift Virtualization, and Ansible modules enable you to automate cluster management tasks such as template, persistent volume claim, and virtual machine operations.

Ansible provides a way to automate OpenShift Virtualization management, which you can also accomplish by using the **oc** CLI tool or APIs. Ansible is unique because it allows you to integrate [KubeVirt modules](#) with other Ansible modules.

8.14.4.2. Automating virtual machine creation

You can use the **kubevirt_vm** Ansible Playbook to create virtual machines in your OpenShift Container Platform cluster using Red Hat Ansible Automation Platform.

Prerequisites

- [Red Hat Ansible Engine](#) version 2.8 or newer

Procedure

1. Edit an Ansible Playbook YAML file so that it includes the **kubevirt_vm** task:

```

kubevirt_vm:
  namespace:
  name:
  cpu_cores:
  memory:
  disks:
    - name:
      volume:
        containerDisk:
          image:
        disk:
          bus:

```

**NOTE**

This snippet only includes the **kubevirt_vm** portion of the playbook.

2. Edit the values to reflect the virtual machine you want to create, including the **namespace**, the number of **cpu_cores**, the **memory**, and the **disks**. For example:

```
kubevirt_vm:
  namespace: default
  name: vm1
  cpu_cores: 1
  memory: 64Mi
  disks:
    - name: containerdisk
      volume:
        containerDisk:
          image: kubevirt/cirros-container-disk-demo:latest
      disk:
        bus: virtio
```

3. If you want the virtual machine to boot immediately after creation, add **state: running** to the YAML file. For example:

```
kubevirt_vm:
  namespace: default
  name: vm1
  state: running 1
  cpu_cores: 1
```

- 1** Changing this value to **state: absent** deletes the virtual machine, if it already exists.

4. Run the **ansible-playbook** command, using your playbook's file name as the only argument:

```
$ ansible-playbook create-vm.yaml
```

5. Review the output to determine if the play was successful:

Example output

```
(...)
TASK [Create my first VM] *****
changed: [localhost]

PLAY RECAP
*****
localhost      :ok=2  changed=1  unreachable=0  failed=0  skipped=0
rescued=0  ignored=0
```

6. If you did not include **state: running** in your playbook file and you want to boot the VM now, edit the file so that it includes **state: running** and run the playbook again:

```
$ ansible-playbook create-vm.yaml
```

To verify that the virtual machine was created, try to [access the VM console](#).

8.14.4.3. Example: Ansible Playbook for creating virtual machines

You can use the `kubevirt_vm` Ansible Playbook to automate virtual machine creation.

The following YAML file is an example of the `kubevirt_vm` playbook. It includes sample values that you must replace with your own information if you run the playbook.

```
---
- name: Ansible Playbook 1
  hosts: localhost
  connection: local
  tasks:
    - name: Create my first VM
      kubevirt_vm:
        namespace: default
        name: vm1
        cpu_cores: 1
        memory: 64Mi
        disks:
          - name: containerdisk
            volume:
              containerDisk:
                image: kubevirt/cirros-container-disk-demo:latest
            disk:
              bus: virtio
```

Additional information

- [Intro to Playbooks](#)
- [Tools for Validating Playbooks](#)

8.14.5. Using EFI mode for virtual machines

You can boot a virtual machine (VM) in Extensible Firmware Interface (EFI) mode.

8.14.5.1. About EFI mode for virtual machines

Extensible Firmware Interface (EFI), like legacy BIOS, initializes hardware components and operating system image files when a computer starts. EFI 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.



NOTE

OpenShift Virtualization only supports a virtual machine (VM) with Secure Boot when using EFI mode. If Secure Boot is not enabled, the VM crashes repeatedly. However, the VM might not support Secure Boot. Before you boot a VM, verify that it supports Secure Boot by checking the VM settings.

8.14.5.2. Booting virtual machines in EFI mode

You can configure a virtual machine to boot in EFI mode by editing the VM or VMI manifest.

Prerequisites

- Install the OpenShift CLI (**oc**).

Procedure

1. Create a YAML file that defines a VM object or a Virtual Machine Instance (VMI) object. Use the firmware stanza of the example YAML file:

Booting in EFI mode with secure boot active

```
apiversion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  labels:
    special: vmi-secureboot
    name: vmi-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 EFI mode to occur.
- 2 OpenShift Virtualization only supports a virtual machine (VM) with Secure Boot when using EFI mode. If Secure Boot is not enabled, the VM crashes repeatedly. However, the VM might not support Secure Boot. Before you boot a VM, verify that it supports Secure Boot by checking the VM settings.

2. Apply the manifest to your cluster by running the following command:

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

8.14.6. 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.

8.14.6.1. Prerequisites

- A Linux bridge must be [connected](#).
- The PXE server must be connected to the same VLAN as the bridge.

8.14.6.2. 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.

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
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "pxe-net-conf",
    "plugins": [
      {
        "type": "cnv-bridge",
        "bridge": "br1",
        "vlan": 1 1
      },
      {
        "type": "cnv-tuning" 2
      }
    ]
  }'
```

- 1** Optional: The VLAN tag.
- 2** The **cnv-tuning** plugin provides support for custom MAC addresses.

**NOTE**

The virtual machine instance will be attached to the bridge **br1** through an access port with the requested VLAN.

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
```

1. Wait for the virtual machine instance to run:

```
$ oc get vmi vmi-pxe-boot -o yaml | grep -i phase
phase: Running
```

2. View the virtual machine instance using VNC:

```
$ virtctl vnc vmi-pxe-boot
```

3. Watch the boot screen to verify that the PXE boot is successful.

4. Log in to the virtual machine instance:

```
$ virtctl console vmi-pxe-boot
```

5. 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
```

8.14.6.3. Template: Virtual machine instance configuration file for PXE booting

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  creationTimestamp: null
  labels:
    special: vmi-pxe-boot
  name: vmi-pxe-boot
spec:
  domain:
  devices:
    disks:
    - disk:
        bus: virtio
        name: containerdisk
        bootOrder: 2
    - disk:
```



```

    bus: virtio
    name: cloudinitdisk
  interfaces:
  - masquerade: {}
    name: default
  - bridge: {}
    name: pxe-net
    macAddress: de:00:00:00:de
    bootOrder: 1
  machine:
    type: ""
  resources:
    requests:
      memory: 1024M
  networks:
  - name: default
    pod: {}
  - multus:
      networkName: pxe-net-conf
      name: pxe-net
  terminationGracePeriodSeconds: 0
  volumes:
  - name: containerdisk
    containerDisk:
      image: kubevirt/fedora-cloud-container-disk-demo
  - cloudInitNoCloud:
      userData: |
        #!/bin/bash
        echo "fedora" | passwd fedora --stdin
      name: cloudinitdisk
  status: {}

```

8.14.7. Managing guest memory

If you want to adjust guest memory settings to suit a specific use case, you can do so by editing the guest's YAML configuration file. OpenShift Virtualization allows you to configure guest memory overcommitment and disable guest memory overhead accounting.



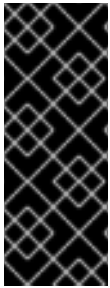
WARNING

The following procedures increase the chance that virtual machine processes will be killed due to memory pressure. Proceed only if you understand the risks.

8.14.7.1. Configuring guest memory overcommitment

If your virtual workload requires more memory than available, you can use memory overcommitment to allocate all or most of the host's memory to your virtual machine instances (VMIs). Enabling memory overcommitment means that you can maximize resources that are normally reserved for the host.

For example, if the host has 32 GB RAM, you can use memory overcommitment to fit 8 virtual machines (VMs) with 4 GB RAM each. This allocation works under the assumption that the virtual machines will not use all of their memory at the same time.



IMPORTANT

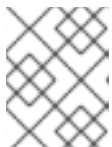
Memory overcommitment increases the potential for virtual machine processes to be killed due to memory pressure (OOM killed).

The potential for a VM to be OOM killed varies based on your specific configuration, node memory, available swap space, virtual machine memory consumption, the use of kernel same-page merging (KSM), and other factors.

Procedure

1. To explicitly tell the virtual machine instance that it has more memory available than was requested from the cluster, edit the virtual machine configuration file and set **spec.domain.memory.guest** to a higher value than **spec.domain.resources.requests.memory**. This process is called memory overcommitment. In this example, **1024M** is requested from the cluster, but the virtual machine instance is told that it has **2048M** available. As long as there is enough free memory available on the node, the virtual machine instance will consume up to 2048M.

```
kind: VirtualMachine
spec:
  template:
    domain:
      resources:
        requests:
          memory: 1024M
        memory:
          guest: 2048M
```



NOTE

The same eviction rules as those for pods apply to the virtual machine instance if the node is under memory pressure.

2. Create the virtual machine:

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

8.14.7.2. Disabling guest memory overhead accounting

A small amount of memory is requested by each virtual machine instance in addition to the amount that you request. This additional memory is used for the infrastructure that wraps each **VirtualMachineInstance** process.

Though it is not usually advisable, it is possible to increase the virtual machine instance density on the node by disabling guest memory overhead accounting.



IMPORTANT

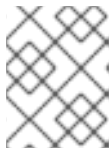
Disabling guest memory overhead accounting increases the potential for virtual machine processes to be killed due to memory pressure (OOM killed).

The potential for a VM to be OOM killed varies based on your specific configuration, node memory, available swap space, virtual machine memory consumption, the use of kernel same-page merging (KSM), and other factors.

Procedure

1. To disable guest memory overhead accounting, edit the YAML configuration file and set the **overcommitGuestOverhead** value to **true**. This parameter is disabled by default.

```
kind: VirtualMachine
spec:
  template:
    domain:
      resources:
        overcommitGuestOverhead: true
      requests:
        memory: 1024M
```



NOTE

If **overcommitGuestOverhead** is enabled, it adds the guest overhead to memory limits, if present.

2. Create the virtual machine:

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

8.14.8. Using huge pages with virtual machines

You can use huge pages as backing memory for virtual machines in your cluster.

8.14.8.1. Prerequisites

- Nodes must have [pre-allocated huge pages configured](#).

8.14.8.2. 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.

8.14.8.3. 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.

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
```

8.14.9. Enabling dedicated resources for virtual machines

To improve performance, you can dedicate node resources, such as CPU, to a virtual machine.

8.14.9.1. 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.

8.14.9.2. 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.

8.14.9.3. Enabling dedicated resources for a virtual machine

You can enable dedicated resources for a virtual machine in the **Details** tab. Virtual machines that were created by using a Red Hat template or the wizard can be enabled with dedicated resources.

Procedure

1. Click **Workloads** → **Virtual Machines** from the side menu.
2. Select a virtual machine to open the **Virtual Machine** tab.
3. Click the **Details** tab.
4. Click the pencil icon to the right of the **Dedicated Resources** field to open the **Dedicated Resources** window.
5. Select **Schedule this workload with dedicated resources (guaranteed policy)**
6. Click **Save**.

8.14.10. 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.

8.14.10.1. Understanding 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.
forbid	The VM is not scheduled even if the feature is supported by the host CPU and CPU node discovery is enabled.

8.14.10.2. 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 instance (VMI):

```

apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: myvmi
spec:
  domain:
    cpu:
      features:
        - name: apic 1
          policy: require 2

```

- 1** Name of the CPU feature for the VM or VMI.
- 2** Policy attribute for the VM or VMI.

8.14.10.3. Scheduling virtual machines with the supported CPU model

You can configure a CPU model for a virtual machine (VM) or a virtual machine instance (VMI) 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 VMI:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  name: myvmi
spec:
  domain:
    cpu:
      model: Conroe 1
```

- 1 CPU model for the VMI.

8.14.10.4. 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 instance (VMI):

```
apiVersion: kubevirt/v1alpha3
kind: VirtualMachineInstance
metadata:
  name: myvmi
spec:
  domain:
    cpu:
      model: host-model 1
```

- 1 The VM or VMI that inherits the CPU model of the node where it is scheduled.

8.14.11. Configuring PCI passthrough

The Peripheral Component Interconnect (PCI) passthrough feature enables you to access and manage hardware devices from a virtual machine. 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).

8.14.11.1. 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.

8.14.11.1.1. Adding kernel arguments to enable the IOMMU driver

To enable the IOMMU (Input-Output Memory Management Unit) driver in the kernel, create the **MachineConfig** object and add the kernel arguments.

Prerequisites

- Administrative privilege to a working OpenShift Container Platform cluster.
- Intel or AMD CPU hardware.
- Intel Virtualization Technology for Directed I/O extensions or AMD IOMMU in the BIOS (Basic Input/Output System) is enabled.

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 1
  name: 100-worker-iommu 2
spec:
  config:
    ignition:
      version: 3.2.0
  kernelArguments:
    - intel_iommu=on 3
  ...
```

- 1** Applies the new kernel argument only to worker nodes.
- 2** The **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**.
- 3** 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

- Verify that the new **MachineConfig** object was added.

```
$ oc get MachineConfig
```


8.14.11.1.2. 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.

Procedure

- 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)
```

- Create a Butane config file, `100-worker-vfiopci.bu`, binding the PCI device to the VFIO driver.



NOTE

See "Creating machine configs with Butane" for information about Butane.

Example

```
variant: openshift
version: 4.8.0
metadata:
  name: 100-worker-vfiopci
  labels:
    machineconfiguration.openshift.io/role: worker ❶
storage:
  files:
    - path: /etc/modprobe.d/vfio.conf
      mode: 0644
      overwrite: true
      contents:
        inline: |
          options vfio-pci ids=10de:1eb8 ❷
    - path: /etc/modules-load.d/vfio-pci.conf ❸
      mode: 0644
      overwrite: true
      contents:
        inline: vfio-pci
```

❶ Applies the new kernel argument only to worker nodes.

❷

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

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	3.2.0	25h
00-worker	d3da910bfa9f4b599af4ed7f5ac270d55950a3a1	3.2.0	25h
01-master-container-runtime	d3da910bfa9f4b599af4ed7f5ac270d55950a3a1	3.2.0	25h
01-master-kubelet	d3da910bfa9f4b599af4ed7f5ac270d55950a3a1	3.2.0	25h
01-worker-container-runtime	d3da910bfa9f4b599af4ed7f5ac270d55950a3a1	3.2.0	25h
01-worker-kubelet	d3da910bfa9f4b599af4ed7f5ac270d55950a3a1	3.2.0	25h
100-worker-iommu		3.2.0	30s
100-worker-vfiopci-configuration		3.2.0	30s

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
```

8.14.11.1.3. 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).

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 leaves the allocation and monitoring to an external device plugin.



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
```

8.14.11.1.4. 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).

Procedure

- Edit the **HyperConverged** CR in your default editor by running the following command:

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

- 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

```

8.14.11.2. 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.

8.14.11.2.1. 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)
```

8.14.11.3. Additional resources

- [Enabling Intel VT-X and AMD-V Virtualization Hardware Extensions in BIOS](#)
- [Managing file permissions](#)
- [Post-installation machine configuration tasks](#)

8.14.12. Configuring a watchdog

Expose a watchdog by configuring the virtual machine (VM) for a watchdog device, installing the watchdog, and starting the watchdog service.

8.14.12.1. Prerequisites

- The virtual machine must have kernel support for an **i6300esb** watchdog device. Red Hat Enterprise Linux (RHEL) images support **i6300esb**.

8.14.12.2. Defining a watchdog device

Define how the watchdog proceeds when the operating system (OS) no longer responds.

Table 8.3. Available actions

poweroff	The virtual machine (VM) powers down immediately. If spec.running is set to true , or spec.runStrategy is not set to manual , then the VM reboots.
reset	The VM reboots in place and the guest OS cannot react. Because the length of time required for the guest OS to reboot can cause liveness probes to timeout, use of this option is discouraged. This timeout can extend the time it takes the VM to reboot if cluster-level protections notice the liveness probe failed and forcibly reschedule it.
shutdown	The VM gracefully powers down by stopping all services.

Procedure

1. Create a YAML file with the following contents:

```

apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm2-rhel84-watchdog
  name: <vm-name>
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm2-rhel84-watchdog
    spec:
      domain:
        devices:
          watchdog:
            name: <watchdog>
            i6300esb:
              action: "poweroff" 1
  ...

```

- 1 Specify the **watchdog** action (**poweroff**, **reset**, or **shutdown**).

The example above configures the **i6300esb** watchdog device on a RHEL8 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
```



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
```

- Terminate the watchdog service:

```
# pkill -9 watchdog
```

8.14.12.3. Installing a watchdog device

Install the **watchdog** package on your virtual machine and start the watchdog service.

Procedure

1. As a root user, install the **watchdog** package and dependencies:

```
# yum install watchdog
```

2. Uncomment the following line in the **/etc/watchdog.conf** file, and save the changes:

```
#watchdog-device = /dev/watchdog
```

3. Enable the watchdog service to start on boot:

```
# systemctl enable --now watchdog.service
```

8.14.12.4. Additional resources

- [Monitoring application health by using health checks](#)

8.15. IMPORTING VIRTUAL MACHINES

8.15.1. TLS certificates for data volume imports

8.15.1.1. Adding TLS certificates for authenticating data volume imports

TLS certificates for registry or HTTPS endpoints must be added to a config map to import data from these sources. This config map must be present in the namespace of the destination data volume.

Create the config map by referencing the relative file path for the TLS certificate.

Procedure

1. Ensure you are in the correct namespace. The config map can only be referenced by data volumes if it is in the same namespace.

```
$ oc get ns
```

2. Create the config map:

```
$ oc create configmap <configmap-name> --from-file=</path/to/file/ca.pem>
```

8.15.1.2. Example: Config map created from a TLS certificate

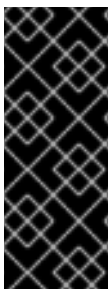
The following example is of a config map created from **ca.pem** TLS certificate.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tls-certs
data:
  ca.pem: |
    -----BEGIN CERTIFICATE-----
    ... <base64 encoded cert> ...
    -----END CERTIFICATE-----
```

8.15.2. Importing virtual machine images with data volumes

Use the Containerized Data Importer (CDI) to import a virtual machine image into a persistent volume claim (PVC) by using a data volume. You can attach a data volume to a virtual machine for persistent storage.

The virtual machine image can be hosted at an HTTP or HTTPS endpoint, or built into a container disk and stored in a container registry.



IMPORTANT

When you import a disk image into a PVC, the disk image is expanded to use the full storage capacity that is requested in the PVC. To use this space, the disk partitions and file system(s) in the virtual machine might need to be expanded.

The resizing procedure varies based on the operating system installed on the virtual machine. See the operating system documentation for details.

8.15.2.1. Prerequisites

- If the endpoint requires a TLS certificate, the certificate must be [included in a config map](#) in the same namespace as the data volume and referenced in the data volume configuration.

- To import a container disk:
 - You might need to [prepare a container disk from a virtual machine image](#) and store it in your container registry before importing it.
 - If the container registry does not have TLS, you must [add the registry to the `insecureRegistries` field of the `HyperConverged` custom resource](#) before you can import a container disk from it.
- You might need to [define a storage class or prepare CDI scratch space](#) for this operation to complete successfully.

8.15.2.2. 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)	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2** ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

Unsupported operation

* Requires scratch space

** Requires scratch space if a custom certificate authority is required



NOTE

CDI now uses the OpenShift Container Platform [cluster-wide proxy configuration](#).

8.15.2.3. About data volumes

DataVolume objects are custom resources that are provided by the Containerized Data Importer (CDI) project. Data volumes orchestrate import, clone, and upload operations that are associated with an underlying persistent volume claim (PVC). Data volumes are integrated with OpenShift Virtualization, and they prevent a virtual machine from being started before the PVC has been prepared.

8.15.2.4. Importing a virtual machine image into storage by using a data volume

You can import a virtual machine image into storage by using a data volume.

The virtual machine image can be hosted at an HTTP or HTTPS endpoint or the image can be built into a container disk and stored in a container registry.

You specify the data source for the image in a **VirtualMachine** configuration file. When the virtual machine is created, the data volume with the virtual machine image is imported into storage.

Prerequisites

- To import a virtual machine image you must have the following:
 - A virtual machine disk image in RAW, ISO, or QCOW2 format, optionally compressed by using **xz** or **gz**.
 - An HTTP or HTTPS endpoint where the image is hosted, along with any authentication credentials needed to access the data source.
- To import a container disk, you must have a virtual machine image built into a container disk and stored in a container registry, along with any authentication credentials needed to access the data source.
- If the virtual machine must communicate with servers that use self-signed certificates or certificates not signed by the system CA bundle, you must create a config map in the same namespace as the data volume.

Procedure

1. If your data source requires authentication, create a **Secret** manifest, specifying the data source credentials, and save it as **endpoint-secret.yaml**:

```
apiVersion: v1
kind: Secret
metadata:
  name: endpoint-secret 1
  labels:
    app: containerized-data-importer
type: Opaque
data:
  accessKeyId: "" 2
  secretKey: "" 3
```

- 1 Specify the name of the **Secret**.
- 2 Specify the Base64-encoded key ID or user name.
- 3 Specify the Base64-encoded secret key or password.

2. Apply the **Secret** manifest:

```
$ oc apply -f endpoint-secret.yaml
```

3. Edit the **VirtualMachine** manifest, specifying the data source for the virtual machine image you want to import, and save it as **vm-fedora-datavolume.yaml**:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  creationTimestamp: null
```

```

labels:
  kubevirt.io/vm: vm-fedora-datavolume
name: vm-fedora-datavolume 1
spec:
  dataVolumeTemplates:
  - metadata:
    creationTimestamp: null
    name: fedora-dv 2
    spec:
      storage:
        resources:
          requests:
            storage: 10Gi
        storageClassName: local
      source:
        http: 3
          url: "https://mirror.arizona.edu/fedora/linux/releases/35/Cloud/x86_64/images/Fedora-
Cloud-Base-35-1.2.x86_64.qcow2" 4
          secretRef: endpoint-secret 5
          certConfigMap: "" 6
        status: {}
      running: true
    template:
      metadata:
        creationTimestamp: null
        labels:
          kubevirt.io/vm: vm-fedora-datavolume
      spec:
        domain:
          devices:
            disks:
              - disk:
                bus: virtio
                name: datavolumedisk1
          machine:
            type: ""
          resources:
            requests:
              memory: 1.5Gi
          terminationGracePeriodSeconds: 60
          volumes:
            - dataVolume:
              name: fedora-dv
              name: datavolumedisk1
        status: {}

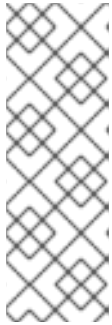
```

- 1** Specify the name of the virtual machine.
- 2** Specify the name of the data volume.
- 3** Specify **http** for an HTTP or HTTPS endpoint. Specify **registry** for a container disk image imported from a registry.
- 4** The source of the virtual machine image you want to import. This example references a

- 5 Required if you created a **Secret** for the data source.
- 6 Optional: Specify a CA certificate config map.

4. Create the virtual machine:

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



NOTE

The **oc create** command creates the data volume and the virtual machine. 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 virtual machine.

Data volume provisioning happens in the background, so there is no need to monitor the process.

Verification

1. The importer pod downloads the virtual machine image or container disk from the specified URL and stores it on the provisioned PV. View the status of the importer pod by running the following command:

```
$ oc get pods
```

2. Monitor the data volume until its status is **Succeeded** by running the following command:

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

- 1 Specify the data volume name that you defined in the **VirtualMachine** manifest.

3. Verify that provisioning is complete and that the virtual machine has started by accessing its serial console:

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

8.15.2.5. Additional resources

- [Configure preallocation mode](#) to improve write performance for data volume operations.

8.15.3. Importing virtual machine images into block storage with data volumes

You can import an existing virtual machine image into your OpenShift Container Platform cluster. OpenShift Virtualization uses data volumes to automate the import of data and the creation of an underlying persistent volume claim (PVC).



IMPORTANT

When you import a disk image into a PVC, the disk image is expanded to use the full storage capacity that is requested in the PVC. To use this space, the disk partitions and file system(s) in the virtual machine might need to be expanded.

The resizing procedure varies based on the operating system that is installed on the virtual machine. See the operating system documentation for details.

8.15.3.1. Prerequisites

- If you require scratch space according to the [CDI supported operations matrix](#), you must first [define a storage class or prepare CDI scratch space](#) for this operation to complete successfully.

8.15.3.2. About data volumes

DataVolume objects are custom resources that are provided by the Containerized Data Importer (CDI) project. Data volumes orchestrate import, clone, and upload operations that are associated with an underlying persistent volume claim (PVC). Data volumes are integrated with OpenShift Virtualization, and they prevent a virtual machine from being started before the PVC has been prepared.

8.15.3.3. About block persistent volumes

A block persistent volume (PV) is a PV that is backed by a raw block device. These volumes do not have a file system and can provide performance benefits for virtual machines by reducing overhead.

Raw block volumes are provisioned by specifying **volumeMode: Block** in the PV and persistent volume claim (PVC) specification.

8.15.3.4. Creating a local block persistent volume

Create a local block persistent volume (PV) on a node by populating a file and mounting it as a loop device. You can then reference this loop device in a PV manifest as a **Block** volume and use it as a block device for a virtual machine image.

Procedure

1. Log in as **root** to the node on which to create the local PV. This procedure uses **node01** for its examples.
2. Create a file and populate it with null characters so that it can be used as a block device. The following example creates a file **loop10** with a size of 2Gb (20 100Mb blocks):

```
$ dd if=/dev/zero of=<loop10> bs=100M count=20
```

3. Mount the **loop10** file as a loop device.

```
$ losetup </dev/loop10>d3 <loop10> 1 2
```

1 File path where the loop device is mounted.

2 The file created in the previous step to be mounted as the loop device.

4. Create a **PersistentVolume** manifest that references the mounted loop device.

```

kind: PersistentVolume
apiVersion: v1
metadata:
  name: <local-block-pv10>
  annotations:
spec:
  local:
    path: </dev/loop10> ❶
  capacity:
    storage: <2Gi>
  volumeMode: Block ❷
  storageClassName: local ❸
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - <node01> ❹

```

- ❶ The path of the loop device on the node.
- ❷ Specifies it is a block PV.
- ❸ Optional: Set a storage class for the PV. If you omit it, the cluster default is used.
- ❹ The node on which the block device was mounted.

5. Create the block PV.

```
# oc create -f <local-block-pv10.yaml> ❶
```

- ❶ The file name of the persistent volume created in the previous step.

8.15.3.5. Importing a virtual machine image into block storage by using a data volume

You can import a virtual machine image into block storage by using a data volume. You reference the data volume in a **VirtualMachine** manifest before you create a virtual machine.

Prerequisites

- A virtual machine disk image in RAW, ISO, or QCOW2 format, optionally compressed by using **xz** or **gz**.
- An HTTP or HTTPS endpoint where the image is hosted, along with any authentication credentials needed to access the data source.

Procedure

1. If your data source requires authentication, create a **Secret** manifest, specifying the data source credentials, and save it as **endpoint-secret.yaml**:

```
apiVersion: v1
kind: Secret
metadata:
  name: endpoint-secret 1
  labels:
    app: containerized-data-importer
type: Opaque
data:
  accessKeyId: "" 2
  secretKey: "" 3
```

- 1 Specify the name of the **Secret**.
- 2 Specify the Base64-encoded key ID or user name.
- 3 Specify the Base64-encoded secret key or password.

2. Apply the **Secret** manifest:

```
$ oc apply -f endpoint-secret.yaml
```

3. Create a **DataVolume** manifest, specifying the data source for the virtual machine image and **Block** for **storage.volumeMode**.

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: import-pv-datavolume 1
spec:
  storageClassName: local 2
  source:
    http:
      url: "https://mirror.arizona.edu/fedora/linux/releases/35/Cloud/x86_64/images/Fedora-Cloud-Base-35-1.2.x86_64.qcow2" 3
      secretRef: endpoint-secret 4
  storage:
    volumeMode: Block 5
  resources:
    requests:
      storage: 10Gi
```

- 1 Specify the name of the data volume.
- 2 Optional: Set the storage class or omit it to accept the cluster default.
- 3 Specify the HTTP or HTTPS URL of the image to import.
- 4 Required if you created a **Secret** for the data source.

- 5 The volume mode and access mode are detected automatically for known storage provisioners. Otherwise, specify **Block**.

4. Create the data volume to import the virtual machine image:

```
$ oc create -f import-pv-datavolume.yaml
```

You can reference this data volume in a **VirtualMachine** manifest before you create a virtual machine.

8.15.3.6. CDI supported operations matrix

This matrix shows the supported CDI operations for content types against endpoints, and which of these operations requires scratch space.

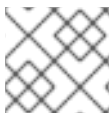
Content types	HTTP	HTTPS	HTTP basic auth	Registry	Upload
KubeVirt (QCOW2)	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2** ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

Unsupported operation

* Requires scratch space

** Requires scratch space if a custom certificate authority is required



NOTE

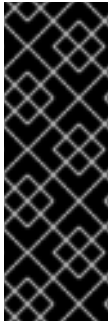
CDI now uses the OpenShift Container Platform [cluster-wide proxy configuration](#).

8.15.3.7. Additional resources

- [Configure preallocation mode](#) to improve write performance for data volume operations.

8.15.4. Importing a single Red Hat Virtualization virtual machine

You can import a single Red Hat Virtualization (RHV) virtual machine into OpenShift Virtualization by using the VM Import wizard or the CLI.



IMPORTANT

Importing a RHV VM is a deprecated feature. Deprecated functionality is still included in OpenShift Virtualization and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

For the most recent list of major functionality that has been deprecated or removed within OpenShift Virtualization, refer to the *Deprecated and removed features* section of the OpenShift Virtualization release notes.

This feature will be replaced by the [Migration Toolkit for Virtualization](#).

8.15.4.1. OpenShift Virtualization storage feature matrix

The following table describes the OpenShift Virtualization storage types that support VM import.

Table 8.4. OpenShift Virtualization storage feature matrix

	RHV VM import
OpenShift Container Storage: RBD block-mode volumes	Yes
OpenShift Virtualization hostpath provisioner	No
Other multi-node writable storage	Yes [1]
Other single-node writable storage	Yes [2]

1. PVCs must request a ReadWriteMany access mode.
2. PVCs must request a ReadWriteOnce access mode.

8.15.4.2. Prerequisites for importing a virtual machine

Importing a virtual machine from Red Hat Virtualization (RHV) into OpenShift Virtualization has the following prerequisites:

- You must have admin user privileges.
- Storage:
 - The OpenShift Virtualization local and shared persistent storage classes must support VM import.
 - If you are using Ceph RBD block-mode volumes and the available storage space is too small for the virtual disk, the import process bar stops at 75% for more than 20 minutes and the migration does not succeed. No error message is displayed in the web console. [BZ#1910019](#)
- Networks:
 - The RHV and OpenShift Virtualization networks must either have the same name or be mapped to each other.

- The RHV VM network interface must be **e1000**, **rtl8139**, or **virtio**.
- VM disks:
 - The disk interface must be **sata**, **virtio_scsi**, or **virtio**.
 - The disk must not be configured as a direct LUN.
 - The disk status must not be **illegal** or **locked**.
 - The storage type must be **image**.
 - SCSI reservation must be disabled.
 - **ScsiGenericIO** must be disabled.
- VM configuration:
 - If the VM uses GPU resources, the nodes providing the GPUs must be configured.
 - The VM must not be configured for vGPU resources.
 - The VM must not have snapshots with disks in an **illegal** state.
 - The VM must not have been created with OpenShift Container Platform and subsequently added to RHV.
 - The VM must not be configured for USB devices.
 - The watchdog model must not be **diag288**.

8.15.4.3. Importing a virtual machine with the VM Import wizard

You can import a single virtual machine with the VM Import wizard.

Procedure

1. In the web console, click **Workloads** → **Virtual Machines**.
2. Click **Create Virtual Machine** and select **Import with Wizard**.
3. Select **Red Hat Virtualization (RHV)** from the **Provider** list.
4. Select **Connect to New Instance** or a saved RHV instance.
 - If you select **Connect to New Instance**, fill in the following fields:
 - **API URL**: For example, **https://<RHV_Manager_FQDN>/ovirt-engine/api**
 - **CA certificate**: Click **Browse** to upload the RHV Manager CA certificate or paste the CA certificate into the field.
View the CA certificate by running the following command:

```
$ openssl s_client -connect <RHV_Manager_FQDN>:443 -showcerts < /dev/null
```

The CA certificate is the second certificate in the output.

- **Username:** RHV Manager user name, for example, **ocpadmin@internal**
 - **Password:** RHV Manager password
 - If you select a saved RHV instance, the wizard connects to the RHV instance using the saved credentials.
5. Click **Check and Save** and wait for the connection to complete.

**NOTE**

The connection details are stored in a secret. If you add a provider with an incorrect URL, user name, or password, click **Workloads** → **Secrets** and delete the provider secret.

6. Select a cluster and a virtual machine.
7. Click **Next**.
8. In the **Review** screen, review your settings.
9. Optional: You can select **Start virtual machine on creation**
10. Click **Edit** to update the following settings:
- **General** → **Name:** The VM name is limited to 63 characters.
 - **General** → **Description:** Optional description of the VM.
 - **Storage Class:** Select **NFS** or **ocs-storagecluster-ceph-rbd**.
If you select **ocs-storagecluster-ceph-rbd**, you must set the **Volume Mode** of the disk to **Block**.
 - **Advanced** → **Volume Mode:** Select **Block**.
 - **Advanced** → **Volume Mode:** Select **Block**.
 - **Networking** → **Network:** You can select a network from a list of available network attachment definition objects.
11. Click **Import** or **Review and Import**, if you have edited the import settings.
A **Successfully created virtual machine** message and a list of resources created for the virtual machine are displayed. The virtual machine appears in **Workloads** → **Virtual Machines**.

Virtual machine wizard fields

Name	Parameter	Description
Name		The name can contain lowercase letters (a-z), numbers (0-9), and hyphens (-), up to a maximum of 253 characters. The first and last characters must be alphanumeric. The name must not contain uppercase letters, spaces, periods (.), or special characters.

Name	Parameter	Description
Description		Optional description field.
Operating System		The operating system that is selected for the virtual machine in the template. You cannot edit this field when creating a virtual machine from a template.
Boot Source	Import via URL (creates PVC)	Import content from an image available from an HTTP or HTTPS endpoint. Example: Obtaining a URL link from the web page with the operating system image.
	Clone existing PVC (creates PVC)	Select an existent persistent volume claim available on the cluster and clone it.
	Import via Registry (creates PVC)	Provision virtual machine from a bootable operating system container located in a registry accessible from the cluster. Example: kubevirt/cirros-registry-disk-demo .
	PXE (network boot - adds network interface)	Boot an operating system from a server on the network. Requires a PXE bootable network attachment definition.
Persistent Volume Claim project		Project name that you want to use for cloning the PVC.
Persistent Volume Claim name		PVC name that should apply to this virtual machine template if you are cloning an existing PVC.
Mount this as a CD-ROM boot source		A CD-ROM requires an additional disk for installing the operating system. Select the checkbox to add a disk and customize it later.

Name	Description
Network	List of available network attachment definitions.
Type	List of available binding methods. For the default pod network, masquerade is the only recommended binding method. For secondary networks, use the bridge binding method. The masquerade method is not supported for non-default networks. Select SR-IOV if you configured an SR-IOV network device and defined that network in the namespace.
MAC Address	MAC address for the network interface controller. If a MAC address is not specified, one is assigned automatically.

Storage fields

Name	Selection	Description
Source	Blank (creates PVC)	Create an empty disk.
	Import via URL (creates PVC)	Import content via URL (HTTP or HTTPS endpoint).
	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	Selection	Description
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 → Volume Mode		Defines whether the persistent volume uses a formatted file system or raw block state. Default is Filesystem .

Advanced storage settings

Name	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 ^[1]	Single User (RWO)	The disk can be mounted as read/write by a single node.
	Shared Access (RWX)	The disk can be mounted as read/write by many nodes.
	Read Only (ROX)	The disk can be mounted as read-only by many nodes.

1. You can change the access mode by using the command line interface.

8.15.4.4. Importing a virtual machine with the CLI

You can import a virtual machine with the CLI by creating the **Secret** and **VirtualMachineImport** custom resources (CRs). The **Secret** CR stores the RHV Manager credentials and CA certificate. The **VirtualMachineImport** CR defines the parameters of the VM import process.

Optional: You can create a **ResourceMapping** CR that is separate from the **VirtualMachineImport** CR. A **ResourceMapping** CR provides greater flexibility, for example, if you import additional RHV VMs.



IMPORTANT

The default target storage class must be NFS. Cinder does not support RHV VM import.

Procedure

1. Create the **Secret** CR by running the following command:

```
$ cat <<EOF | oc create -f -
apiVersion: v1
kind: Secret
metadata:
  name: rhv-credentials
  namespace: default 1
type: Opaque
stringData:
  ovirt: |
    apiUrl: <api_endpoint> 2
    username: ocpadmin@internal
    password: 3
    caCert: |
      -----BEGIN CERTIFICATE-----
      4
      -----END CERTIFICATE-----
EOF
```

- 1** Optional. You can specify a different namespace in all the CRs.
- 2** Specify the API endpoint of the RHV Manager, for example, `"https://www.example.com:8443/ovirt-engine/api"`
- 3** Specify the password for `ocpadmin@internal`.
- 4** Specify the RHV Manager CA certificate. You can obtain the CA certificate by running the following command:

```
$ openssl s_client -connect :443 -showcerts < /dev/null
```

2. Optional: Create a **ResourceMapping** CR if you want to separate the resource mapping from the **VirtualMachineImport** CR by running the following command:

```
$ cat <<EOF | kubectl create -f -
```

```

apiVersion: v2v.kubevirt.io/v1alpha1
kind: ResourceMapping
metadata:
  name: resourcemapping_example
  namespace: default
spec:
  ovirt:
    networkMappings:
      - source:
          name: <rhv_logical_network>/<vnic_profile> ❶
        target:
          name: <target_network> ❷
        type: pod
    storageMappings: ❸
      - source:
          name: <rhv_storage_domain> ❹
        target:
          name: <target_storage_class> ❺
        volumeMode: <volume_mode> ❻
EOF

```

- ❶ Specify the RHV logical network and vNIC profile.
- ❷ Specify the OpenShift Virtualization network.
- ❸ If storage mappings are specified in both the **ResourceMapping** and the **VirtualMachineImport** CRs, the **VirtualMachineImport** CR takes precedence.
- ❹ Specify the RHV storage domain.
- ❺ Specify **nfs** or **ocs-storagecluster-ceph-rbd**.
- ❻ If you specified the **ocs-storagecluster-ceph-rbd** storage class, you must specify **Block** as the volume mode.

3. Create the **VirtualMachineImport** CR by running the following command:

```

$ cat <<EOF | oc create -f -
apiVersion: v2v.kubevirt.io/v1beta1
kind: VirtualMachineImport
metadata:
  name: vm-import
  namespace: default
spec:
  providerCredentialsSecret:
    name: rhv-credentials
    namespace: default
  # resourceMapping: ❶
  # name: resourcemapping-example
  # namespace: default
  targetVmName: vm_example ❷
  startVm: true
  source:
    ovirt:

```

```

vm:
  id: <source_vm_id> 3
  name: <source_vm_name> 4
cluster:
  name: <source_cluster_name> 5
mappings: 6
networkMappings:
  - source:
    name: <source_logical_network>/<vnic_profile> 7
  target:
    name: <target_network> 8
  type: pod
storageMappings: 9
  - source:
    name: <source_storage_domain> 10
  target:
    name: <target_storage_class> 11
  accessMode: <volume_access_mode> 12
diskMappings:
  - source:
    id: <source_vm_disk_id> 13
  target:
    name: <target_storage_class> 14

```

EOF

- 1 If you create a **ResourceMapping** CR, uncomment the **resourceMapping** section.
- 2 Specify the target VM name.
- 3 Specify the source VM ID, for example, **80554327-0569-496b-bdeb-fcbbf52b827b**. You can obtain the VM ID by entering <https://www.example.com/ovirt-engine/api/vms/> in a web browser on the Manager machine to list all VMs. Locate the VM you want to import and its corresponding VM ID. You do not need to specify a VM name or cluster name.
- 4 If you specify the source VM name, you must also specify the source cluster. Do not specify the source VM ID.
- 5 If you specify the source cluster, you must also specify the source VM name. Do not specify the source VM ID.
- 6 If you create a **ResourceMapping** CR, comment out the **mappings** section.
- 7 Specify the logical network and vNIC profile of the source VM.
- 8 Specify the OpenShift Virtualization network.
- 9 If storage mappings are specified in both the **ResourceMapping** and the **VirtualMachineImport** CRs, the **VirtualMachineImport** CR takes precedence.
- 10 Specify the source storage domain.
- 11 Specify the target storage class.
- 12 Specify **ReadWriteOnce**, **ReadWriteMany**, or **ReadOnlyMany**. If no access mode is

- If the RHV VM migration mode is **Allow manual and automatic migration**, the default access mode is **ReadWriteMany**.
 - If the RHV virtual disk access mode is **ReadOnly**, the default access mode is **ReadOnlyMany**.
 - For all other settings, the default access mode is **ReadWriteOnce**.
- 13** Specify the source VM disk ID, for example, **8181ecc1-5db8-4193-9c92-3ddab3be7b05**. You can obtain the disk ID by entering **<https://www.example.com/ovirt-engine/api/vms/vm23>** in a web browser on the Manager machine and reviewing the VM details.
- 14** Specify the target storage class.
4. Follow the progress of the virtual machine import to verify that the import was successful:

```
$ oc get vmimports vm-import -n default
```

The output indicating a successful import resembles the following example:

Example output

```
...
status:
conditions:
- lastHeartbeatTime: "2020-07-22T08:58:52Z"
  lastTransitionTime: "2020-07-22T08:58:52Z"
  message: Validation completed successfully
  reason: ValidationCompleted
  status: "True"
  type: Valid
- lastHeartbeatTime: "2020-07-22T08:58:52Z"
  lastTransitionTime: "2020-07-22T08:58:52Z"
  message: 'VM specifies IO Threads: 1, VM has NUMA tune mode specified: interleave'
  reason: MappingRulesVerificationReportedWarnings
  status: "True"
  type: MappingRulesVerified
- lastHeartbeatTime: "2020-07-22T08:58:56Z"
  lastTransitionTime: "2020-07-22T08:58:52Z"
  message: Copying virtual machine disks
  reason: CopyingDisks
  status: "True"
  type: Processing
dataVolumes:
- name: fedora32-b870c429-11e0-4630-b3df-21da551a48c0
  targetVmName: fedora32
```

8.15.4.4.1. Creating a config map for importing a VM

You can create a config map to map the Red Hat Virtualization (RHV) virtual machine operating system to an OpenShift Virtualization template if you want to override the default **vm-import-controller** mapping or to add additional mappings.

The default **vm-import-controller** config map contains the following RHV operating systems and their corresponding common OpenShift Virtualization templates.

Table 8.5. Operating system and template mapping

RHV VM operating system	OpenShift Virtualization template
rhel_6_10_plus_ppc64	rhel6.10
rhel_6_ppc64	rhel6.10
rhel_6	rhel6.10
rhel_6x64	rhel6.10
rhel_6_9_plus_ppc64	rhel6.9
rhel_7_ppc64	rhel7.7
rhel_7_s390x	rhel7.7
rhel_7x64	rhel7.7
rhel_8x64	rhel8.1
sles_11_ppc64	opensuse15.0
sles_11	opensuse15.0
sles_12_s390x	opensuse15.0
ubuntu_12_04	ubuntu18.04
ubuntu_12_10	ubuntu18.04
ubuntu_13_04	ubuntu18.04
ubuntu_13_10	ubuntu18.04
ubuntu_14_04_ppc64	ubuntu18.04
ubuntu_14_04	ubuntu18.04
ubuntu_16_04_s390x	ubuntu18.04
windows_10	win10

RHV VM operating system	OpenShift Virtualization template
windows_10x64	win10
windows_2003	win10
windows_2003x64	win10
windows_2008R2x64	win2k8
windows_2008	win2k8
windows_2008x64	win2k8
windows_2012R2x64	win2k12r2
windows_2012x64	win2k12r2
windows_2016x64	win2k16
windows_2019x64	win2k19
windows_7	win10
windows_7x64	win10
windows_8	win10
windows_8x64	win10
windows_xp	win10

Procedure

1. In a web browser, identify the REST API name of the RHV VM operating system by navigating to http://<RHV_Manager_FQDN>/ovirt-engine/api/vms/<VM_ID>. The operating system name appears in the `<os>` section of the XML output, as shown in the following example:

```
...
<os>
...
<type>rhel_8x64</type>
</os>
```

2. View a list of the available OpenShift Virtualization templates:

```
$ oc get templates -n openshift --show-labels | tr ',' '\n' | grep os.template.kubevirt.io | sed -r 's#os.template.kubevirt.io/(.*)=.*#\1#g' | sort -u
```

Example output

```
fedora31
fedora32
...
rhel8.1
rhel8.2
...
```

3. If an OpenShift Virtualization template that matches the RHV VM operating system does not appear in the list of available templates, create a template with the OpenShift Virtualization web console.
4. Create a config map to map the RHV VM operating system to the OpenShift Virtualization template:

```
$ cat <<EOF | oc create -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: os-configmap
  namespace: default 1
data:
  guestos2common: |
    "Red Hat Enterprise Linux Server": "rhel"
    "CentOS Linux": "centos"
    "Fedora": "fedora"
    "Ubuntu": "ubuntu"
    "openSUSE": "opensuse"
  osinfo2common: |
    "<rhv-operating-system>": "<vm-template>" 2
EOF
```

- 1** Optional: You can change the value of the **namespace** parameter.
- 2** Specify the REST API name of the RHV operating system and its corresponding VM template as shown in the following example.

Config map example

```
$ cat <<EOF | oc apply -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: os-configmap
  namespace: default
data:
  osinfo2common: |
    "other_linux": "fedora31"
EOF
```

5. Verify that the custom config map was created:

```
$ oc get cm -n default os-configmap -o yaml
```

6. Patch the **vm-import-controller-config** config map to apply the new config map:

```
$ oc patch configmap vm-import-controller-config -n openshift-cnv --patch '{
  "data": {
    "osConfigMap.name": "os-configmap",
    "osConfigMap.namespace": "default" 1
  }
}'
```

- 1 Update the namespace if you changed it in the config map.

7. Verify that the template appears in the OpenShift Virtualization web console:
 - a. Click **Workloads** → **Virtualization** from the side menu.
 - b. Click the **Virtual Machine Templates** tab and find the template in the list.

8.15.4.5. Troubleshooting a virtual machine import

8.15.4.5.1. Logs

You can check the VM Import Controller pod log for errors.

Procedure

1. View the VM Import Controller pod name by running the following command:

```
$ oc get pods -n <namespace> | grep import 1
```

- 1 Specify the namespace of your imported virtual machine.

Example output

```
vm-import-controller-f66f7d-zqkz7      1/1   Running    0      4h49m
```

2. View the VM Import Controller pod log by running the following command:

```
$ oc logs <vm-import-controller-f66f7d-zqkz7> -f -n <namespace> 1
```

- 1 Specify the VM Import Controller pod name and the namespace.

8.15.4.5.2. Error messages

The following error message might appear:

- The following error message is displayed in the VM Import Controller pod log and the progress bar stops at 10% if the OpenShift Virtualization storage PV is not suitable:

Failed to bind volumes: provisioning failed for PVC

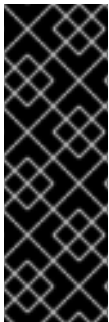
You must use a compatible storage class. The Cinder storage class is not supported.

8.15.4.5.3. Known issues

- If you are using Ceph RBD block-mode volumes and the available storage space is too small for the virtual disk, the import process bar stops at 75% for more than 20 minutes and the migration does not succeed. No error message is displayed in the web console. [BZ#1910019](#)

8.15.5. Importing a single VMware virtual machine or template

You can import a VMware vSphere 6.5, 6.7, or 7.0 VM or VM template into OpenShift Virtualization by using the VM Import wizard. If you import a VM template, OpenShift Virtualization creates a virtual machine based on the template.



IMPORTANT

Importing a VMware VM is a deprecated feature. Deprecated functionality is still included in OpenShift Virtualization and continues to be supported; however, it will be removed in a future release of this product and is not recommended for new deployments.

For the most recent list of major functionality that has been deprecated or removed within OpenShift Virtualization, refer to the *Deprecated and removed features* section of the OpenShift Virtualization release notes.

This feature will be replaced by the [Migration Toolkit for Virtualization](#).

8.15.5.1. OpenShift Virtualization storage feature matrix

The following table describes the OpenShift Virtualization storage types that support VM import.

Table 8.6. OpenShift Virtualization storage feature matrix

VMware VM import	
OpenShift Container Storage: RBD block-mode volumes	Yes
OpenShift Virtualization hostpath provisioner	Yes
Other multi-node writable storage	Yes ^[1]
Other single-node writable storage	Yes ^[2]

1. PVCs must request a ReadWriteMany access mode.
2. PVCs must request a ReadWriteOnce access mode.

8.15.5.2. Preparing a VDDK image

The import process uses the VMware Virtual Disk Development Kit (VDDK) to copy the VMware virtual disk.

You can download the VDDK SDK, create a VDDK image, upload the image to an image registry, and add it to the **spec.vddkInitImage** field of the **HyperConverged** custom resource (CR).

You can configure either an internal OpenShift Container Platform image registry or a secure external image registry for the VDDK image. The registry must be accessible to your OpenShift Virtualization environment.



NOTE

Storing the VDDK image in a public registry might violate the terms of the VMware license.

8.15.5.2.1. Configuring an internal image registry

You can configure the internal OpenShift Container Platform image registry on bare metal by updating the Image Registry Operator configuration.

You can access the registry directly, from within the OpenShift Container Platform cluster, or externally, by exposing the registry with a route.

Changing the image registry's management state

To start the image registry, you must change the Image Registry Operator configuration's **managementState** from **Removed** to **Managed**.

Procedure

- Change **managementState** Image Registry Operator configuration from **Removed** to **Managed**. For example:

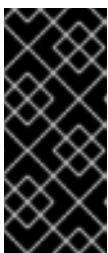
```
$ oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec": {"managementState": "Managed"}}'
```

Configuring registry storage for bare metal and other manual installations

As a cluster administrator, following installation you must configure your registry to use storage.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.
- You have a cluster that uses manually-provisioned Red Hat Enterprise Linux CoreOS (RHCOS) nodes, such as bare metal.
- You have provisioned persistent storage for your cluster, such as Red Hat OpenShift Container Storage.



IMPORTANT

OpenShift Container Platform supports **ReadWriteOnce** access for image registry storage when you have only one replica. **ReadWriteOnce** access also requires that the registry uses the **Recreate** rollout strategy. To deploy an image registry that supports high availability with two or more replicas, **ReadWriteMany** access is required.

- Must have 100Gi capacity.

Procedure

1. To configure your registry to use storage, change the **spec.storage.pvc** in the **configs.imageregistry/cluster** resource.



NOTE

When using shared storage, review your security settings to prevent outside access.

2. Verify that you do not have a registry pod:

```
$ oc get pod -n openshift-image-registry -l docker-registry=default
```

Example output

```
No resources found in openshift-image-registry namespace
```



NOTE

If you do have a registry pod in your output, you do not need to continue with this procedure.

3. Check the registry configuration:

```
$ oc edit configs.imageregistry.operator.openshift.io
```

Example output

```
storage:
  pvc:
  claim:
```

Leave the **claim** field blank to allow the automatic creation of an **image-registry-storage** PVC.

4. Check the **clusteroperator** status:

```
$ oc get clusteroperator image-registry
```

Example output

NAME	VERSION	AVAILABLE	PROGRESSING	DEGRADED
image-registry	4.7	True	False	False

5. Ensure that your registry is set to managed to enable building and pushing of images.

- Run:

```
$ oc edit configs.imageregistry/cluster
```

Then, change the line

```
managementState: Removed
```

to

```
managementState: Managed
```

Accessing registry directly from the cluster

You can access the registry from inside the cluster.

Procedure

Access the registry from the cluster by using internal routes:

1. Access the node by getting the node's name:

```
$ oc get nodes
```

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

2. To enable access to tools such as **oc** and **podman** on the node, change your root directory to **/host**:

```
sh-4.2# chroot /host
```

3. Log in to the container image registry by using your access token:

```
sh-4.2# oc login -u kubeadmin -p <password_from_install_log> https://api-int.
<cluster_name>.<base_domain>:6443
```

```
sh-4.2# podman login -u kubeadmin -p $(oc whoami -t) image-registry.openshift-image-
registry.svc:5000
```

You should see a message confirming login, such as:

```
Login Succeeded!
```



NOTE

You can pass any value for the user name; the token contains all necessary information. Passing a user name that contains colons will result in a login failure.

Since the Image Registry Operator creates the route, it will likely be similar to **default-route-openshift-image-registry.<cluster_name>**.

4. Perform **podman pull** and **podman push** operations against your registry:



IMPORTANT

You can pull arbitrary images, but if you have the **system:registry** role added, you can only push images to the registry in your project.

In the following examples, use:

Component	Value
<registry_ip>	172.30.124.220
<port>	5000
<project>	openshift
<image>	image
<tag>	omitted (defaults to latest)

- a. Pull an arbitrary image:

```
sh-4.2# podman pull name.io/image
```

- b. Tag the new image with the form **<registry_ip>:<port>/<project>/<image>**. The project name must appear in this pull specification for OpenShift Container Platform to correctly place and later access the image in the registry:

```
sh-4.2# podman tag name.io/image image-registry.openshift-image-registry.svc:5000/openshift/image
```



NOTE

You must have the **system:image-builder** role for the specified project, which allows the user to write or push an image. Otherwise, the **podman push** in the next step will fail. To test, you can create a new project to push the image.

- c. Push the newly tagged image to your registry:

```
sh-4.2# podman push image-registry.openshift-image-registry.svc:5000/openshift/image
```

Exposing a secure registry manually

Instead of logging in to the OpenShift Container Platform registry from within the cluster, you can gain external access to it by exposing it with a route. This allows you to log in to the registry from outside the cluster using the route address, and to tag and push images to an existing project by using the route host.

Prerequisites:

- The following prerequisites are automatically performed:

- Deploy the Registry Operator.
- Deploy the Ingress Operator.

Procedure

You can expose the route by using **DefaultRoute** parameter in the **configs.imageregistry.operator.openshift.io** resource or by using custom routes.

To expose the registry using **DefaultRoute**:

1. Set **DefaultRoute** to **True**:

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster --patch '{"spec": {"defaultRoute":true}}' --type=merge
```

2. Log in with **podman**:

```
$ HOST=$(oc get route default-route -n openshift-image-registry --template='{{ .spec.host }}')
```

```
$ podman login -u kubeadmin -p $(oc whoami -t) --tls-verify=false $HOST 1
```

- 1** **--tls-verify=false** is needed if the cluster's default certificate for routes is untrusted. You can set a custom, trusted certificate as the default certificate with the Ingress Operator.

To expose the registry using custom routes:

1. Create a secret with your route's TLS keys:

```
$ oc create secret tls public-route-tls \
  -n openshift-image-registry \
  --cert=</path/to/tls.crt> \
  --key=</path/to/tls.key>
```

This step is optional. If you do not create a secret, the route uses the default TLS configuration from the Ingress Operator.

2. On the Registry Operator:

```
spec:
  routes:
    - name: public-routes
      hostname: myregistry.mycorp.organization
      secretName: public-route-tls
  ...
```



NOTE

Only set **secretName** if you are providing a custom TLS configuration for the registry's route.

8.15.5.2.2. Configuring an external image registry

If you use an external image registry for the VDDK image, you can add the external image registry's certificate authorities to the OpenShift Container Platform cluster.

Optionally, you can create a pull secret from your Docker credentials and add it to your service account.

Adding certificate authorities to the cluster

You can add certificate authorities (CA) to the cluster for use when pushing and pulling images with the following procedure.

Prerequisites

- You must have cluster administrator privileges.
- You must have access to the public certificates of the registry, usually a **hostname/ca.crt** file located in the **/etc/docker/certs.d/** directory.

Procedure

1. Create a **ConfigMap** in the **openshift-config** namespace containing the trusted certificates for the registries that use self-signed certificates. For each CA file, ensure the key in the **ConfigMap** is the hostname of the registry in the **hostname[..port]** format:

```
$ oc create configmap registry-cas -n openshift-config \
  --from-file=myregistry.corp.com..5000=/etc/docker/certs.d/myregistry.corp.com:5000/ca.crt \
  --from-file=otherregistry.com=/etc/docker/certs.d/otherregistry.com/ca.crt
```

2. Update the cluster image configuration:

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
{"name":"registry-cas"}}}' --type=merge
```

Allowing pods to reference images from other secured registries

The **.dockercfg** **\$HOME/.docker/config.json** file for Docker clients is a Docker credentials file that stores your authentication information if you have previously logged into a secured or insecure registry.

To pull a secured container image that is not from OpenShift Container Platform's internal registry, you must create a pull secret from your Docker credentials and add it to your service account.

Procedure

- If you already have a **.dockercfg** file for the secured registry, you can create a secret from that file by running:

```
$ oc create secret generic <pull_secret_name> \
  --from-file=.dockercfg=<path/to/.dockercfg> \
  --type=kubernetes.io/dockercfg
```

- Or if you have a **\$HOME/.docker/config.json** file:

```
$ oc create secret generic <pull_secret_name> \
  --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
  --type=kubernetes.io/dockerconfigjson
```

- If you do not already have a Docker credentials file for the secured registry, you can create a secret by running:

```
$ oc create secret docker-registry <pull_secret_name> \
  --docker-server=<registry_server> \
  --docker-username=<user_name> \
  --docker-password=<password> \
  --docker-email=<email>
```

- To use a secret for pulling images for pods, you must add the secret to your service account. The name of the service account in this example should match the name of the service account the pod uses. The default service account is **default**:

```
$ oc secrets link default <pull_secret_name> --for=pull
```

8.15.5.2.3. Creating and using a VDDK image

You can download the VMware Virtual Disk Development Kit (VDDK), build a VDDK image, and push the VDDK image to your image registry. You then add the VDDK image to the **spec.vddkInitImage** field of the **HyperConverged** custom resource (CR).

Prerequisites

- You must have access to an OpenShift Container Platform internal image registry or a secure external registry.

Procedure

1. Create and navigate to a temporary directory:

```
$ mkdir /tmp/<dir_name> && cd /tmp/<dir_name>
```

2. In a browser, navigate to [VMware code](#) and click **SDKs**.
3. Under **Compute Virtualization**, click **Virtual Disk Development Kit (VDDK)**
4. Select the VDDK version that corresponds to your VMware vSphere version, for example, VDDK 7.0 for vSphere 7.0, click **Download**, and then save the VDDK archive in the temporary directory.
5. Extract the VDDK archive:

```
$ tar -xzf VMware-vix-disklib-<version>.x86_64.tar.gz
```

6. Create a **Dockerfile**:

```
$ cat > Dockerfile <<EOF
FROM busybox:latest
COPY vmware-vix-disklib-distrib /vmware-vix-disklib-distrib
RUN mkdir -p /opt
ENTRYPOINT ["cp", "-r", "/vmware-vix-disklib-distrib", "/opt"]
EOF
```

7. Build the image:


```
$ podman build . -t <registry_route_or_server_path>/vddk:<tag> 1
```

1 Specify your image registry:

- For an internal OpenShift Container Platform registry, use the internal registry route, for example, **image-registry.openshift-image-registry.svc:5000/openshift/vddk:<tag>**.
- For an external registry, specify the server name, path, and tag, for example, **server.example.com:5000/vddk:<tag>**.

8. Push the image to the registry:

```
$ podman push <registry_route_or_server_path>/vddk:<tag>
```

9. Ensure that the image is accessible to your OpenShift Virtualization environment.

10. Edit the **HyperConverged** CR in the **openshift-cnv** project:

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

11. Add the **vddkInitImage** parameter to the **spec** stanza:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  vddkInitImage: <registry_route_or_server_path>/vddk:<tag>
```

8.15.5.3. Importing a virtual machine with the VM Import wizard

You can import a single virtual machine with the VM Import wizard.

You can also import a VM template. If you import a VM template, OpenShift Virtualization creates a virtual machine based on the template.

Prerequisites

- You must have admin user privileges.
- The VMware Virtual Disk Development Kit (VDDK) image must be in an image registry that is accessible to your OpenShift Virtualization environment.
- The VDDK image must be added to the **spec.vddkInitImage** field of the **HyperConverged** custom resource (CR).
- The VM must be powered off.
- Virtual disks must be connected to IDE or SCSI controllers. If virtual disks are connected to a SATA controller, you can change them to IDE controllers and then migrate the VM.

- The OpenShift Virtualization local and shared persistent storage classes must support VM import.
- The OpenShift Virtualization storage must be large enough to accommodate the virtual disk.



WARNING

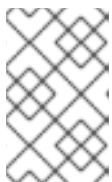
If you are using Ceph RBD block-mode volumes, the storage must be large enough to accommodate the virtual disk. If the disk is too large for the available storage, the import process fails and the PV that is used to copy the virtual disk is not released. You will not be able to import another virtual machine or to clean up the storage because there are insufficient resources to support object deletion. To resolve this situation, you must add more object storage devices to the storage back end.

- The OpenShift Virtualization egress network policy must allow the following traffic:

Destination	Protocol	Port
VMware ESXi hosts	TCP	443
VMware ESXi hosts	TCP	902
VMware vCenter	TCP	5840


Procedure

1. In the web console, click **Workloads → Virtual Machines**.
2. Click **Create Virtual Machine** and select **Import with Wizard**.
3. Select **VMware** from the **Provider** list.
4. Select **Connect to New Instance** or a saved vCenter instance.
 - If you select **Connect to New Instance**, enter the **vCenter hostname**, **Username**, and **Password**.
 - If you select a saved vCenter instance, the wizard connects to the vCenter instance using the saved credentials.
5. Click **Check and Save** and wait for the connection to complete.



NOTE

The connection details are stored in a secret. If you add a provider with an incorrect hostname, user name, or password, click **Workloads → Secrets** and delete the provider secret.

6. Select a virtual machine or a template.
7. Click **Next**.
8. In the **Review** screen, review your settings.
9. Click **Edit** to update the following settings:
 - **General:**
 - **Description**
 - **Operating System**
 - **Flavor**
 - **Memory**
 - **CPUs**
 - **Workload Profile**
 - **Networking:**
 - **Name**
 - **Model**
 - **Network**
 - **Type**
 - **MAC Address**
 - **Storage:** Click the Options menu  of the VM disk and select **Edit** to update the following fields:
 - **Name**
 - **Source:** For example, **Import Disk**.
 - **Size**
 - **Interface**
 - **Storage Class:** Select **NFS** or **ocs-storagecluster-ceph-rbd (ceph-rbd)**.
If you select **ocs-storagecluster-ceph-rbd**, you must set the **Volume Mode** of the disk to **Block**.

Other storage classes might work, but they are not officially supported.
 - **Advanced → Volume Mode:** Select **Block**.
 - **Advanced → Access Mode**
 - **Advanced → Cloud-init**

- **Form:** Enter the **Hostname** and **Authenticated SSH Keys**.
- **Custom script** Enter the **cloud-init** script in the text field.
- **Advanced → Virtual Hardware:** You can attach a virtual CD-ROM to the imported virtual machine.

10. Click **Import** or **Review and Import**, if you have edited the import settings.

A **Successfully created virtual machine** message and a list of resources created for the virtual machine are displayed. The virtual machine appears in **Workloads → Virtual Machines**.

Virtual machine wizard fields

Name	Parameter	Description
Name		The name can contain lowercase letters (a-z), numbers (0-9), and hyphens (-), up to a maximum of 253 characters. The first and last characters must be alphanumeric. The name must not contain uppercase letters, spaces, periods (.), or special characters.
Description		Optional description field.
Operating System		The operating system that is selected for the virtual machine in the template. You cannot edit this field when creating a virtual machine from a template.
Boot Source	Import via URL (creates PVC)	Import content from an image available from an HTTP or HTTPS endpoint. Example: Obtaining a URL link from the web page with the operating system image.
	Clone existing PVC (creates PVC)	Select an existent persistent volume claim available on the cluster and clone it.
	Import via Registry (creates PVC)	Provision virtual machine from a bootable operating system container located in a registry accessible from the cluster. Example: kubevirt/cirros-registry-disk-demo .
	PXE (network boot - adds network interface)	Boot an operating system from a server on the network. Requires a PXE bootable network attachment definition.

Name	Parameter	Description
Persistent Volume Claim project		Project name that you want to use for cloning the PVC.
Persistent Volume Claim name		PVC name that should apply to this virtual machine template if you are cloning an existing PVC.
Mount this as a CD-ROM boot source		A CD-ROM requires an additional disk for installing the operating system. Select the checkbox to add a disk and customize it later.
Flavor	Tiny, Small, Medium, Large, Custom	<p>Presets the amount of CPU and memory in a virtual machine template with predefined values that are allocated to the virtual machine, depending on the operating system associated with that template.</p> <p>If you choose a default template, you can override the cpus and memsize values in the template using custom values to create a custom template. Alternatively, you can create a custom template by modifying the cpus and memsize values in the Details tab on the Workloads → Virtualization page.</p>
<p>Workload Type</p>  <p>NOTE</p> <p>If you choose the incorrect Workload Type, there could be performance or resource utilization issues (such as a slow UI).</p>	<p>Desktop</p> <hr/> <p>Server</p> <hr/> <p>High-Performance</p>	<p>A virtual machine configuration for use on a desktop. Ideal for consumption on a small scale. Recommended for use with the web console.</p> <hr/> <p>Balances performance and it is compatible with a wide range of server workloads.</p> <hr/> <p>A virtual machine configuration that is optimized for high-performance workloads.</p>

Name	Parameter	Description
Start this virtual machine after creation.		This checkbox is selected by default and the virtual machine starts running after creation. Clear the checkbox if you do not want the virtual machine to start when it is created.

Cloud-init fields

Name	Description
Hostname	Sets a specific hostname for the virtual machine.
Authorized SSH Keys	The user's public key that is copied to <code>~/.ssh/authorized_keys</code> on the virtual machine.
Custom script	Replaces other options with a field in which you paste a custom cloud-init script.

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 .
Network	List of available network attachment definitions.
Type	List of available binding methods. For the default pod network, masquerade is the only recommended binding method. For secondary networks, use the bridge binding method. The masquerade method is not supported for non-default networks. Select SR-IOV if you configured an SR-IOV network device and defined that network in the namespace.
MAC Address	MAC address for the network interface controller. If a MAC address is not specified, one is assigned automatically.


Storage fields

Name	Selection	Description
Source	Blank (creates PVC)	Create an empty disk.
	Import via URL (creates PVC)	Import content via URL (HTTP or HTTPS endpoint).
	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 → Volume Mode		Defines whether the persistent volume uses a formatted file system or raw block state. Default is Filesystem .

Name	Selection	Description
Advanced → Access Mode		Access mode of the persistent volume. Supported access modes are Single User (RWO) , Shared Access (RWX) , and Read Only (ROX) .

Advanced storage settings

The following advanced storage settings are available for **Blank**, **Import via URL**, and **Clone existing PVC** disks. These parameters are optional. If you do not specify these parameters, the system uses the default values from the **kubevirt-storage-class-defaults** config map.

Name	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	Single User (RWO)	The disk can be mounted as read/write by a single node.
	Shared Access (RWX)	The disk can be mounted as read/write by many nodes.  NOTE This is required for some features, such as live migration of virtual machines between nodes.
	Read Only (ROX)	The disk can be mounted as read-only by many nodes.

8.15.5.3.1. Updating the NIC name of an imported virtual machine

You must update the NIC name of a virtual machine that you imported from VMware to conform to OpenShift Virtualization naming conventions.

Procedure

1. Log in to the virtual machine.
2. Navigate to the **/etc/sysconfig/network-scripts** directory.
3. Rename the network configuration file:


```
$ mv vmnic0 ifcfg-eth0 1
```

- 1 The first network configuration file is named **ifcfg-eth0**. Additional network configuration files are numbered sequentially, for example, **ifcfg-eth1**, **ifcfg-eth2**.

4. Update the **NAME** and **DEVICE** parameters in the network configuration file:

```
NAME=eth0
DEVICE=eth0
```

5. Restart the network:

```
$ systemctl restart network
```

8.15.5.4. Troubleshooting a virtual machine import

8.15.5.4.1. Logs

You can check the V2V Conversion pod log for errors.

Procedure

1. View the V2V Conversion pod name by running the following command:

```
$ oc get pods -n <namespace> | grep v2v 1
```

- 1 Specify the namespace of your imported virtual machine.

Example output

```
kubevirt-v2v-conversion-f66f7d-zqkz7      1/1   Running   0      4h49m
```

2. View the V2V Conversion pod log by running the following command:

```
$ oc logs <kubevirt-v2v-conversion-f66f7d-zqkz7> -f -n <namespace> 1
```

- 1 Specify the VM Conversion pod name and the namespace.

8.15.5.4.2. Error messages

The following error messages might appear:

- If the VMware VM is not shut down before import, the imported virtual machine displays the error message, **Readiness probe failed** in the OpenShift Container Platform console and the V2V Conversion pod log displays the following error message:

```
INFO - have error: ('virt-v2v error: internal error: invalid argument: libvirt domain
'v2v_migration_vm_1' is running or paused. It must be shut down in order to perform virt-v2v
conversion',)
```

- The following error message is displayed in the OpenShift Container Platform console if a non-admin user tries to import a VM:

```
Could not load config map vmware-to-kubevirt-os in kube-public namespace
Restricted Access: configmaps "vmware-to-kubevirt-os" is forbidden: User cannot get
resource "configmaps" in API group "" in the namespace "kube-public"
```

Only an admin user can import a VM.

8.16. CLONING VIRTUAL MACHINES

8.16.1. 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.

8.16.1.1. Prerequisites

- Only a user with the **cluster-admin** role can create cluster roles.

8.16.1.2. About data volumes

DataVolume objects are custom resources that are provided by the Containerized Data Importer (CDI) project. Data volumes orchestrate import, clone, and upload operations that are associated with an underlying persistent volume claim (PVC). Data volumes are integrated with OpenShift Virtualization, and they prevent a virtual machine from being started before the PVC has been prepared.

8.16.1.3. Creating RBAC resources for cloning data volumes

Create a new cluster role that enables permissions for all actions for the **datavolumes** resource.

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.

8.16.2. Cloning a virtual machine disk into a new data volume

You can clone the persistent volume claim (PVC) of a virtual machine disk into a new data volume by referencing the source PVC in your data volume configuration file.



WARNING

Cloning operations between different volume modes are supported, such as cloning from a persistent volume (PV) with **volumeMode: Block** to a PV with **volumeMode: Filesystem**.

However, you can only clone between different volume modes if they are of the **contentType: kubevirt**.

TIP

When you enable preallocation globally, or for a single data volume, the Containerized Data Importer (CDI) preallocates disk space during cloning. Preallocation enhances write performance. For more information, see [Using preallocation for data volumes](#).

8.16.2.1. Prerequisites

- Users need [additional permissions](#) to clone the PVC of a virtual machine disk into another namespace.

8.16.2.2. About data volumes

DataVolume objects are custom resources that are provided by the Containerized Data Importer (CDI) project. Data volumes orchestrate import, clone, and upload operations that are associated with an underlying persistent volume claim (PVC). Data volumes are integrated with OpenShift Virtualization, and they prevent a virtual machine from being started before the PVC has been prepared.

8.16.2.3. Cloning the persistent volume claim of a virtual machine disk into a new data volume

You can clone a persistent volume claim (PVC) of an existing virtual machine disk into a new data volume. The new data volume can then be used for a new virtual machine.



NOTE

When a data volume is created independently of a virtual machine, the lifecycle of the data volume is independent of the virtual machine. If the virtual machine is deleted, neither the data volume nor its associated PVC is deleted.

Prerequisites

- Determine the PVC of an existing virtual machine disk to use. You must power down the virtual machine that is associated with the PVC before you can clone it.
- Install the OpenShift CLI (**oc**).

Procedure

1. Examine the virtual machine disk you want to clone to identify the name and namespace of the associated PVC.

2. Create a YAML file for a data volume that specifies the name of the new data volume, the name and namespace of the source PVC, and the size of the new data volume.

For example:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <cloner-datavolume> 1
spec:
  source:
    pvc:
      namespace: "<source-namespace>" 2
      name: "<my-favorite-vm-disk>" 3
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi> 4
```

- 1 The name of the new data volume.
- 2 The namespace where the source PVC exists.
- 3 The name of the source PVC.
- 4 The size of the new data volume. You must allocate enough space, or the cloning operation fails. The size must be the same as or larger than the source PVC.

3. Start cloning the PVC by creating the data volume:

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



NOTE

Data volumes prevent a virtual machine from starting before the PVC is prepared, so you can create a virtual machine that references the new data volume while the PVC clones.

8.16.2.4. Template: Data volume clone configuration file

example-clone-dv.yaml

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: "example-clone-dv"
spec:
  source:
    pvc:
      name: source-pvc
      namespace: example-ns
  pvc:
```

```

accessModes:
- ReadWriteOnce
resources:
requests:
storage: "1G"

```

8.16.2.5. 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)	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2** ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

Unsupported operation

* Requires scratch space

** Requires scratch space if a custom certificate authority is required

8.16.3. Cloning a virtual machine by using a data volume template

You can create a new virtual machine by cloning the persistent volume claim (PVC) of an existing VM. By including a **dataVolumeTemplate** in your virtual machine configuration file, you create a new data volume from the original PVC.



WARNING

Cloning operations between different volume modes are supported, such as cloning from a persistent volume (PV) with **volumeMode: Block** to a PV with **volumeMode: Filesystem**.

However, you can only clone between different volume modes if they are of the **contentType: kubevirt**.

TIP

When you enable preallocation globally, or for a single data volume, the Containerized Data Importer (CDI) preallocates disk space during cloning. Preallocation enhances write performance. For more information, see [Using preallocation for data volumes](#).

8.16.3.1. Prerequisites

- Users need [additional permissions](#) to clone the PVC of a virtual machine disk into another namespace.

8.16.3.2. About data volumes

DataVolume objects are custom resources that are provided by the Containerized Data Importer (CDI) project. Data volumes orchestrate import, clone, and upload operations that are associated with an underlying persistent volume claim (PVC). Data volumes are integrated with OpenShift Virtualization, and they prevent a virtual machine from being started before the PVC has been prepared.

8.16.3.3. Creating a new virtual machine from a cloned persistent volume claim by using a data volume template

You can create a virtual machine that clones the persistent volume claim (PVC) of an existing virtual machine into a data volume. Reference a **dataVolumeTemplate** in the virtual machine manifest and the **source** PVC is cloned to a data volume, which is then automatically used for the creation of the virtual machine.

**NOTE**

When a data volume is created as part of the data volume template of a virtual machine, the lifecycle of the data volume is then dependent on the virtual machine. If the virtual machine is deleted, the data volume and associated PVC are also deleted.

Prerequisites

- Determine the PVC of an existing virtual machine disk to use. You must power down the virtual machine that is associated with the PVC before you can clone it.
- Install the OpenShift CLI (**oc**).

Procedure

1. Examine the virtual machine you want to clone to identify the name and namespace of the associated PVC.
2. Create a YAML file for a **VirtualMachine** object. The following virtual machine example clones **my-favorite-vm-disk**, which is located in the **source-namespace** namespace. The **2Gi** data volume called **favorite-clone** is created from **my-favorite-vm-disk**.
For example:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm-dv-clone
```

```

name: vm-dv-clone 1
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm-dv-clone
    spec:
      domain:
        devices:
          disks:
            - disk:
                bus: virtio
                name: root-disk
          resources:
            requests:
              memory: 64M
          volumes:
            - dataVolume:
                name: favorite-clone
                name: root-disk
      dataVolumeTemplates:
        - metadata:
            name: favorite-clone
          spec:
            pvc:
              accessModes:
                - ReadWriteOnce
              resources:
                requests:
                  storage: 2Gi
            source:
              pvc:
                namespace: "source-namespace"
                name: "my-favorite-vm-disk"

```

1 The virtual machine to create.

3. Create the virtual machine with the PVC-cloned data volume:

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

8.16.3.4. Template: Data volume virtual machine configuration file

example-dv-vm.yaml

```

apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: example-vm
  name: example-vm
spec:
  dataVolumeTemplates:

```



```

- metadata:
  name: example-dv
  spec:
    pvc:
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 1G
    source:
      http:
        url: "" 1
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: example-vm
    spec:
      domain:
        cpu:
          cores: 1
        devices:
          disks:
            - disk:
                bus: virtio
                name: example-dv-disk
          machine:
            type: q35
          resources:
            requests:
              memory: 1G
        terminationGracePeriodSeconds: 0
      volumes:
        - dataVolume:
            name: example-dv
            name: example-dv-disk

```

1 The **HTTP** source of the image you want to import, if applicable.

8.16.3.5. 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* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*

Content types	HTTP	HTTPS	HTTP basic auth	Registry	Upload
KubeVirt (RAW)	<input checked="" type="checkbox"/> RAW <input checked="" type="checkbox"/> GZ <input checked="" type="checkbox"/> XZ	<input checked="" type="checkbox"/> RAW <input checked="" type="checkbox"/> GZ <input checked="" type="checkbox"/> XZ	<input checked="" type="checkbox"/> RAW <input checked="" type="checkbox"/> GZ <input checked="" type="checkbox"/> XZ	<input checked="" type="checkbox"/> RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	<input checked="" type="checkbox"/> RAW* <input checked="" type="checkbox"/> GZ* <input checked="" type="checkbox"/> XZ*

Supported operation

Unsupported operation

* Requires scratch space

** Requires scratch space if a custom certificate authority is required

8.16.4. Cloning a virtual machine disk into a new block storage data volume

You can clone the persistent volume claim (PVC) of a virtual machine disk into a new block data volume by referencing the source PVC in your data volume configuration file.



WARNING

Cloning operations between different volume modes are supported, such as cloning from a persistent volume (PV) with **volumeMode: Block** to a PV with **volumeMode: Filesystem**.

However, you can only clone between different volume modes if they are of the **contentType: kubevirt**.

TIP

When you enable preallocation globally, or for a single data volume, the Containerized Data Importer (CDI) preallocates disk space during cloning. Preallocation enhances write performance. For more information, see [Using preallocation for data volumes](#).

8.16.4.1. Prerequisites

- Users need [additional permissions](#) to clone the PVC of a virtual machine disk into another namespace.

8.16.4.2. About data volumes

DataVolume objects are custom resources that are provided by the Containerized Data Importer (CDI) project. Data volumes orchestrate import, clone, and upload operations that are associated with an underlying persistent volume claim (PVC). Data volumes are integrated with OpenShift Virtualization, and they prevent a virtual machine from being started before the PVC has been prepared.

8.16.4.3. About block persistent volumes

A block persistent volume (PV) is a PV that is backed by a raw block device. These volumes do not have a file system and can provide performance benefits for virtual machines by reducing overhead.

Raw block volumes are provisioned by specifying **volumeMode: Block** in the PV and persistent volume claim (PVC) specification.

8.16.4.4. Creating a local block persistent volume

Create a local block persistent volume (PV) on a node by populating a file and mounting it as a loop device. You can then reference this loop device in a PV manifest as a **Block** volume and use it as a block device for a virtual machine image.

Procedure

1. Log in as **root** to the node on which to create the local PV. This procedure uses **node01** for its examples.
2. Create a file and populate it with null characters so that it can be used as a block device. The following example creates a file **loop10** with a size of 2Gb (20 100Mb blocks):

```
$ dd if=/dev/zero of=<loop10> bs=100M count=20
```

3. Mount the **loop10** file as a loop device.

```
$ losetup </dev/loop10>d3 <loop10> 1 2
```

- 1 File path where the loop device is mounted.
- 2 The file created in the previous step to be mounted as the loop device.

4. Create a **PersistentVolume** manifest that references the mounted loop device.

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: <local-block-pv10>
  annotations:
spec:
  local:
    path: </dev/loop10> 1
  capacity:
    storage: <2Gi>
  volumeMode: Block 2
  storageClassName: local 3
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
```

```
operator: In
values:
- <node01> 4
```

- 1 The path of the loop device on the node.
- 2 Specifies it is a block PV.
- 3 Optional: Set a storage class for the PV. If you omit it, the cluster default is used.
- 4 The node on which the block device was mounted.

5. Create the block PV.

```
# oc create -f <local-block-pv10.yaml> 1
```

- 1 The file name of the persistent volume created in the previous step.

8.16.4.5. Cloning the persistent volume claim of a virtual machine disk into a new data volume

You can clone a persistent volume claim (PVC) of an existing virtual machine disk into a new data volume. The new data volume can then be used for a new virtual machine.



NOTE

When a data volume is created independently of a virtual machine, the lifecycle of the data volume is independent of the virtual machine. If the virtual machine is deleted, neither the data volume nor its associated PVC is deleted.

Prerequisites

- Determine the PVC of an existing virtual machine disk to use. You must power down the virtual machine that is associated with the PVC before you can clone it.
- Install the OpenShift CLI (**oc**).
- At least one available block persistent volume (PV) that is the same size as or larger than the source PVC.

Procedure

1. Examine the virtual machine disk you want to clone to identify the name and namespace of the associated PVC.
2. Create a YAML file for a data volume that specifies the name of the new data volume, the name and namespace of the source PVC, **volumeMode: Block** so that an available block PV is used, and the size of the new data volume.

For example:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
```

```

name: <cloner-datavolume> ❶
spec:
  source:
    pvc:
      namespace: "<source-namespace>" ❷
      name: "<my-favorite-vm-disk>" ❸
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi> ❹
    volumeMode: Block ❺

```

- ❶ The name of the new data volume.
- ❷ The namespace where the source PVC exists.
- ❸ The name of the source PVC.
- ❹ The size of the new data volume. You must allocate enough space, or the cloning operation fails. The size must be the same as or larger than the source PVC.
- ❺ Specifies that the destination is a block PV

3. Start cloning the PVC by creating the data volume:

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



NOTE

Data volumes prevent a virtual machine from starting before the PVC is prepared, so you can create a virtual machine that references the new data volume while the PVC clones.

8.16.4.6. 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)	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2** ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

□ Unsupported operation

* Requires scratch space

** Requires scratch space if a custom certificate authority is required

8.17. VIRTUAL MACHINE NETWORKING

8.17.1. Configuring the virtual machine for 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

8.17.1.1. Configuring masquerade mode from the command line

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

- The virtual machine must be configured to use DHCP to acquire IPv4 addresses. The examples below are configured to use DHCP.

Procedure

1. Edit the **interfaces** spec of your virtual machine configuration file:

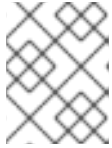
```
kind: VirtualMachine
spec:
  domain:
    devices:
      interfaces:
        - name: default
          masquerade: {} 1
        ports: 2
          - port: 80
      networks:
        - name: default
          pod: {}
```

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
```

8.17.1.2. Configuring masquerade mode with dual-stack (IPv4 and IPv6)

You can configure a new virtual machine to use both IPv6 and IPv4 on the default pod network by using cloud-init.

The IPv6 network address must be statically set to **fd10:0:2::2/120** with a default gateway of **fd10:0:2::1** in the virtual machine configuration. These are used by the virt-launcher pod to route IPv6 traffic to the virtual machine and are not used externally.

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 provider configured for dual-stack.

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
...
  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 You must use the IPv6 address **fd10:0:2::2/120**.
- 4 You must use the gateway **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}"
```

8.17.2. Creating a service to expose a virtual machine

You can expose a virtual machine within the cluster or outside the cluster by using a **Service** object.

8.17.2.1. About services

A Kubernetes *service* is an abstract way to expose an application running on a set of pods as a network service. Services allow your applications to receive traffic. Services can be exposed in different ways by specifying a **spec.type** in the **Service** object:

ClusterIP

Exposes the service on an internal IP address within the cluster. **ClusterIP** is the default service **type**.

NodePort

Exposes the service on the same port of each selected node in the cluster. **NodePort** makes a service accessible from outside the cluster.

LoadBalancer

Creates an external load balancer in the current cloud (if supported) and assigns a fixed, external IP address to the service.

8.17.2.1.1. 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]**

8.17.2.2. Exposing a virtual machine as a service

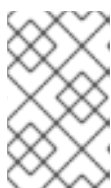
Create a **ClusterIP**, **NodePort**, or **LoadBalancer** service to connect to a running virtual machine (VM) from within or outside the cluster.

Procedure

1. Edit the **VirtualMachine** manifest to add the label for service creation:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-ephemeral
  namespace: example-namespace
spec:
  running: false
  template:
    metadata:
      labels:
        special: key 1
# ...
```

- 1 Add the label **special: key** in the **spec.template.metadata.labels** section.



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: vmervice 1
  namespace: example-namespace 2
spec:
  externalTrafficPolicy: Cluster 3
  ports:
    - nodePort: 30000 4
      port: 27017
      protocol: TCP
      targetPort: 22 5
  selector:
    special: key 6
  type: NodePort 7

```

- 1 The name of the **Service** object.
- 2 The namespace where the **Service** object resides. This must match the **metadata.namespace** field of the **VirtualMachine** manifest.
- 3 Optional: Specifies how the nodes distribute service traffic that is received on external IP addresses. This only applies to **NodePort** and **LoadBalancer** service types. The default value is **Cluster** which routes traffic evenly to all cluster endpoints.
- 4 Optional: When set, the **nodePort** value must be unique across all services. If not specified, a value in the range above **30000** is dynamically allocated.
- 5 Optional: The VM port to be exposed by the service. It must reference an open port if a port list is defined in the VM manifest. If **targetPort** is not specified, it takes the same value as **port**.
- 6 The reference to the label that you added in the **spec.template.metadata.labels** stanza of the **VirtualMachine** manifest.
- 7 The type of service. Possible values are **ClusterIP**, **NodePort** and **LoadBalancer**.

4. Save the **Service** manifest file.
5. Create the service by running the following command:

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

6. Start the VM. If the VM is already running, restart it.

Verification

1. Query the **Service** object to verify that it is available:

```
$ oc get service -n example-namespace
```

Example output for ClusterIP service

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
vmervice	ClusterIP	172.30.3.149	<none>	27017/TCP	2m

Example output for NodePort service

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
vmervice	NodePort	172.30.232.73	<none>	27017:30000/TCP	5m

Example output for LoadBalancer service

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
vmervice	LoadBalancer	172.30.27.5	172.29.10.235,172.29.10.235	27017:31829/TCP	5s

2. Choose the appropriate method to connect to the virtual machine:

- For a **ClusterIP** service, connect to the VM from within the cluster by using the service IP address and the service port. For example:

```
$ ssh fedora@172.30.3.149 -p 27017
```

- For a **NodePort** service, connect to the VM by specifying the node IP address and the node port outside the cluster network. For example:

```
$ ssh fedora@$NODE_IP -p 30000
```

- For a **LoadBalancer** service, use the **vinagre** client to connect to your virtual machine by using the public IP address and port. External ports are dynamically allocated.

8.17.2.3. Additional resources

- [Configuring ingress cluster traffic using a NodePort](#)
- [Configuring ingress cluster traffic using a load balancer](#)

8.17.3. Attaching a virtual machine to a Linux bridge network

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

You must create a Linux bridge network attachment definition (NAD) in order to connect to additional networks.

To attach a virtual machine to an additional network:

1. Create a Linux bridge node network configuration policy.
2. Create a Linux bridge network attachment definition.
3. Configure the virtual machine, enabling the virtual machine to recognize the network attachment definition.

For more information about scheduling, interface types, and other node networking activities, see the [node networking](#) section.

8.17.3.1. Connecting to the network through the network attachment definition

8.17.3.1.1. Creating a Linux bridge node network configuration policy

Use a **NodeNetworkConfigurationPolicy** manifest YAML file to create the Linux bridge.

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.

8.17.3.2. Creating a Linux bridge network attachment definition

8.17.3.2.1. Prerequisites

- A Linux bridge must be configured and attached on every node. See the [node networking](#) section for more information.



WARNING

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

8.17.3.2.2. Creating a Linux bridge network attachment definition in the web console

The network attachment definition is a custom resource that exposes layer-2 devices to a specific namespace in your OpenShift Virtualization cluster.

Network administrators can create network attachment definitions to provide existing layer-2 networking to pods and virtual machines.

Procedure

1. In the web console, click **Networking** → **Network Attachment Definitions**.
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. Click the **Network Type** list and select **CNV Linux bridge**.
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.
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**.



NOTE

A Linux bridge network attachment definition is the most efficient method for connecting a virtual machine to a VLAN.

8.17.3.2.3. Creating a Linux bridge network attachment definition in the CLI

As a network administrator, you can configure a network attachment definition of type **cnv-bridge** to provide layer-2 networking to pods and virtual machines.



NOTE

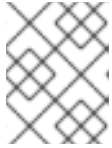
The network attachment definition must be in the same namespace as the pod or virtual machine.

Procedure

1. Create a network attachment definition in the same namespace as the virtual machine.
2. Add the virtual machine to the network attachment definition, 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/<bridge-interface> 2
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "<bridge-network>", 3
    "type": "cnv-bridge", 4
    "bridge": "<bridge-interface>", 5
    "macspoofchk": true, 6
    "vlan": 1 7
  }'
```

- 1 The name for the **NetworkAttachmentDefinition** object.
- 2 Optional: Annotation key-value pair for node selection, where **bridge-interface** is the name of a 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 **bridge-interface** 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.
- 6 Optional: Flag to enable MAC spoof check. When set to **true**, you cannot change the MAC address of the pod or guest interface. This attribute provides security against a MAC spoofing attack by allowing only a single MAC address to exit the pod.
- 7 Optional: The VLAN tag. No additional VLAN configuration is required on the node network configuration policy.

**NOTE**

A Linux bridge network attachment definition is the most efficient method for connecting a virtual machine to a VLAN.

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>
```

8.17.3.3. Configuring the virtual machine for a Linux bridge network**8.17.3.3.1. Creating a NIC for a virtual machine in the web console**

Create and attach additional NICs to a virtual machine from the web console.

Procedure

1. In the correct project in the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click **Network Interfaces** to display the NICs already attached to the virtual machine.
5. Click **Add Network Interface** to create a new slot in the list.
6. Use the **Network** drop-down list to select the network attachment definition for the additional network.
7. Fill in the **Name**, **Model**, **Type**, and **MAC Address** for the new NIC.
8. Click **Add** to save and attach the NIC to the virtual machine.

8.17.3.3.2. Networking fields

Name	Description
Name	Name for the network interface controller.

Name	Description
Model	Indicates the model of the network interface controller. Supported values are e1000e and virtio .
Network	List of available network attachment definitions.
Type	List of available binding methods. For the default pod network, masquerade is the only recommended binding method. For secondary networks, use the bridge binding method. The masquerade method is not supported for non-default networks. Select SR-IOV if you configured an SR-IOV network device and defined that network in the namespace.
MAC Address	MAC address for the network interface controller. If a MAC address is not specified, one is assigned automatically.

8.17.3.3.3. Attaching a virtual machine to an additional network in the CLI

Attach a virtual machine to an additional network by adding a bridge interface and specifying a network attachment definition in the virtual machine configuration.

This procedure uses a YAML file to demonstrate editing the configuration and applying the updated file to the cluster. You can alternatively use the **oc edit <object> <name>** command to edit an existing virtual machine.

Prerequisites

- 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. Create or edit a configuration of a virtual machine that you want to connect to the bridge network.
2. Add the bridge interface to the **spec.template.spec.domain.devices.interfaces** list and the network attachment definition to the **spec.template.spec.networks** list. This example adds a bridge interface called **bridge-net** that connects to the **a-bridge-network** network attachment definition:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: <example-vm>
spec:
  template:
    spec:
      domain:
        devices:
```



```

    interfaces:
      - masquerade: {}
        name: <default>
      - bridge: {}
        name: <bridge-net> ❶
    ...
  networks:
    - name: <default>
      pod: {}
    - name: <bridge-net> ❷
      multus:
        networkName: <network-namespace>/<a-bridge-network> ❸
    ...

```

- ❶ The name of the bridge interface.
- ❷ The name of the network. This value must match the **name** value of the corresponding **spec.template.spec.domain.devices.interfaces** entry.
- ❸ The name of the network attachment definition, prefixed by the namespace where it exists. The namespace must be either the **default** namespace or the same namespace where the VM is to be created. In this case, **multus** is used. Multus is a cloud network interface (CNI) plugin that allows multiple CNIs to exist so that a pod or virtual machine can use the interfaces it needs.

3. Apply the configuration:

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

4. Optional: If you edited a running virtual machine, you must restart it for the changes to take effect.

8.17.4. Configuring IP addresses for virtual machines

You can configure either dynamically or statically provisioned IP addresses for virtual machines.

Prerequisites

- The virtual machine must connect to an [external network](#).
- You must have a DHCP server available on the additional network to configure a dynamic IP for the virtual machine.

8.17.4.1. Configuring an IP address for a new virtual machine using cloud-init

You can use cloud-init to configure an IP address when you create a virtual machine. The IP address can be dynamically or statically provisioned.

Procedure

- Create a virtual machine configuration and include the cloud-init network details in the **spec.volumes.cloudInitNoCloud.networkData** field of the virtual machine configuration:
 - a. To configure a dynamic IP, specify the interface name and the **dhcp4** boolean:

```

kind: VirtualMachine
spec:
  ...
  volumes:
  - cloudInitNoCloud:
    networkData: |
      version: 2
      ethernets:
        eth1: 1
          dhcp4: true 2

```

- 1 The interface name.
- 2 Uses DHCP to provision an IPv4 address.

b. To configure a static IP, specify the interface name and the IP address:

```

kind: VirtualMachine
spec:
  ...
  volumes:
  - cloudInitNoCloud:
    networkData: |
      version: 2
      ethernets:
        eth1: 1
          addresses:
            - 10.10.10.14/24 2

```

- 1 The interface name.
- 2 The static IP address for the virtual machine.

8.17.5. Configuring an SR-IOV network device for virtual machines

You can configure a Single Root I/O Virtualization (SR-IOV) device for virtual machines in your cluster. This process is similar but not identical to configuring an SR-IOV device for OpenShift Container Platform.



NOTE

Live migration is supported for virtual machines that are attached to an SR-IOV network interface only if the **sriovLiveMigration** feature gate is enabled in the HyperConverged Cluster custom resource (CR). When the **spec.featureGates.sriovLiveMigration** field is set to **true**, the **virt-launcher** pod runs with the **SYS_RESOURCE** capability. This might degrade its security.

8.17.5.1. Prerequisites

- You must have [installed the SR-IOV Operator](#).
- You must have [configured the SR-IOV Operator](#).

8.17.5.2. Automated discovery of SR-IOV network devices

The SR-IOV Network Operator searches your cluster for SR-IOV capable network devices on worker nodes. The Operator creates and updates a `SriovNetworkNodeState` custom resource (CR) for each worker node that provides a compatible SR-IOV network device.

The CR is assigned the same name as the worker node. The **status.interfaces** list provides information about the network devices on a node.



IMPORTANT

Do not modify a **SriovNetworkNodeState** object. The Operator creates and manages these resources automatically.

8.17.5.2.1. Example SriovNetworkNodeState object

The following YAML is an example of a **SriovNetworkNodeState** object created by the SR-IOV Network Operator:

An SriovNetworkNodeState object

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodeState
metadata:
  name: node-25 1
  namespace: openshift-sriov-network-operator
  ownerReferences:
  - apiVersion: sriovnetwork.openshift.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: SriovNetworkNodePolicy
    name: default
spec:
  dpConfigVersion: "39824"
status:
  interfaces: 2
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f0
    pciAddress: "0000:18:00.0"
    totalvfs: 8
    vendor: 15b3
  - deviceID: "1017"
    driver: mlx5_core
    mtu: 1500
    name: ens785f1
    pciAddress: "0000:18:00.1"
    totalvfs: 8
    vendor: 15b3
  - deviceID: 158b
    driver: i40e
    mtu: 1500
    name: ens817f0
    pciAddress: 0000:81:00.0
```

```

totalvfs: 64
vendor: "8086"
- deviceID: 158b
  driver: i40e
  mtu: 1500
  name: ens817f1
  pciAddress: 0000:81:00.1
totalvfs: 64
vendor: "8086"
- deviceID: 158b
  driver: i40e
  mtu: 1500
  name: ens803f0
  pciAddress: 0000:86:00.0
totalvfs: 64
vendor: "8086"
syncStatus: Succeeded

```

- 1 The value of the **name** field is the same as the name of the worker node.
- 2 The **interfaces** stanza includes a list of all of the SR-IOV devices discovered by the Operator on the worker node.

8.17.5.3. 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.

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 **128**.
- 8 The **nicSelector** mapping selects the Ethernet device for the Operator to configure. You do not need to specify values for all the parameters. 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}'
```

8.17.5.4. Next steps

- [Configuring an SR-IOV network attachment for virtual machines](#)

8.17.6. Defining an SR-IOV network

You can create a network attachment for a Single Root I/O Virtualization (SR-IOV) device for virtual machines.

After the network is defined, you can attach virtual machines to the SR-IOV network.

8.17.6.1. Prerequisites

- You must have [configured an SR-IOV device for virtual machines](#).

8.17.6.2. Configuring SR-IOV additional network

You can configure an additional network that uses SR-IOV hardware by creating a **SriovNetwork** object. When you create a **SriovNetwork** object, the SR-IOV Operator automatically creates a **NetworkAttachmentDefinition** object.

Users can then attach virtual machines to the SR-IOV network by specifying the network in the virtual machine configurations.



NOTE

Do not modify or delete a **SriovNetwork** object if it is attached to any pods or virtual machines in the **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 `<spooof_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.

- To create the object, enter the following command. Replace `<name>` with a name for this additional network.

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

- 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>
```

8.17.6.3. Next steps

- [Attaching a virtual machine to an SR-IOV network.](#)

8.17.7. Attaching a virtual machine to an SR-IOV network

You can attach a virtual machine to use a Single Root I/O Virtualization (SR-IOV) network as a secondary network.

8.17.7.1. Prerequisites

- You must have [configured an SR-IOV device for virtual machines](#).
- You must have [defined an SR-IOV network](#).

8.17.7.2. Attaching a virtual machine to an SR-IOV network

You can attach the virtual machine to the SR-IOV network by including the network details in the virtual machine configuration.

Procedure

1. Include the SR-IOV network details in the **spec.domain.devices.interfaces** and **spec.networks** of the virtual machine configuration:

```
kind: VirtualMachine
...
spec:
  domain:
    devices:
      interfaces:
        - name: <default> 1
          masquerade: {} 2
        - name: <nic1> 3
          sriov: {}
      networks:
        - name: <default> 4
          pod: {}
        - name: <nic1> 5
          multus:
            networkName: <sriov-network> 6
...

```

- 1 A unique name for the interface that is connected to the pod network.
- 2 The **masquerade** binding to the default pod network.
- 3 A unique name for the SR-IOV interface.
- 4 The name of the pod network interface. This must be the same as the **interfaces.name** that you defined earlier.
- 5 The name of the SR-IOV interface. This must be the same as the **interfaces.name** that you defined earlier.
- 6 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.

8.17.8. Viewing the IP address of NICs on a virtual machine

You can view the IP address for a network interface controller (NIC) by using the web console or the **oc** client. The [QEMU guest agent](#) displays additional information about the virtual machine's secondary networks.

8.17.8.1. Viewing the IP address of a virtual machine interface in the CLI

The network interface configuration is included in the **oc describe vmi <vmi_name>** command.

You can also view the IP address information by running **ip addr** on the virtual machine, or by running **oc get vmi <vmi_name> -o yaml**.

Procedure

- Use the **oc describe** command to display the virtual machine interface configuration:

```
$ 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:fe4:25/64
  Mac:          0a:58:0a:f4:00:25
  Name:         default
  Interface Name: v2
  Ip Address:    1.1.1.7/24
  Ip Addresses:
    1.1.1.7/24
    fe80::f4d9:70ff:fe13:9089/64
  Mac:          f6:d9:70:13:90:89
  Interface Name: v1
  Ip Address:    1.1.1.1/24
  Ip Addresses:
    1.1.1.1/24
    1.1.1.2/24
    1.1.1.4/24
    2001:de7:0:f101::1/64
    2001:db8:0:f101::1/64
    fe80::1420:84ff:fe10:17aa/64
  Mac:          16:20:84:10:17:aa
```

8.17.8.2. Viewing the IP address of a virtual machine interface in the web console

The IP information displays in the **Virtual Machine Overview** screen for the virtual machine.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.

2. Click the **Virtual Machines** tab.
3. Select a virtual machine name to open the **Virtual Machine Overview** screen.

The information for each attached NIC is displayed under **IP Address**.

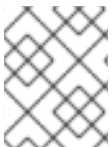
8.17.9. Using a MAC address pool for virtual machines

The *KubeMacPool* component provides a MAC address pool service for virtual machine NICs in a namespace.

8.17.9.1. About KubeMacPool

KubeMacPool provides a MAC address pool per namespace and allocates MAC addresses for virtual machine NICs from the pool. This ensures that the NIC is assigned a unique MAC address that does not conflict with the MAC address of another virtual machine.

Virtual machine instances created from that virtual machine retain the assigned MAC address across reboots.



NOTE

KubeMacPool does not handle virtual machine instances created independently from a virtual machine.

KubeMacPool is enabled by default when you install OpenShift Virtualization. You can disable a MAC address pool for a namespace by adding the **mutatevirtualmachines.kubemacpool.io=ignore** label to the namespace. Re-enable KubeMacPool for the namespace by removing the label.

8.17.9.2. Disabling a MAC address pool for a namespace in the CLI

Disable a MAC address pool for virtual machines in a namespace by adding the **mutatevirtualmachines.kubemacpool.io=ignore** label to the namespace.

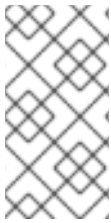
Procedure

- Add the **mutatevirtualmachines.kubemacpool.io=ignore** label to the namespace. The following example disables KubeMacPool for two namespaces, **<namespace1>** and **<namespace2>**:

```
$ oc label namespace <namespace1> <namespace2>
mutatevirtualmachines.kubemacpool.io=ignore
```

8.17.9.3. Re-enabling a MAC address pool for a namespace in the CLI

If you disabled KubeMacPool for a namespace and want to re-enable it, remove the **mutatevirtualmachines.kubemacpool.io=ignore** label from the namespace.

**NOTE**

Earlier versions of OpenShift Virtualization used the label **mutatevirtualmachines.kubemacpool.io=allocate** to enable KubeMacPool for a namespace. This is still supported but redundant as KubeMacPool is now enabled by default.

Procedure

- Remove the KubeMacPool label from the namespace. The following example re-enables KubeMacPool for two namespaces, **<namespace1>** and **<namespace2>**:

```
$ oc label namespace <namespace1> <namespace2>
mutatevirtualmachines.kubemacpool.io-
```

8.18. VIRTUAL MACHINE DISKS**8.18.1. Storage features**

Use the following table to determine feature availability for local and shared persistent storage in OpenShift Virtualization.

8.18.1.1. OpenShift Virtualization storage feature matrix**Table 8.7. OpenShift Virtualization storage feature matrix**

	Virtual machine live migration	Host-assisted virtual machine disk cloning	Storage-assisted virtual machine disk cloning	Virtual machine snapshots
OpenShift Container Storage: RBD block-mode volumes	Yes	Yes	Yes	Yes
OpenShift Virtualization hostpath provisioner	No	Yes	No	No
Other multi-node writable storage	Yes ^[1]	Yes	Yes ^[2]	Yes ^[2]
Other single-node writable storage	No	Yes	Yes ^[2]	Yes ^[2]

- PVCs must request a ReadWriteMany access mode.
- Storage provider must support both Kubernetes and CSI snapshot APIs



NOTE

You cannot live migrate virtual machines that use:

- A storage class with ReadWriteOnce (RWO) access mode
- Passthrough features such as GPUs or SR-IOV network interfaces that have the **sriovLiveMigration** feature gate disabled

Do not set the **evictionStrategy** field to **LiveMigrate** for these virtual machines.

8.18.2. Configuring local storage for virtual machines

You can configure local storage for your virtual machines by using the `hostpath` provisioner feature.

8.18.2.1. About the `hostpath` provisioner

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.

When you install the OpenShift Virtualization Operator, the `hostpath` provisioner Operator is automatically installed. To use it, you must:

- Configure SELinux:
 - If you use Red Hat Enterprise Linux CoreOS (RHCOS) 8 workers, you must create a **MachineConfig** object on each node.
 - Otherwise, apply the SELinux label **container_file_t** to the persistent volume (PV) backing directory on each node.
- Create a **HostPathProvisioner** custom resource.
- Create a **StorageClass** object for the `hostpath` provisioner.

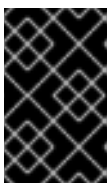
The `hostpath` provisioner Operator deploys the provisioner as a *DaemonSet* on each node when you create its custom resource. In the custom resource file, you specify the backing directory for the persistent volumes that the `hostpath` provisioner creates.

8.18.2.2. Configuring SELinux for the `hostpath` provisioner on Red Hat Enterprise Linux CoreOS (RHCOS) 8

You must configure SELinux before you create the **HostPathProvisioner** custom resource. To configure SELinux on Red Hat Enterprise Linux CoreOS (RHCOS) 8 workers, you must create a **MachineConfig** object on each node.

Prerequisites

- Create a backing directory on each node for the persistent volumes (PVs) that the `hostpath` provisioner creates.



IMPORTANT

The backing directory must not be located in the filesystem's root directory because the `/` partition is read-only on RHCOS. For example, you can use `/var/<directory_name>` but not `<directory_name>`.

**WARNING**

If you select a directory that shares space with your operating system, you might exhaust the space on that partition and your node might become non-functional. Create a separate partition and point the hostpath provisioner to the separate partition to avoid interference with your operating system.

Procedure

1. Create the **MachineConfig** file. For example:

```
$ touch machineconfig.yaml
```

2. Edit the file, ensuring that you include the directory where you want the hostpath provisioner to create PVs. For example:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: 50-set-selinux-for-hostpath-provisioner
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Set SELinux chcon for hostpath provisioner
            Before=kubelet.service

            [Service]
            ExecStart=/usr/bin/chcon -Rt container_file_t <backing_directory_path> 1

            [Install]
            WantedBy=multi-user.target
          enabled: true
          name: hostpath-provisioner.service
```

- 1 Specify the backing directory where you want the provisioner to create PVs. This directory must not be located in the filesystem's root directory (*/*).

3. Create the **MachineConfig** object:

```
$ oc create -f machineconfig.yaml -n <namespace>
```

8.18.2.3. Using the hostpath provisioner to enable local storage

To deploy the hostpath provisioner and enable your virtual machines to use local storage, first create a **HostPathProvisioner** custom resource.

Prerequisites

- Create a backing directory on each node for the persistent volumes (PVs) that the hostpath provisioner creates.



IMPORTANT

The backing directory must not be located in the filesystem's root directory because the / partition is read-only on Red Hat Enterprise Linux CoreOS (RHCOS). For example, you can use `/var/<directory_name>` but not `<directory_name>`.



WARNING

If you select a directory that shares space with your operating system, you might exhaust the space on that partition and your node becomes non-functional. Create a separate partition and point the hostpath provisioner to the separate partition to avoid interference with your operating system.

- Apply the SELinux context **container_file_t** to the PV backing directory on each node. For example:

```
$ sudo chcon -t container_file_t -R <backing_directory_path>
```



NOTE

If you use Red Hat Enterprise Linux CoreOS (RHCOS) 8 workers, you must configure SELinux by using a **MachineConfig** manifest instead.

Procedure

1. Create the **HostPathProvisioner** custom resource file. For example:

```
$ touch hostpathprovisioner_cr.yaml
```

2. Edit the file, ensuring that the **spec.pathConfig.path** value is the directory where you want the hostpath provisioner to create PVs. For example:

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
```

```
pathConfig:
  path: "<backing_directory_path>" ❶
  useNamingPrefix: false ❷
workload: ❸
```

- ❶ Specify the backing directory where you want the provisioner to create PVs. This directory must not be located in the filesystem's root directory (*/*).
- ❷ Change this value to **true** if you want to use the name of the persistent volume claim (PVC) that is bound to the created PV as the prefix of the directory name.
- ❸ Optional: You can use the **spec.workload** field to configure node placement rules for the hostpath provisioner.



NOTE

If you did not create the backing directory, the provisioner attempts to create it for you. If you did not apply the **container_file_t** SELinux context, this can cause **Permission denied** errors.

3. Create the custom resource in the **openshift-cnv** namespace:

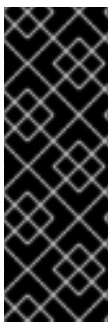
```
$ oc create -f hostpathprovisioner_cr.yaml -n openshift-cnv
```

Additional resources

- [Specifying nodes for virtualization components](#)

8.18.2.4. Creating a storage class

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.



IMPORTANT

When using OpenShift Virtualization with OpenShift Container Platform Container Storage, specify RBD block mode persistent volume claims (PVCs) when creating virtual machine disks. With 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**.

Procedure

1. Create a YAML file for defining the storage class. For example:

```
$ touch storageclass.yaml
```

2. Edit the file. For example:


```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: hostpath-provisioner ❶
provisioner: kubevirt.io/hostpath-provisioner
reclaimPolicy: Delete ❷
volumeBindingMode: WaitForFirstConsumer ❸

```

- ❶ You can optionally rename the storage class by changing this value.
- ❷ The two possible **reclaimPolicy** values are **Delete** and **Retain**. If you do not specify a value, the storage class defaults to **Delete**.
- ❸ The **volumeBindingMode** value determines when dynamic provisioning and volume binding occur. Specify **WaitForFirstConsumer** to delay the binding and provisioning of a 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.



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 PVC to a PV on the correct node. By using **StorageClass** with **volumeBindingMode** set to **WaitForFirstConsumer**, the binding and provisioning of the PV is delayed until a **Pod** is created using the PVC.

3. Create the **StorageClass** object:

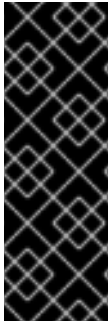
```
$ oc create -f storageclass.yaml
```

Additional resources

- [Storage classes](#)

8.18.3. Creating data volumes using profiles

When you create a data volume, the Containerized Data Importer (CDI) creates a persistent volume claim (PVC) and populates the PVC with your data. You can create a data volume as either a standalone resource or by using a **dataVolumeTemplate** resource in a virtual machine specification. You create a data volume by using either the PVC API or storage APIs.



IMPORTANT

When using OpenShift Virtualization with OpenShift Container Platform Container Storage, specify RBD block mode persistent volume claims (PVCs) when creating virtual machine disks. With 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**.

TIP

Whenever possible, use the storage API to optimize space allocation and maximize performance.

A *storage profile* is a custom resource that the CDI manages. It provides recommended storage settings based on the associated storage class. A storage profile is allocated for each storage class.

Storage profiles enable you to create data volumes quickly while reducing coding and minimizing potential errors.

For recognized storage types, the CDI provides values that optimize the creation of PVCs. However, you can configure automatic settings for a storage class if you customize the storage profile.

8.18.3.1. Creating data volumes using the storage API

When you create a data volume using the storage API, the Containerized Data Interface (CDI) optimizes your persistent volume claim (PVC) allocation based on the type of storage supported by your selected storage class. You only have to specify the data volume name, namespace, and the amount of storage that you want to allocate.

For example:

- When using Ceph RBD, **accessModes** is automatically set to **ReadWriteMany**, which enables live migration. **volumeMode** is set to **Block** to maximize performance.
- When you are using **volumeMode: Filesystem**, more space will automatically be requested by the CDI, if required to accommodate file system overhead.

In the following YAML, using the storage API requests a data volume with two gigabytes of usable space. The user does not need to know the **volumeMode** in order to correctly estimate the required persistent volume claim (PVC) size. The CDI chooses the optimal combination of **accessModes** and **volumeMode** attributes automatically. These optimal values are based on the type of storage or the defaults that you define in your storage profile. If you want to provide custom values, they override the system-calculated values.

Example DataVolume definition

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

```

    name: "<my_vm_disk>" 4
  storage: 5
  resources:
    requests:
      storage: 2Gi 6
  storageClassName: <storage_class> 7

```

- 1 The name of the new data volume.
- 2 Indicate that the source of the import is an existing persistent volume claim (PVC).
- 3 The namespace where the source PVC exists.
- 4 The name of the source PVC.
- 5 Indicates allocation using the storage API.
- 6 Specifies the amount of available space that you request for the PVC.
- 7 Optional: The name of the storage class. If the storage class is not specified, the system default storage class is used.

8.18.3.2. Creating data volumes using the PVC API

When you create a data volume using the PVC API, the Containerized Data Interface (CDI) creates the data volume based on what you specify for the following fields:

- **accessModes** (**ReadWriteOnce**, **ReadWriteMany**, or **ReadOnlyMany**)
- **volumeMode** (**Filesystem** or **Block**)
- **capacity** of **storage** (**5Gi**, for example)

In the following YAML, using the PVC API allocates a data volume with a storage capacity of two gigabytes. You specify an access mode of **ReadWriteMany** to enable live migration. Because you know the values your system can support, you specify **Block** storage instead of the default, **Filesystem**.

Example DataVolume definition

```

apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <datavolume> 1
spec:
  source:
    pvc: 2
      namespace: "<source_namespace>" 3
      name: "<my_vm_disk>" 4
  pvc: 5
    accessModes: 6
      - ReadWriteMany
  resources:
    requests:

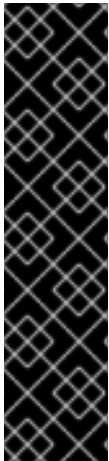
```

```

storage: 2Gi 7
volumeMode: Block 8
storageClassName: <storage_class> 9

```

- 1** The name of the new data volume.
- 2** In the **source** section, **pvc** indicates that the source of the import is an existing persistent volume claim (PVC).
- 3** The namespace where the source PVC exists.
- 4** The name of the source PVC.
- 5** Indicates allocation using the PVC API.
- 6** **accessModes** is required when using the PVC API.
- 7** Specifies the amount of space you are requesting for your data volume.
- 8** Specifies that the destination is a block PVC.
- 9** Optionally, specify the storage class. If the storage class is not specified, the system default storage class is used.



IMPORTANT

When you explicitly allocate a data volume by using the PVC API and you are not using **volumeMode: Block**, consider file system overhead.

File system overhead is the amount of space required by the file system to maintain its metadata. The amount of space required for file system metadata is file system dependent. Failing to account for file system overhead in your storage capacity request can result in an underlying persistent volume claim (PVC) that is not large enough to accommodate your virtual machine disk.

If you use the storage API, the CDI will factor in file system overhead and request a larger persistent volume claim (PVC) to ensure that your allocation request is successful.

8.18.3.3. 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.

Prerequisites

- 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.



NOTE

An empty **status** section in a storage profile indicates that a storage provisioner is not recognized by the Containerized Data Interface (CDI). Customizing a storage profile is necessary if you have a storage provisioner that is not recognized by the CDI. In this case, the administrator sets appropriate values in the storage profile to ensure successful allocations.



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.

Procedure

1. Edit the storage profile. In this example, the provisioner is not recognized by CDI:

```
$ oc edit -n openshift-cnv storageprofile <storage_class>
```

Example storage profile

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <some_unknown_provisioner_class>
  # ...
spec: {}
status:
  provisioner: <some_unknown_provisioner>
  storageClass: <some_unknown_provisioner_class>
```

2. Provide the needed attribute values in the storage profile:

Example storage profile

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <some_unknown_provisioner_class>
  # ...
spec:
  claimPropertySets:
  - accessModes:
    - ReadWriteOnce 1
    volumeMode:
      Filesystem 2
status:
  provisioner: <some_unknown_provisioner>
  storageClass: <some_unknown_provisioner_class>
```

-
- 1 The **accessModes** that you select.
- 2 The **volumeMode** that you select.

After you save your changes, the selected values appear in the storage profile **status** element.

8.18.3.4. Additional resources

- [Creating a storage class](#)
- [Overriding the default file system overhead value](#)
- [Cloning a data volume using smart cloning](#)

8.18.4. Reserving PVC space for file system overhead

By default, the Containerized Data Importer (CDI) reserves space for file system overhead data in persistent volume claims (PVCs) that use the **Filesystem** volume mode. You can set the percentage that CDI reserves for this purpose globally and for specific storage classes.

8.18.4.1. How file system overhead affects space for virtual machine disks

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 virtual machine disk.
- The space that the Containerized Data Importer (CDI) reserves for file system overhead, such as metadata.

By default, CDI reserves 5.5% of the PVC space for overhead, reducing the space available for virtual machine disks by that amount.

If a different value works better for your use case, you can configure the overhead value by editing the **CDI** object. You can change the value globally and you can specify values for specific storage classes.

8.18.4.2. Overriding the default file system overhead value

Change the amount of persistent volume claim (PVC) space that the Containerized Data Importer (CDI) reserves for file system overhead by editing the **spec.config.filesystemOverhead** attribute of the **CDI** object.

Prerequisites

- Install the OpenShift CLI (**oc**).

Procedure

1. Open the **CDI** object for editing by running the following command:

```
$ oc edit cdi
```

2. Edit the **spec.config.filesystemOverhead** fields, populating them with your chosen values:

```

...
spec:
  config:
    filesystemOverhead:
      global: "<new_global_value>" 1
      storageClass:
        <storage_class_name>: "<new_value_for_this_storage_class>" 2

```

- 1 The file system overhead percentage that CDI uses across the cluster. 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 **CDI** object.

Verification

- View the **CDI** status and verify your changes by running the following command:

```
$ oc get cdi -o yaml
```

8.18.5. Configuring CDI to work with namespaces that have a compute resource quota

You can use the Containerized Data Importer (CDI) to import, upload, and clone virtual machine disks into namespaces that are subject to CPU and memory resource restrictions.

8.18.5.1. 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.

8.18.5.2. 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 hco -n openshift-cnvd kubevirt-hyperconverged
```

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.

8.18.5.3. Additional resources

- [Resource quotas per project](#)

8.18.6. 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.

8.18.6.1. 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: dv-ann
  annotations:
    v1.multus-cni.io/default-network: bridge-network 1
spec:
  source:
    http:
      url: "example.exampleurl.com"
  pvc:
    accessModes:
      - ReadWriteOnce
```



```
resources:
  requests:
    storage: 1Gi
```

1 Multus network annotation

8.18.7. 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.

8.18.7.1. 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.

8.18.7.2. 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
  ...
pvc:
  ...
preallocation: true 2
```

- 1 All CDI source types support preallocation, however preallocation is ignored for cloning operations.
- 2 The **preallocation** field is a boolean that defaults to false.

8.18.8. Uploading local disk images by using the web console

You can upload a locally stored disk image file by using the web console.

8.18.8.1. Prerequisites

- You must have a virtual machine image file in IMG, ISO, or QCOW2 format.
- If you require scratch space according to the [CDI supported operations matrix](#), you must first [define a storage class](#) or [prepare CDI scratch space](#) for this operation to complete successfully.

8.18.8.2. 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)	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2** ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

Unsupported operation

* Requires scratch space

** Requires scratch space if a custom certificate authority is required

8.18.8.3. Uploading an image file using the web console

Use the web console to upload an image file to a new persistent volume claim (PVC). You can later use this PVC to attach the image to new virtual machines.

Prerequisites

- You must have one of the following:
 - A raw virtual machine image file in either ISO or IMG format.
 - A virtual machine image file in QCOW2 format.

- For best results, compress your image file according to the following guidelines before you upload it:
 - Compress a raw image file by using **xz** or **gzip**.

**NOTE**

Using a compressed raw image file results in the most efficient upload.

- Compress a QCOW2 image file by using the method that is recommended for your client:
 - If you use a Linux client, *sparsify* the QCOW2 file by using the [virt-sparsify](#) tool.
 - If you use a Windows client, compress the QCOW2 file by using **xz** or **gzip**.

Procedure

1. From the side menu of the web console, click **Storage** → **Persistent Volume Claims**
2. Click the **Create Persistent Volume Claim** drop-down list to expand it.
3. Click **With Data Upload Form** to open the **Upload Data to Persistent Volume Claim** page.
4. Click **Browse** to open the file manager and select the image that you want to upload, or drag the file into the **Drag a file here or browse to upload** field.
5. Optional: Set this image as the default image for a specific operating system.
 - a. Select the **Attach this data to a virtual machine operating system** check box.
 - b. Select an operating system from the list.
6. The **Persistent Volume Claim Name** field is automatically filled with a unique name and cannot be edited. Take note of the name assigned to the PVC so that you can identify it later, if necessary.
7. Select a storage class from the **Storage Class** list.
8. In the **Size** field, enter the size value for the PVC. Select the corresponding unit of measurement from the drop-down list.

**WARNING**

The PVC size must be larger than the size of the uncompressed virtual disk.

9. Select an **Access Mode** that matches the storage class that you selected.
10. Click **Upload**.

8.18.8.4. Additional resources

- [Configure preallocation mode](#) to improve write performance for data volume operations.

8.18.9. Uploading local disk images by using the virtctl tool

You can upload a locally stored disk image to a new or existing data volume by using the **virtctl** command-line utility.

8.18.9.1. Prerequisites

- [Install](#) the **kubevirt-virtctl** package.
- If you require scratch space according to the [CDI supported operations matrix](#), you must first [define a storage class or prepare CDI scratch space](#) for this operation to complete successfully.

8.18.9.2. About data volumes

DataVolume objects are custom resources that are provided by the Containerized Data Importer (CDI) project. Data volumes orchestrate import, clone, and upload operations that are associated with an underlying persistent volume claim (PVC). Data volumes are integrated with OpenShift Virtualization, and they prevent a virtual machine from being started before the PVC has been prepared.

8.18.9.3. Creating an upload data volume

You can manually create a data volume with an **upload** data source to use for uploading local disk images.

Procedure

1. Create a data volume configuration that specifies **spec: source: upload{}**:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <upload-datavolume> 1
spec:
  source:
    upload: {}
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi> 2
```

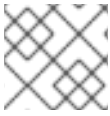
- 1 The name of the data volume.
- 2 The size of the data volume. Ensure that this value is greater than or equal to the size of the disk that you upload.

2. Create the data volume by running the following command:

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

8.18.9.4. Uploading a local disk image to a data volume

You can use the **virtctl** CLI utility to upload a local disk image from a client machine to a data volume (DV) in your cluster. You can use a DV that already exists in your cluster or create a new DV during this procedure.



NOTE

After you upload a local disk image, you can add it to a virtual machine.

Prerequisites

- You must have one of the following:
 - A raw virtual machine image file in either ISO or IMG format.
 - A virtual machine image file in QCOW2 format.
- For best results, compress your image file according to the following guidelines before you upload it:
 - Compress a raw image file by using **xz** or **gzip**.



NOTE

Using a compressed raw image file results in the most efficient upload.

- Compress a QCOW2 image file by using the method that is recommended for your client:
 - If you use a Linux client, *sparsify* the QCOW2 file by using the [virt-sparsify](#) tool.
 - If you use a Windows client, compress the QCOW2 file by using **xz** or **gzip**.
- The **kubevirt-virtctl** package must be installed on the client machine.
- The client machine must be configured to trust the OpenShift Container Platform router's certificate.

Procedure

1. Identify the following items:
 - The name of the upload data volume that you want to use. If this data volume does not exist, it is created automatically.
 - The size of the data volume, if you want it to be created during the upload procedure. The size must be greater than or equal to the size of the disk image.
 - The file location of the virtual machine disk image that you want to upload.
2. Upload the disk image by running the **virtctl image-upload** command. Specify the parameters that you identified in the previous step. For example:

```
$ 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 virtual machine disk 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. Be aware that when you use the `--insecure` flag, the authenticity of the upload endpoint is **not** verified.

3. Optional. To verify that a data volume was created, view all data volumes by running the following command:

```
$ oc get dvs
```

8.18.9.5. 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)	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2** ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

Unsupported operation

* Requires scratch space

** Requires scratch space if a custom certificate authority is required

8.18.9.6. Additional resources

- [Configure preallocation mode](#) to improve write performance for data volume operations.

8.18.10. Uploading a local disk image to a block storage data volume

You can upload a local disk image into a block data volume by using the **virtctl** command-line utility.

In this workflow, you create a local block device to use as a persistent volume, associate this block volume with an **upload** data volume, and use **virtctl** to upload the local disk image into the data volume.

8.18.10.1. Prerequisites

- Install the **kubevirt-virtctl** package.
- If you require scratch space according to the [CDI supported operations matrix](#), you must first [define a storage class or prepare CDI scratch space](#) for this operation to complete successfully.

8.18.10.2. About data volumes

DataVolume objects are custom resources that are provided by the Containerized Data Importer (CDI) project. Data volumes orchestrate import, clone, and upload operations that are associated with an underlying persistent volume claim (PVC). Data volumes are integrated with OpenShift Virtualization, and they prevent a virtual machine from being started before the PVC has been prepared.

8.18.10.3. About block persistent volumes

A block persistent volume (PV) is a PV that is backed by a raw block device. These volumes do not have a file system and can provide performance benefits for virtual machines by reducing overhead.

Raw block volumes are provisioned by specifying **volumeMode: Block** in the PV and persistent volume claim (PVC) specification.

8.18.10.4. Creating a local block persistent volume

Create a local block persistent volume (PV) on a node by populating a file and mounting it as a loop device. You can then reference this loop device in a PV manifest as a **Block** volume and use it as a block device for a virtual machine image.

Procedure

1. Log in as **root** to the node on which to create the local PV. This procedure uses **node01** for its examples.
2. Create a file and populate it with null characters so that it can be used as a block device. The following example creates a file **loop10** with a size of 2Gb (20 100Mb blocks):

```
$ dd if=/dev/zero of=<loop10> bs=100M count=20
```

3. Mount the **loop10** file as a loop device.

```
$ losetup </dev/loop10>d3 <loop10> 1 2
```

- 1** File path where the loop device is mounted.
- 2** The file created in the previous step to be mounted as the loop device.

4. Create a **PersistentVolume** manifest that references the mounted loop device.

```

kind: PersistentVolume
apiVersion: v1
metadata:
  name: <local-block-pv10>
  annotations:
spec:
  local:
    path: </dev/loop10> 1
  capacity:
    storage: <2Gi>
  volumeMode: Block 2
  storageClassName: local 3
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - <node01> 4

```

- 1 The path of the loop device on the node.
- 2 Specifies it is a block PV.
- 3 Optional: Set a storage class for the PV. If you omit it, the cluster default is used.
- 4 The node on which the block device was mounted.

5. Create the block PV.

```
# oc create -f <local-block-pv10.yaml> 1
```

- 1 The file name of the persistent volume created in the previous step.

8.18.10.5. Creating an upload data volume

You can manually create a data volume with an **upload** data source to use for uploading local disk images.

Procedure

1. Create a data volume configuration that specifies **spec: source: upload{}**:

```

apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <upload-datavolume> 1

```



```
spec:
  source:
    upload: {}
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi> 2
```

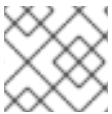
- 1 The name of the data volume.
- 2 The size of the data volume. Ensure that this value is greater than or equal to the size of the disk that you upload.

2. Create the data volume by running the following command:

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

8.18.10.6. Uploading a local disk image to a data volume

You can use the **virtctl** CLI utility to upload a local disk image from a client machine to a data volume (DV) in your cluster. You can use a DV that already exists in your cluster or create a new DV during this procedure.



NOTE

After you upload a local disk image, you can add it to a virtual machine.

Prerequisites

- You must have one of the following:
 - A raw virtual machine image file in either ISO or IMG format.
 - A virtual machine image file in QCOW2 format.
- For best results, compress your image file according to the following guidelines before you upload it:
 - Compress a raw image file by using **xz** or **gzip**.



NOTE

Using a compressed raw image file results in the most efficient upload.

- Compress a QCOW2 image file by using the method that is recommended for your client:
 - If you use a Linux client, *sparsify* the QCOW2 file by using the [virt-sparsify](#) tool.
 - If you use a Windows client, compress the QCOW2 file by using **xz** or **gzip**.
- The **kubevirt-virtctl** package must be installed on the client machine.

- The client machine must be configured to trust the OpenShift Container Platform router's certificate.

Procedure

1. Identify the following items:
 - The name of the upload data volume that you want to use. If this data volume does not exist, it is created automatically.
 - The size of the data volume, if you want it to be created during the upload procedure. The size must be greater than or equal to the size of the disk image.
 - The file location of the virtual machine disk image that you want to upload.
2. Upload the disk image by running the **virtctl image-upload** command. Specify the parameters that you identified in the previous step. For example:

```
$ 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 virtual machine disk 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. Be aware that when you use the **--insecure** flag, the authenticity of the upload endpoint is **not** verified.

3. Optional. To verify that a data volume was created, view all data volumes by running the following command:

```
$ oc get dvs
```

8.18.10.7. 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)	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2** ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

Unsupported operation

* Requires scratch space

** Requires scratch space if a custom certificate authority is required

8.18.10.8. Additional resources

- [Configure preallocation mode](#) to improve write performance for data volume operations.

8.18.11. Managing offline virtual machine snapshots

You can create, restore, and delete virtual machine (VM) snapshots for VMs that are powered off (offline). OpenShift Virtualization supports offline VM snapshots on:

- Red Hat OpenShift Container Storage
- Any other storage provider with the Container Storage Interface (CSI) driver that supports the Kubernetes Volume Snapshot API

8.18.11.1. About virtual machine 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.

An offline VM snapshot is created from a VM that is powered off (Stopped state). 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.

With the offline VM snapshots feature, cluster administrators and application developers can:

- Create a new snapshot
- List all snapshots attached to a specific VM
- Restore a VM from a snapshot
- Delete an existing VM snapshot

8.18.11.1. Virtual machine snapshot controller and custom resource definitions (CRDs)

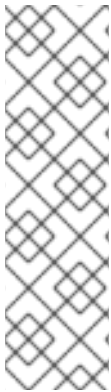
The VM snapshot feature introduces three new API objects defined as CRDs for managing snapshots:

- **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.

8.18.11.2. Creating an offline virtual machine snapshot in the web console

You can create a virtual machine (VM) snapshot by using the web console.



NOTE

The VM snapshot only includes disks that meet the following requirements:

- Must be either a data volume or persistent volume claim
- Belong to a storage class that supports Container Storage Interface (CSI) volume snapshots

If your VM storage includes disks that do not support snapshots, you can either edit them or contact your cluster administrator.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. If the virtual machine is running, click **Actions** → **Stop Virtual Machine** to power it down.
5. Click the **Snapshots** tab and then click **Take Snapshot**.
6. Fill in the **Snapshot Name** and optional **Description** fields.
7. Expand **Disks included in this Snapshot** to see the storage volumes to be included in the snapshot.
8. If your VM has disks that cannot be included in the snapshot and you still wish to proceed, select the **I am aware of this warning and wish to proceed** checkbox.
9. Click **Save**.

8.18.11.3. Creating an offline virtual machine snapshot in the CLI

You can create a virtual machine (VM) snapshot for an offline VM by creating a **VirtualMachineSnapshot** object.

Prerequisites

- Ensure that the persistent volume claims (PVCs) are in a storage class that supports Container Storage Interface (CSI) volume snapshots.
- Install the OpenShift CLI (**oc**).
- 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.

For example:

```
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineSnapshot
metadata:
  name: my-vmsnapshot 1
spec:
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: my-vm 2
```

1 The name of the new **VirtualMachineSnapshot** object.

2 The name of the source VM.

2. Create the **VirtualMachineSnapshot** resource. The snapshot controller creates a **VirtualMachineSnapshotContent** object, binds it to the **VirtualMachineSnapshot** and updates the **status** and **readyToUse** fields of the **VirtualMachineSnapshot** object.

```
$ oc create -f <my-vmsnapshot>.yaml
```

Verification

1. Verify that the **VirtualMachineSnapshot** object is created and bound with **VirtualMachineSnapshotContent**. The **readyToUse** flag must be set to **true**.

```
$ oc describe vmsnapshot <my-vmsnapshot>
```

Example output

```
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineSnapshot
metadata:
  creationTimestamp: "2020-09-30T14:41:51Z"
finalizers:
- snapshot.kubevirt.io/vmsnapshot-protection
```

```

generation: 5
name: mysnap
namespace: default
resourceVersion: "3897"
selfLink:
/apis/snapshot.kubevirt.io/v1alpha1/namespaces/default/virtualmachinesnapshots/my-
vmsnapshot
uid: 28eedf08-5d6a-42c1-969c-2eda58e2a78d
spec:
source:
apiGroup: kubevirt.io
kind: VirtualMachine
name: my-vm
status:
conditions:
- lastProbeTime: null
lastTransitionTime: "2020-09-30T14:42:03Z"
reason: Operation complete
status: "False" ❶
type: Progressing
- lastProbeTime: null
lastTransitionTime: "2020-09-30T14:42:03Z"
reason: Operation complete
status: "True" ❷
type: Ready
creationTime: "2020-09-30T14:42:03Z"
readyToUse: true ❸
sourceUID: 355897f3-73a0-4ec4-83d3-3c2df9486f4f
virtualMachineSnapshotContentName: vmsnapshot-content-28eedf08-5d6a-42c1-969c-
2eda58e2a78d ❹

```

- ❶ The **status** field of the **Progressing** condition specifies if the snapshot is still being created.
- ❷ The **status** field of the **Ready** condition specifies if the snapshot creation process is complete.
- ❸ Specifies if the snapshot is ready to be used.
- ❹ Specifies that the snapshot is bound to a **VirtualMachineSnapshotContent** object created by the snapshot controller.

2. Check the **spec:volumeBackups** property of the **VirtualMachineSnapshotContent** resource to verify that the expected PVCs are included in the snapshot.

8.18.11.4. Restoring a virtual machine from a snapshot in the web console

You can restore a virtual machine (VM) to a previous configuration represented by a snapshot in the web console.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.

2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. If the virtual machine is running, click **Actions → Stop Virtual Machine** to power it down.
5. Click the **Snapshots** tab. The page displays a list of snapshots associated with the virtual machine.
6. Choose one of the following methods to restore a VM snapshot:
 - a. For the snapshot that you want to use as the source to restore the VM, click **Restore**.
 - b. Select a snapshot to open the **Snapshot Details** screen and click **Actions → Restore Virtual Machine Snapshot**.
7. In the confirmation pop-up window, click **Restore** to restore the VM to its previous configuration represented by the snapshot.

8.18.11.5. Restoring a virtual machine from a snapshot in the CLI

You can restore an existing virtual machine (VM) to a previous configuration by using a VM snapshot .

Prerequisites

- Install the OpenShift CLI (**oc**).
- Power down the VM you want to restore to a previous state.

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.

For example:

```
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineRestore
metadata:
  name: my-vmrestore 1
spec:
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: my-vm 2
  virtualMachineSnapshotName: my-vmssnapshot 3
```

- 1** The name of the new **VirtualMachineRestore** object.
- 2** The name of the target VM you want to restore.
- 3** The name of the **VirtualMachineSnapshot** object to be used as the source.

2. Create the **VirtualMachineRestore** resource. The snapshot controller updates the status fields of the **VirtualMachineRestore** object and replaces the existing VM configuration with the snapshot content.

```
$ oc create -f <my-vmrestore>.yaml
```

Verification

- Verify that the VM is restored to the previous state represented by the snapshot. The **complete** flag must be set to **true**.

```
$ oc get vmrestore <my-vmrestore>
```

Example output

```
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineRestore
metadata:
  creationTimestamp: "2020-09-30T14:46:27Z"
  generation: 5
  name: my-vmrestore
  namespace: default
  ownerReferences:
  - apiVersion: kubevirt.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: VirtualMachine
    name: my-vm
    uid: 355897f3-73a0-4ec4-83d3-3c2df9486f4f
  resourceVersion: "5512"
  selfLink:
  /apis/snapshot.kubevirt.io/v1alpha1/namespaces/default/virtualmachinerestores/my-
  vmrestore
  uid: 71c679a8-136e-46b0-b9b5-f57175a6a041
  spec:
    target:
      apiGroup: kubevirt.io
      kind: VirtualMachine
      name: my-vm
  virtualMachineSnapshotName: my-vmsnapshot
  status:
    complete: true ①
    conditions:
    - lastProbeTime: null
      lastTransitionTime: "2020-09-30T14:46:28Z"
      reason: Operation complete
      status: "False" ②
      type: Progressing
    - lastProbeTime: null
      lastTransitionTime: "2020-09-30T14:46:28Z"
      reason: Operation complete
      status: "True" ③
      type: Ready
    deletedDataVolumes:
```



```

- test-dv1
restoreTime: "2020-09-30T14:46:28Z"
restores:
- dataVolumeName: restore-71c679a8-136e-46b0-b9b5-f57175a6a041-datavolumedisk1
persistentVolumeClaim: restore-71c679a8-136e-46b0-b9b5-f57175a6a041-
datavolumedisk1
volumeName: datavolumedisk1
volumeSnapshotName: vmsnapshot-28eedf08-5d6a-42c1-969c-2eda58e2a78d-volume-
datavolumedisk1


```

- 1 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.

8.18.11.6. Deleting a virtual machine snapshot in the web console

You can delete an existing virtual machine snapshot by using the web console.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Snapshots** tab. The page displays a list of snapshots associated with the virtual machine.
5. Choose one of the following methods to delete a virtual machine snapshot:
 - a. Click the Options menu  of the virtual machine snapshot that you want to delete and select **Delete Virtual Machine Snapshot**
 - b. Select a snapshot to open the **Snapshot Details** screen and click **Actions** → **Delete Virtual Machine Snapshot**.
6. In the confirmation pop-up window, click **Delete** to delete the snapshot.

8.18.11.7. 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. The snapshot controller deletes the **VirtualMachineSnapshot** along with the associated **VirtualMachineSnapshotContent** object.

```
$ oc delete vmsnapshot <my-vmsnapshot>
```

Verification

- Verify that the snapshot is deleted and no longer attached to this VM:

```
$ oc get vmsnapshot
```

8.18.11.8. Additional resources

- [CSI Volume Snapshots](#)

8.18.12. Moving a local virtual machine disk to a different node

Virtual machines that use local volume storage can be moved so that they run on a specific node.

You might want to move the virtual machine to a specific node for the following reasons:

- The current node has limitations to the local storage configuration.
- The new node is better optimized for the workload of that virtual machine.

To move a virtual machine that uses local storage, you must clone the underlying volume by using a data volume. After the cloning operation is complete, you can [edit the virtual machine configuration](#) so that it uses the new data volume, or [add the new data volume to another virtual machine](#).

TIP

When you enable preallocation globally, or for a single data volume, the Containerized Data Importer (CDI) preallocates disk space during cloning. Preallocation enhances write performance. For more information, see [Using preallocation for data volumes](#).



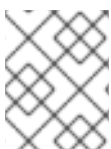
NOTE

Users without the **cluster-admin** role require [additional user permissions](#) to clone volumes across namespaces.

8.18.12.1. Cloning a local volume to another node

You can move a virtual machine disk so that it runs on a specific node by cloning the underlying persistent volume claim (PVC).

To ensure the virtual machine disk is cloned to the correct node, you must either create a new persistent volume (PV) or identify one on the correct node. Apply a unique label to the PV so that it can be referenced by the data volume.



NOTE

The destination PV must be the same size or larger than the source PVC. If the destination PV is smaller than the source PVC, the cloning operation fails.

Prerequisites

- The virtual machine must not be running. Power down the virtual machine before cloning the virtual machine disk.

Procedure

1. Either create a new local PV on the node, or identify a local PV already on the node:
 - Create a local PV that includes the **nodeAffinity.nodeSelectorTerms** parameters. The following manifest creates a **10Gi** local PV on **node01**.

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: <destination-pv> 1
  annotations:
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 10Gi 2
  local:
    path: /mnt/local-storage/local/disk1 3
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
            - node01 4
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local
  volumeMode: Filesystem
```

- 1** The name of the PV.
 - 2** The size of the PV. You must allocate enough space, or the cloning operation fails. The size must be the same as or larger than the source PVC.
 - 3** The mount path on the node.
 - 4** The name of the node where you want to create the PV.
- Identify a PV that already exists on the target node. You can identify the node where a PV is provisioned by viewing the **nodeAffinity** field in its configuration:

```
$ oc get pv <destination-pv> -o yaml
```

The following snippet shows that the PV is on **node01**:

Example output

```

...
spec:
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname ❶
          operator: In
          values:
            - node01 ❷
...

```

❶ The **kubernetes.io/hostname** key uses the node hostname to select a node.

❷ The hostname of the node.

2. Add a unique label to the PV:

```
$ oc label pv <destination-pv> node=node01
```

3. Create a data volume manifest that references the following:

- The PVC name and namespace of the virtual machine.
- The label you applied to the PV in the previous step.
- The size of the destination PV.

```

apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <clone-datavolume> ❶
spec:
  source:
    pvc:
      name: "<source-vm-disk>" ❷
      namespace: "<source-namespace>" ❸
  pvc:
    accessModes:
      - ReadWriteOnce
    selector:
      matchLabels:
        node: node01 ❹
    resources:
      requests:
        storage: <10Gi> ❺

```

❶ The name of the new data volume.

❷ The name of the source PVC. If you do not know the PVC name, you can find it in the virtual machine configuration: **spec.volumes.persistentVolumeClaim.claimName**.

❸ The namespace where the source PVC exists.

- 4 The label that you applied to the PV in the previous step.
- 5 The size of the destination PV.

4. Start the cloning operation by applying the data volume manifest to your cluster:

```
$ oc apply -f <clone-datavolume.yaml>
```

The data volume clones the PVC of the virtual machine into the PV on the specific node.

8.18.13. Expanding virtual storage by adding blank disk images

You can increase your storage capacity or create new data partitions by adding blank disk images to OpenShift Virtualization.

8.18.13.1. About data volumes

DataVolume objects are custom resources that are provided by the Containerized Data Importer (CDI) project. Data volumes orchestrate import, clone, and upload operations that are associated with an underlying persistent volume claim (PVC). Data volumes are integrated with OpenShift Virtualization, and they prevent a virtual machine from being started before the PVC has been prepared.

8.18.13.2. Creating a blank disk image with data volumes

You can create a new blank disk image in a persistent volume claim by customizing and deploying a data volume configuration file.

Prerequisites

- At least one available persistent volume.
- Install the OpenShift CLI (**oc**).

Procedure

1. Edit the data volume configuration file:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: blank-image-datavolume
spec:
  source:
    blank: {}
  pvc:
    # Optional: Set the storage class or omit to accept the default
    # storageClassName: "hostpath"
    accessModes:
      - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

2. Create the blank disk image by running the following command:

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

8.18.13.3. Template: Data volume configuration file for blank disk images

blank-image-datavolume.yaml

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: blank-image-datavolume
spec:
  source:
    blank: {}
  pvc:
    # Optional: Set the storage class or omit to accept the default
    # storageClassName: "hostpath"
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

8.18.13.4. Additional resources

- [Configure preallocation mode](#) to improve write performance for data volume operations.

8.18.14. Cloning a data volume using smart-cloning

Smart-cloning is a built-in feature of OpenShift Container Platform Storage (OCS), designed to enhance performance of the cloning process. Clones created with smart-cloning are faster and more efficient than host-assisted cloning.

You do not need to perform any action to enable smart-cloning, but you need to ensure your storage environment is compatible with smart-cloning to use this feature.

When you create a data volume with a persistent volume claim (PVC) source, you automatically initiate the cloning process. You always receive a clone of the data volume, if your environment supports smart-cloning or not. However, you will only receive the performance benefits of smart cloning if your storage provider supports smart-cloning.

8.18.14.1. Understanding smart-cloning

When a data volume is smart-cloned, the following occurs:

1. A snapshot of the source persistent volume claim (PVC) is created.
2. A PVC is created from the snapshot.
3. The snapshot is deleted.

8.18.14.2. Cloning a data volume

Prerequisites

For smart-cloning to occur, the following conditions are required:

- Your storage provider must support snapshots.
- The source and target PVCs must be defined to the same storage class.
- The **VolumeSnapshotClass** object must reference the storage class defined to both the source and target PVCs.

Procedure

To initiate cloning of a data volume:

1. Create a YAML file for a **DataVolume** object that specifies the name of the new data volume and the name and namespace of the source PVC. In this example, because you specify the **storage** API, there is no need to specify **accessModes** or **volumeMode**. The optimal values will be calculated for you automatically.

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <cloner-datavolume> 1
spec:
  source:
    pvc:
      namespace: "<source-namespace>" 2
      name: "<my-favorite-vm-disk>" 3
  storage: 4
  resources:
    requests:
      storage: <2Gi> 5
```

- 1 The name of the new data volume.
- 2 The namespace where the source PVC exists.
- 3 The name of the source PVC.
- 4 Specifies allocation with the **storage** API
- 5 The size of the new data volume.

2. Start cloning the PVC by creating the data volume:

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



NOTE

Data volumes prevent a virtual machine from starting before the PVC is prepared, so you can create a virtual machine that references the new data volume while the PVC clones.

8.18.14.3. Additional resources

- [Cloning the persistent volume claim of a virtual machine disk into a new data volume](#)
- [Configure preallocation mode](#) to improve write performance for data volume operations.

8.18.15. Creating and using boot sources

A boot source contains a bootable operating system (OS) and all of the configuration settings for the OS, such as drivers.

You use a boot source to create virtual machine templates with specific configurations. These templates can be used to create any number of available virtual machines.

Quick Start tours are available in the OpenShift Container Platform web console to assist you in creating a custom boot source, uploading a boot source, and other tasks. Select **Quick Starts** from the **Help** menu to view the Quick Start tours.

8.18.15.1. About virtual machines and boot sources

Virtual machines consist of a virtual machine definition and one or more disks that are backed by data volumes. Virtual machine templates enable you to create virtual machines using predefined virtual machine specifications.

Every virtual machine template requires a boot source, which is a fully configured virtual machine disk image including configured drivers. Each virtual machine template contains a virtual machine 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) are created with the cluster's default storage class. If you select a different default storage class after configuration, you must delete the existing data volumes in the cluster namespace that are configured with the previous default storage class.

To use the boot sources feature, install the latest release of OpenShift Virtualization. The namespace **openshift-virtualization-os-images** enables the feature and is installed with the OpenShift Virtualization Operator. Once the boot source feature is installed, you can create boot sources, attach them to templates, and create virtual machines from the templates.

Define a boot source by using a persistent volume claim (PVC) that is populated by uploading a local file, cloning an existing PVC, importing from a registry, or by URL. Attach a boot source to a virtual machine template by using the web console. After the boot source is attached to a virtual machine template, you create any number of fully configured ready-to-use virtual machines from the template.

8.18.15.2. Importing a Red Hat Enterprise Linux image as a boot source

You can import a Red Hat Enterprise Linux (RHEL) image as a boot source by specifying the URL address for the image.

Prerequisites

- You must have access to the web server with the operating system image. For example: Red Hat Enterprise Linux web page with images.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Templates** tab.
3. Identify the RHEL template for which you want to configure a boot source and click **Add source**.
4. In the **Add boot source to template** window, select **Import via URL (creates PVC)** from the **Boot source type** list.
5. Click **RHEL download page** to access the Red Hat Customer Portal. A list of available installers and images is displayed on the Download Red Hat Enterprise Linux page.
6. Identify the Red Hat Enterprise Linux KVM guest image that you want to download. Right-click **Download Now**, and copy the URL for the image.
7. In the **Add boot source to template** window, paste the copied URL of the guest image into the **Import URL** field, and click **Save and import**.

Verification

To verify that a boot source was added to the template:

1. Click the **Templates** tab.
2. Confirm that the tile for this template displays a green checkmark.

You can now use this template to create RHEL virtual machines.

8.18.15.3. Adding a boot source for a virtual machine template

A boot source can be configured for any virtual machine template that you want to use for creating virtual machines or custom templates. When virtual machine templates are configured with a boot source, they are labeled **Available** in the **Templates** tab. After you add a boot source to a template, you can create a new virtual machine from the template.

There are four methods for selecting and adding a boot source in the web console:

- **Upload local file (creates PVC)**
- **Import via URL (creates PVC)**
- **Clone existing PVC (creates PVC)**
- **Import via Registry (creates PVC)**

Prerequisites

- To add a boot source, you must be logged in as a user with the **os-images.kubevirt.io:edit** RBAC role or as an administrator. You do not need special privileges to create a virtual machine from a template with a boot source added.
- To upload a local file, the operating system image file must exist on your local machine.
- To import via URL, access to the web server with the operating system image is required. For example: the Red Hat Enterprise Linux web page with images.

- To clone an existing PVC, access to the project with a PVC is required.
- To import via registry, access to the container registry is required.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Templates** tab.
3. Identify the virtual machine template for which you want to configure a boot source and click **Add source**.
4. In the **Add boot source to template** window, click **Select boot source**, select a method for creating a persistent volume claim (PVC): **Upload local file**, **Import via URL**, **Clone existing PVC**, or **Import via Registry**.
5. Optional: Click **This is a CD-ROM boot source** to mount a CD-ROM and use it to install the operating system on to an empty disk. The additional empty disk is automatically created and mounted by OpenShift Virtualization. If the additional disk is not needed, you can remove it when you create the virtual machine.
6. Enter a value for **Persistent Volume Claim size** to specify the PVC size that is adequate for the uncompressed image and any additional space that is required.
 - a. Optional: Enter a name for **Source provider** to associate the name with this template.
 - b. Optional: **Advanced Storage settings**: Click **Storage class** and select the storage class that is used to create the disk. Typically, this storage class is the default storage class that is created for use by all PVCs.



NOTE

Provided boot sources are updated automatically to the latest version of the operating system. For auto-updated boot sources, persistent volume claims (PVCs) are created with the cluster's default storage class. If you select a different default storage class after configuration, you must delete the existing data volumes in the cluster namespace that are configured with the previous default storage class.

- c. Optional: **Advanced Storage settings**: Click **Access mode** and select an access mode for the persistent volume:
 - **Single User (RWO)** mounts the volume as read-write by a single node.
 - **Shared Access (RWX)** mounts the volume as read-write by many nodes.
 - **Read Only (ROX)** mounts the volume as read-only by many nodes.
 - d. Optional: **Advanced Storage settings**: Click **Volume mode** if you want to select **Block** instead of the default value **Filesystem**. OpenShift Virtualization can statically provision raw block volumes. These volumes do not have a file system, and can provide performance benefits for applications that either write to the disk directly or implement their own storage service.
7. Select the appropriate method to save your boot source:

- a. Click **Save and upload** if you uploaded a local file.
- b. Click **Save and import** if you imported content from a URL or the registry.
- c. Click **Save and clone** if you cloned an existing PVC.

Your custom virtual machine template with a boot source is listed in the **Templates** tab, and you can create virtual machines by using this template.

8.18.15.4. Creating a virtual machine from a template with an attached boot source

After you add a boot source to a template, you can create a new virtual machine from the template.

Procedure

1. In the OpenShift Container Platform web console, click **Workloads > Virtualization** in the side menu.
2. From the **Virtual Machines** tab or the **Templates** tab, click **Create** and select **Virtual Machine with Wizard**.
3. In the **Select a template** step, select an OS from the Operating System list that has the **(Source available)** label next to the OS and version name. The **(Source available)** label indicates that a boot source is available for this OS.
4. Click **Review and Confirm**.
5. Review your virtual machine settings and edit them, if required.
6. Click **Create Virtual Machine** to create your virtual machine. The **Successfully created virtual machine** page is displayed.

8.18.15.5. Creating a custom boot source

You can prepare a custom disk image, based on an existing disk image, for use as a boot source.

Use this procedure to complete the following tasks:

- Preparing a custom disk image
- Creating a boot source from the custom disk image
- Attaching the boot source to a custom template

Procedure

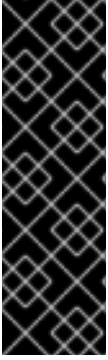
1. In the OpenShift Virtualization console, click **Workloads > Virtualization** from the side menu.
2. Click the **Templates** tab.
3. Click the link in the **Source provider** column for the template you want to customize. A window displays, indicating that the template currently has a **defined** source.
4. In the window, click the **Customize source** link.

5. Click **Continue** in the **About boot source customization** window to proceed with customization after reading the information provided about the boot source customization process.
6. On the **Prepare boot source customization** page, in the **Define new template** section:
 - a. Select the **New template namespace** field and then choose a project.
 - b. Enter the name of the custom template in the **New template name** field.
 - c. Enter the name of the template provider in the **New template provider** field.
 - d. Select the **New template support** field and then choose the appropriate value, indicating support contacts for the custom template you create.
 - e. Select the **New template flavor** field and then choose the appropriate CPU and memory values for the custom image you create.
7. In the **Prepare boot source for customization** section, customize the **cloud-init** YAML script, if needed, to define login credentials. Otherwise, the script generates default credentials for you.
8. Click **Start Customization**. The customization process begins and the **Preparing boot source customization** page displays, followed by the **Customize boot source** page. The **Customize boot source** page displays the output of the running script. When the script completes, your custom image is available.
9. In the **VNC console**, click **show password** in the **Guest login credentials** section. Your login credentials display.
10. When the image is ready for login, sign in with the **VNC Console** by providing the user name and password displayed in the **Guest login credentials** section.
11. Verify the custom image works as expected. If it does, click **Make this boot source available**.
12. In the **Finish customization and make template available** window, select **I have sealed the boot source so it can be used as a template** and then click **Apply**.
13. On the **Finishing boot source customization** page, wait for the template creation process to complete. Click **Navigate to template details** or **Navigate to template list** to view your customized template, created from your custom boot source.

8.18.15.6. Additional resources

- [Creating virtual machine templates](#)
- [Creating a Microsoft Windows boot source from a cloud image](#)
- [Customizing existing Microsoft Windows boot sources in OpenShift Container Platform](#)
- [Setting a PVC as a boot source for a Microsoft Windows template using the CLI](#)
- [Creating boot sources using automated scripting](#)
- [Creating a boot source automatically within a pod](#)

8.18.16. Hot-plugging virtual disks



IMPORTANT

Hot-plugging virtual disks 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 <https://access.redhat.com/support/offerings/techpreview/>.

Hot-plug and hot-unplug virtual disks when you want to add or remove them without stopping your virtual machine or virtual machine instance. This capability is helpful when you need to add storage to a running virtual machine without incurring down-time.

When you *hot-plug* a virtual disk, you attach a virtual disk to a virtual machine instance while the virtual machine is running.

When you *hot-unplug* a virtual disk, you detach a virtual disk from a virtual machine instance while the virtual machine is running.

Only data volumes and persistent volume claims (PVCs) can be hot-plugged and hot-unplugged. You cannot hot-plug or hot-unplug container disks.



WARNING

When you attach a hot-plugged virtual disk to a virtual machine, you cannot perform live migration on the virtual machine. If you attempt to perform live migration on a virtual machine with a hot-plugged disk attached, live migration fails.

8.18.16.1. Hot-plugging a virtual disk using the CLI

Hot-plug virtual disks that you want to attach to a virtual machine instance (VMI) while a virtual machine is running.

Prerequisites

- You must have a running virtual machine to hot-plug a virtual disk.
- You must have at least one data volume or persistent volume claim (PVC) available for hot-plugging.

Procedure

- Hot-plug a virtual disk by running the following command:

```
$ virtctl addvolume <virtual-machine|virtual-machine-instance> --volume-name=  
<datavolume|PVC> \  
[--persist] [--serial=<label-name>]
```

- Use the optional **--persist** flag to add the hot-plugged disk to the virtual machine specification as a permanently mounted virtual disk. Stop, restart, or reboot the virtual machine to permanently mount the virtual disk. After specifying the **--persist** flag, you can no longer hot-plug or hot-unplug the virtual disk. The **--persist** flag applies to virtual machines, not virtual machine interfaces.
- 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.

8.18.16.2. Hot-unplugging a virtual disk using the CLI

Hot-unplug virtual disks that you want to detach from a virtual machine instance (VMI) while a virtual machine is running.

Prerequisites

- Your virtual machine must be running.
- You must have at least one data volume or persistent volume claim (PVC) available and hot-plugged.

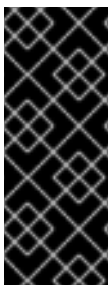
Procedure

- Hot-unplug a virtual disk by running the following command:

```
$ virtctl removevolume <virtual-machine|virtual-machine-instance> --volume-name=  
<datavolume|PVC>
```

8.18.17. Using container disks with virtual machines

You can build a virtual machine image into a container disk and store it in your container registry. You can then import the container disk into persistent storage for a virtual machine or attach it directly to the virtual machine for ephemeral storage.



IMPORTANT

If you use large container disks, I/O traffic might increase, impacting worker nodes. This can lead to unavailable nodes. You can resolve this by:

- [Pruning `DeploymentConfig` objects](#)
- [Configuring garbage collection](#)

8.18.17.1. About container disks

A container disk is a virtual machine image that is stored as a container image in a container image registry. You can use container disks to deliver the same disk images to multiple virtual machines and to create large numbers of virtual machine clones.

A container disk can either be imported into a persistent volume claim (PVC) by using a data volume that is attached to a virtual machine, or attached directly to a virtual machine as an ephemeral **containerDisk** volume.

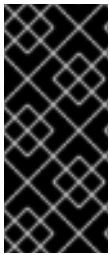
8.18.17.1.1. Importing a container disk into a PVC by using a data volume

Use the Containerized Data Importer (CDI) to import the container disk into a PVC by using a data volume. You can then attach the data volume to a virtual machine for persistent storage.

8.18.17.1.2. Attaching a container disk to a virtual machine as a `containerDisk` volume

A `containerDisk` volume is ephemeral. It is discarded when the virtual machine is stopped, restarted, or deleted. When a virtual machine with a `containerDisk` volume starts, the container image is pulled from the registry and hosted on the node that is hosting the virtual machine.

Use `containerDisk` volumes for read-only file systems such as CD-ROMs or for disposable virtual machines.



IMPORTANT

Using `containerDisk` volumes for read-write file systems is not recommended because the data is temporarily written to local storage on the hosting node. This slows live migration of the virtual machine, such as in the case of node maintenance, because the data must be migrated to the destination node. Additionally, all data is lost if the node loses power or otherwise shuts down unexpectedly.

8.18.17.2. Preparing a container disk for virtual machines

You must build a container disk with a virtual machine image and push it to a container registry before it can be used with a virtual machine. You can then either import the container disk into a PVC using a data volume and attach it to a virtual machine, or you can attach the container disk directly to a virtual machine as an ephemeral `containerDisk` volume.

The size of a disk image inside 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

- Install `podman` if it is not already installed.
- The virtual machine image must be either QCOW2 or RAW format.

Procedure

1. Create a Dockerfile to build the virtual machine image into a container image. The virtual machine 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 virtual machine image in either QCOW2 or RAW format. To use a remote virtual machine image, replace `<vm_image>.qcow2` with the complete url for the remote image.

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
```

If your container registry does not have TLS you must add it as an insecure registry before you can import container disks into persistent storage.

8.18.17.3. Disabling TLS for a container registry to use as insecure 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

- Log in to the cluster as a user with the `cluster-admin` role.

Procedure

- Edit the `HyperConverged` custom resource and add a list of insecure registries to the `spec.storageImport.insecureRegistries` field.

```

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.

8.18.17.4. Next steps

- [Import the container disk into persistent storage for a virtual machine .](#)

- [Create a virtual machine](#) that uses a **containerDisk** volume for ephemeral storage.

8.18.18. Preparing CDI scratch space

8.18.18.1. About data volumes

DataVolume objects are custom resources that are provided by the Containerized Data Importer (CDI) project. Data volumes orchestrate import, clone, and upload operations that are associated with an underlying persistent volume claim (PVC). Data volumes are integrated with OpenShift Virtualization, and they prevent a virtual machine from being started before the PVC has been prepared.

8.18.18.2. Understanding 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.

8.18.18.3. 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.

Type	Reason
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.

8.18.18.4. 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 hco -n openshift-cnv kubevirt-hyperconverged
```

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.

8.18.18.5. 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)	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2** ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2 ✓ GZ* ✓ XZ* 	<ul style="list-style-type: none"> ✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW ✓ GZ ✓ XZ 	<ul style="list-style-type: none"> ✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ 	<ul style="list-style-type: none"> ✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

Unsupported operation

* Requires scratch space

** Requires scratch space if a custom certificate authority is required

8.18.18.6. Additional resources

- [Dynamic provisioning](#)

8.18.19. Re-using persistent volumes

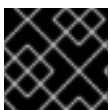
To re-use a statically provisioned persistent volume (PV), you must first reclaim the volume. This involves deleting the PV so that the storage configuration can be re-used.

8.18.19.1. About reclaiming statically provisioned persistent volumes

When you reclaim a persistent volume (PV), you unbind the PV from a persistent volume claim (PVC) and delete the PV. Depending on the underlying storage, you might need to manually delete the shared storage.

You can then re-use the PV configuration to create a PV with a different name.

Statically provisioned PVs must have a reclaim policy of **Retain** to be reclaimed. If they do not, the PV enters a failed state when the PVC is unbound from the PV.



IMPORTANT

The **Recycle** reclaim policy is deprecated in OpenShift Container Platform 4.

8.18.19.2. Reclaiming statically provisioned persistent volumes

Reclaim a statically provisioned persistent volume (PV) by unbinding the persistent volume claim (PVC) and deleting the PV. You might also need to manually delete the shared storage.

Reclaiming a statically provisioned PV is dependent on the underlying storage. This procedure provides a general approach that might need to be customized depending on your storage.

Procedure

1. Ensure that the reclaim policy of the PV is set to **Retain**:

- a. Check the reclaim policy of the PV:

```
$ oc get pv <pv_name> -o yaml | grep 'persistentVolumeReclaimPolicy'
```

- b. If the **persistentVolumeReclaimPolicy** is not set to **Retain**, edit the reclaim policy with the following command:

```
$ oc patch pv <pv_name> -p '{"spec":{"persistentVolumeReclaimPolicy":"Retain"}}'
```

2. Ensure that no resources are using the PV:

```
$ oc describe pvc <pvc_name> | grep 'Mounted By:'
```

Remove any resources that use the PVC before continuing.

3. Delete the PVC to release the PV:

```
$ oc delete pvc <pvc_name>
```

4. Optional: Export the PV configuration to a YAML file. If you manually remove the shared storage later in this procedure, you can refer to this configuration. You can also use **spec** parameters in this file as the basis to create a new PV with the same storage configuration after you reclaim the PV:

```
$ oc get pv <pv_name> -o yaml > <file_name>.yaml
```

5. Delete the PV:

```
$ oc delete pv <pv_name>
```

6. Optional: Depending on the storage type, you might need to remove the contents of the shared storage folder:

```
$ rm -rf <path_to_share_storage>
```

7. Optional: Create a PV that uses the same storage configuration as the deleted PV. If you exported the reclaimed PV configuration earlier, you can use the **spec** parameters of that file as the basis for a new PV manifest:



NOTE

To avoid possible conflict, it is good practice to give the new PV object a different name than the one that you deleted.

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

Additional resources

- [Configuring local storage for virtual machines](#)
- The OpenShift Container Platform Storage documentation has more information on [Persistent Storage](#).

8.18.20. Deleting data volumes

You can manually delete a data volume by using the **oc** command-line interface.



NOTE

When you delete a virtual machine, the data volume it uses is automatically deleted.

8.18.20.1. About data volumes

DataVolume objects are custom resources that are provided by the Containerized Data Importer (CDI) project. Data volumes orchestrate import, clone, and upload operations that are associated with an underlying persistent volume claim (PVC). Data volumes are integrated with OpenShift Virtualization, and they prevent a virtual machine from being started before the PVC has been prepared.

8.18.20.2. Listing all data volumes

You can list the data volumes in your cluster by using the **oc** command-line interface.

Procedure

- List all data volumes by running the following command:

```
$ oc get dvs
```

8.18.20.3. Deleting a data volume

You can delete a data volume by using the **oc** command-line interface (CLI).

Prerequisites

- Identify the name of the data volume that you want to delete.

Procedure

- Delete the data volume by running the following command:

```
$ oc delete dv <datavolume_name>
```



NOTE

This command only deletes objects that exist in the current project. Specify the **-n <project_name>** option if the object you want to delete is in a different project or namespace.

CHAPTER 9. VIRTUAL MACHINE TEMPLATES

9.1. CREATING VIRTUAL MACHINE TEMPLATES

9.1.1. About virtual machine templates

Preconfigured Red Hat virtual machine templates are listed in the **Templates** tab within the **Virtualization** page. These templates are available for different versions of Red Hat Enterprise Linux, Fedora, Microsoft Windows 10, and Microsoft Windows Servers. Each Red Hat virtual machine template is preconfigured with the operating system image, default settings for the operating system, flavor (CPU and memory), and workload type (server).

The **Templates** tab displays four types of virtual machine templates:

- **Red Hat Supported** templates are fully supported by Red Hat.
- **User Supported** templates are **Red Hat Supported** templates that were cloned and created by users.
- **Red Hat Provided** templates have limited support from Red Hat.
- **User Provided** templates are **Red Hat Provided** templates that were cloned and created by users.



NOTE

In the Templates tab, you cannot edit or delete Red Hat Supported or Red Hat Provided templates. You can only edit or delete custom virtual machine templates that were created by users.

Using a Red Hat template is convenient because the template is already preconfigured. When you select a Red Hat template to create your own custom template, the **Create Virtual Machine Template**wizard prompts you to add a boot source if a boot source was not added previously. Then, you can either save your custom template or continue to customize it and save it.

You can also select the **Create Virtual Machine Template**wizard directly and create a custom virtual machine template. The wizard prompts you to provide configuration details for the operating system, flavor, workload type, and other settings. You can add a boot source and continue to customize your template and save it.

9.1.2. About virtual machines and boot sources

Virtual machines consist of a virtual machine definition and one or more disks that are backed by data volumes. Virtual machine templates enable you to create virtual machines using predefined virtual machine specifications.

Every virtual machine template requires a boot source, which is a fully configured virtual machine disk image including configured drivers. Each virtual machine template contains a virtual machine 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) are created with the cluster's default

storage class. If you select a different default storage class after configuration, you must delete the existing data volumes in the cluster namespace that are configured with the previous default storage class.

To use the boot sources feature, install the latest release of OpenShift Virtualization. The namespace **openshift-virtualization-os-images** enables the feature and is installed with the OpenShift Virtualization Operator. Once the boot source feature is installed, you can create boot sources, attach them to templates, and create virtual machines from the templates.

Define a boot source by using a persistent volume claim (PVC) that is populated by uploading a local file, cloning an existing PVC, importing from a registry, or by URL. Attach a boot source to a virtual machine template by using the web console. After the boot source is attached to a virtual machine template, you create any number of fully configured ready-to-use virtual machines from the template.

9.1.3. Adding a boot source for a virtual machine template

A boot source can be configured for any virtual machine template that you want to use for creating virtual machines or custom templates. When virtual machine templates are configured with a boot source, they are labeled **Available** in the **Templates** tab. After you add a boot source to a template, you can create a new virtual machine from the template.

There are four methods for selecting and adding a boot source in the web console:

- **Upload local file (creates PVC)**
- **Import via URL (creates PVC)**
- **Clone existing PVC (creates PVC)**
- **Import via Registry (creates PVC)**

Prerequisites

- To add a boot source, you must be logged in as a user with the **os-images.kubevirt.io:edit** RBAC role or as an administrator. You do not need special privileges to create a virtual machine from a template with a boot source added.
- To upload a local file, the operating system image file must exist on your local machine.
- To import via URL, access to the web server with the operating system image is required. For example: the Red Hat Enterprise Linux web page with images.
- To clone an existing PVC, access to the project with a PVC is required.
- To import via registry, access to the container registry is required.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Templates** tab.
3. Identify the virtual machine template for which you want to configure a boot source and click **Add source**.

4. In the **Add boot source to template** window, click **Select boot source**, select a method for creating a persistent volume claim (PVC): **Upload local file**, **Import via URL**, **Clone existing PVC**, or **Import via Registry**.
5. Optional: Click **This is a CD-ROM boot source** to mount a CD-ROM and use it to install the operating system on to an empty disk. The additional empty disk is automatically created and mounted by OpenShift Virtualization. If the additional disk is not needed, you can remove it when you create the virtual machine.
6. Enter a value for **Persistent Volume Claim size** to specify the PVC size that is adequate for the uncompressed image and any additional space that is required.
 - a. Optional: Enter a name for **Source provider** to associate the name with this template.
 - b. Optional: **Advanced Storage settings**: Click **Storage class** and select the storage class that is used to create the disk. Typically, this storage class is the default storage class that is created for use by all PVCs.



NOTE


Provided boot sources are updated automatically to the latest version of the operating system. For auto-updated boot sources, persistent volume claims (PVCs) are created with the cluster's default storage class. If you select a different default storage class after configuration, you must delete the existing data volumes in the cluster namespace that are configured with the previous default storage class.

- c. Optional: **Advanced Storage settings**: Click **Access mode** and select an access mode for the persistent volume:
 - **Single User (RWO)** mounts the volume as read-write by a single node.
 - **Shared Access (RWX)** mounts the volume as read-write by many nodes.
 - **Read Only (ROX)** mounts the volume as read-only by many nodes.
 - d. Optional: **Advanced Storage settings**: Click **Volume mode** if you want to select **Block** instead of the default value **Filesystem**. OpenShift Virtualization can statically provision raw block volumes. These volumes do not have a file system, and can provide performance benefits for applications that either write to the disk directly or implement their own storage service.
7. Select the appropriate method to save your boot source:
 - a. Click **Save and upload** if you uploaded a local file.
 - b. Click **Save and import** if you imported content from a URL or the registry.
 - c. Click **Save and clone** if you cloned an existing PVC.

Your custom virtual machine template with a boot source is listed in the **Templates** tab, and you can create virtual machines by using this template.

9.1.3.1. Virtual machine template fields for adding a boot source

The following table describes the fields for **Add boot source to template** window. This window displays when you click **Add Source** for a virtual machine template in the **Templates** tab.

Name	Parameter	Description
Boot source type	Upload local file (creates PVC)	Upload a file from your local device. Supported file types include gz, xz, tar, and qcow2.
	Import via URL (creates PVC)	Import content from an image available from an HTTP or HTTPS endpoint. Obtain the download link URL from the web page where the image download is available and enter that URL link in the Import via URL (creates PVC) field. Example: For a Red Hat Enterprise Linux image, log on to the Red Hat Customer Portal, access the image download page, and copy the download link URL for the KVM guest image.
	Clone existing PVC (creates PVC)	Use a PVC that is already available in the cluster and clone it.
	Import via Registry (creates PVC)	Specify the bootable operating system container that is located in a registry and accessible from the cluster. Example: kubevirt/cirros-registry-dis-demo.
Source provider		Optional field. Add descriptive text about the source for the template or the name of the user who created the template. Example: Red Hat.
Advanced	Storage class	The storage class that is used to create the disk.
	Access mode	<p>Access mode of the persistent volume. Supported access modes are Single User (RWO), Shared Access (RWX), Read Only (ROX). If Single User (RWO) is selected, the disk can be mounted as read/write by a single node. If Shared Access (RWX) is selected, the disk can be mounted as read-write by many nodes. The kubevirt-storage-class-defaults config map provides access mode defaults for data volumes. The default value is set according to the best option for each storage class in the cluster.</p> <p>+</p> <div style="display: flex; align-items: flex-start;">  <div> <p>NOTE</p> <p>Shared Access (RWX) is required for some features, such as live migration of virtual machines between nodes.</p> </div> </div>


Name	Parameter	Description
	Volume mode	Defines whether the persistent volume uses a formatted file system or raw block state. Supported modes are Block and Filesystem . The kubevirt-storage-class-defaults config map provides volume mode defaults for data volumes. The default value is set according to the best option for each storage class in the cluster.

9.1.4. Marking virtual machine templates as favorites

For easier access to virtual machine templates that are used frequently, you can mark those templates as favorites.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Templates** tab.
3. Identify the Red Hat template that you want to mark as a favorite.

4. Click the Options menu  and select **Favorite template**. The template moves up higher in the list of displayed templates.

9.1.5. Filtering the list of virtual machine templates by providers

In the **Templates** tab, you can use the **Search by name** field to search for virtual machine templates by specifying either the name of the template or a label that identifies the template. You can also filter templates by the provider, and display only those templates that meet your filtering criteria.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Templates** tab.
3. To filter templates, click **Filter**.
4. Select the appropriate checkbox from the list to filter the templates: **Red Hat Supported**, **User Supported**, **Red Hat Provided**, and **User Provided**.


9.1.6. Creating a virtual machine template with the wizard in the web console

The web console features the **Create Virtual Machine Template** wizard that guides you through the **General**, **Networking**, **Storage**, **Advanced**, and **Review** steps to simplify the process of creating virtual machine templates. All required fields are marked with a *. The **Create Virtual Machine Template** wizard prevents you from moving to the next step until you provide values in the required fields.

**NOTE**

The wizard guides you to create a custom virtual machine template where you specify the operating system, boot source, flavor, and other settings.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Templates** tab.
3. Click **Create** and select **Template with Wizard**
4. Fill in all required fields in the **General** step.
5. Click **Next** to progress to the **Networking** step. A NIC that is named **nic0** is attached by default.
 - a. Optional: Click **Add Network Interface** to create additional NICs.
 - b. Optional: You can remove any or all NICs by clicking the Options menu  and selecting **Delete**. Virtual machines created from a template do not need a NIC attached. NICs can be created after a virtual machine has been created.
6. Click **Next** to progress to the **Storage** step.
7. Click **Add Disk** to add a disk, and complete your selections for the fields in the **Add Disk** screen.

**NOTE**

If **Import via URL (creates PVC)**, **Import via Registry (creates PVC)**, or **Container (ephemeral)** is selected as **Source**, a **rootdisk** disk is created and attached to the virtual machine as the **Bootable Disk**.

A **Bootable Disk** is not required for virtual machines provisioned from a **PXE** source if there are no disks attached to the virtual machine. If one or more disks are attached to the virtual machine, you must select one as the **Bootable Disk**.

Blank disks, PVC disks without a valid boot source, and the cloudinitdisk cannot be used as a boot source.

8. Optional: Click **Advanced** to configure cloud-init and SSH access.

**NOTE**

Statically inject an SSH key by using the custom script in cloud-init or in the wizard. This allows you to securely and remotely manage virtual machines and manage and transfer information. This step is strongly recommended to secure your VM.

9. Click **Review** to review and confirm your settings.
10. Click **Create Virtual Machine template**
11. Click **See virtual machine template details** to view details about the virtual machine template.

The template is also listed in the **Templates** tab.


9.1.7. Virtual machine template wizard fields

The following tables describe the fields for the **General**, **Networking**, **Storage**, and **Advanced** steps in the **Create Virtual Machine Template** wizard.

9.1.7.1. Virtual machine template wizard fields

Name	Parameter	Description
Template		Template from which to create the virtual machine. Selecting a template will automatically complete other fields.
Name		The name can contain lowercase letters (a-z), numbers (0-9), and hyphens (-), up to a maximum of 253 characters. The first and last characters must be alphanumeric. The name must not contain uppercase letters, spaces, periods (.), or special characters.
Template provider		The name of the user who is creating the template for the cluster or any meaningful name that identifies this template.
Template support	No additional support	This template does not have additional support in the cluster.
	Support by template provider	This template is supported by the template provider.
Description		Optional description field.
Operating System		The operating system that is selected for the virtual machine. Selecting an operating system automatically selects the default Flavor and Workload Type for that operating system.
Boot Source	Import via URL (creates PVC)	Import content from an image available from an HTTP or HTTPS endpoint. Example: Obtaining a URL link from the web page with the operating system image.

Name	Parameter	Description
	Clone existing PVC (creates PVC)	Select an existent persistent volume claim available on the cluster and clone it.
	Import via Registry (creates PVC)	Provision virtual machine from a bootable operating system container located in a registry accessible from the cluster. Example: kubevirt/cirros-registry-disk-demo .
	PXE (network boot - adds network interface)	Boot an operating system from a server on the network. Requires a PXE bootable network attachment definition.
Persistent Volume Claim project		Project name that you want to use for cloning the PVC.
Persistent Volume Claim name		PVC name that should apply to this virtual machine template if you are cloning an existing PVC.
Mount this as a CD-ROM boot source		A CD-ROM requires an additional disk for installing the operating system. Select the checkbox to add a disk and customize it later.
Flavor	Tiny, Small, Medium, Large, Custom	<p>Presets the amount of CPU and memory in a virtual machine template with predefined values that are allocated to the virtual machine, depending on the operating system associated with that template.</p> <p>If you choose a default template, you can override the cpus and memsize values in the template using custom values to create a custom template. Alternatively, you can create a custom template by modifying the cpus and memsize values in the Details tab on the Workloads → Virtualization page.</p>

Name	Parameter	Description
Workload Type  NOTE If you choose the incorrect Workload Type , there could be performance or resource utilization issues (such as a slow UI).	Desktop	A virtual machine configuration for use on a desktop. Ideal for consumption on a small scale. Recommended for use with the web console.
	Server	Balances performance and it is compatible with a wide range of server workloads.
	High-Performance	A virtual machine configuration that is optimized for high-performance workloads.

9.1.7.2. 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 .
Network	List of available network attachment definitions.
Type	List of available binding methods. For the default pod network, masquerade is the only recommended binding method. For secondary networks, use the bridge binding method. The masquerade method is not supported for non-default networks. Select SR-IOV if you configured an SR-IOV network device and defined that network in the namespace.
MAC Address	MAC address for the network interface controller. If a MAC address is not specified, one is assigned automatically.


9.1.7.3. Storage fields

Name	Selection	Description
Source	Blank (creates PVC)	Create an empty disk.
	Import via URL (creates PVC)	Import content via URL (HTTP or HTTPS endpoint).
	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 → Volume Mode		Defines whether the persistent volume uses a formatted file system or raw block state. Default is Filesystem .

Name	Selection	Description
Advanced → Access Mode		Access mode of the persistent volume. Supported access modes are Single User (RWO) , Shared Access (RWX) , and Read Only (ROX) .

Advanced storage settings

The following advanced storage settings are available for **Blank**, **Import via URL**, and **Clone existing PVC** disks. These parameters are optional. If you do not specify these parameters, the system uses the default values from the **kubevirt-storage-class-defaults** config map.

Name	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	Single User (RWO)	The disk can be mounted as read/write by a single node.
	Shared Access (RWX)	The disk can be mounted as read/write by many nodes.  NOTE This is required for some features, such as live migration of virtual machines between nodes.
	Read Only (ROX)	The disk can be mounted as read-only by many nodes.

9.1.7.4. Cloud-init fields

Name	Description
Hostname	Sets a specific hostname for the virtual machine.
Authorized SSH Keys	The user's public key that is copied to <code>~/.ssh/authorized_keys</code> on the virtual machine.

Name	Description
Custom script	Replaces other options with a field in which you paste a custom cloud-init script.

9.1.8. Additional resources

- [Configuring the SR-IOV Network Operator](#)
- [Creating and using boot sources](#)

9.2. EDITING VIRTUAL MACHINE TEMPLATES

You can update a virtual machine template in the web console, either by editing the full configuration in the YAML editor or by selecting a custom template in the **Templates** tab and modifying the editable items.

9.2.1. Editing a virtual machine template in the web console

Edit select values of a virtual machine template in the web console by clicking the pencil icon next to the relevant field. Other values can be edited using the CLI.

Labels and annotations are editable for both preconfigured Red Hat templates and your custom virtual machine templates. All other values are editable only for custom virtual machine templates that users have created using the Red Hat templates or the **Create Virtual Machine Template**wizard.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Templates** tab.
3. Select a virtual machine template.
4. Click the **VM Template Details** tab.
5. Click the pencil icon to make a field editable.
6. Make the relevant changes and click **Save**.

Editing a virtual machine template will not affect virtual machines already created from that template.

9.2.2. Editing virtual machine template YAML configuration in the web console

You can edit the YAML configuration of a virtual machine template from the web console.

Some parameters cannot be modified. If you click **Save** with an invalid configuration, an error message indicates the parameter that cannot be modified.



NOTE

Navigating away from the YAML screen while editing cancels any changes to the configuration that you made.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Templates** tab.
3. Select a template to open the **VM Template Details** screen.
4. Click the **YAML** tab to display the editable configuration.
5. Edit the file and click **Save**.

A confirmation message, which includes the updated version number for the object, shows that the YAML configuration was successfully edited.

9.2.3. Adding a virtual disk to a virtual machine template

Use this procedure to add a virtual disk to a virtual machine template.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Templates** tab.
3. Select a virtual machine template to open the **VM Template Details** screen.
4. Click the **Disks** tab.
5. In the **Add Disk** window, specify the **Source**, **Name**, **Size**, **Type**, **Interface**, and **Storage Class**.
 - a. Optional: In the **Advanced** list, specify 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.
6. Click **Add**.

9.2.4. Adding a network interface to a virtual machine template

Use this procedure to add a network interface to a virtual machine template.


Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Templates** tab.
3. Select a virtual machine template to open the **VM Template Details** screen.
4. Click the **Network Interfaces** tab.
5. Click **Add Network Interface**.
6. In the **Add Network Interface** window, specify the **Name**, **Model**, **Network**, **Type**, and **MAC Address** of the network interface.
7. Click **Add**.

9.2.5. Editing CD-ROMs for Templates

Use the following procedure to edit CD-ROMs for virtual machine templates.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Templates** tab.
3. Select a virtual machine template to open the **VM Template Details** screen.
4. Click the **Disks** tab.
5. Click the Options menu  for the CD-ROM that you want to edit and select **Edit**.
6. In the **Edit CD-ROM** window, edit the fields: **Source**, **Persistent Volume Claim**, **Name**, **Type**, and **Interface**.
7. Click **Save**.

9.3. ENABLING DEDICATED RESOURCES FOR VIRTUAL MACHINE TEMPLATES

Virtual machines can have resources of a node, such as CPU, dedicated to them to improve performance.

9.3.1. 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.

9.3.2. Prerequisites

- The [CPU Manager](#) must be configured on the node. Verify that the node has the **cpumanager = true** label before scheduling virtual machine workloads.

9.3.3. Enabling dedicated resources for a virtual machine template

You can enable dedicated resources for a virtual machine template in the **Details** tab. Virtual machines that were created by using a Red Hat template or the wizard can be enabled with dedicated resources.

Procedure

1. Click **Workloads** → **Virtual Machine Templates** from the side menu.
2. Select a virtual machine template to open the **Virtual Machine Template** tab.
3. Click the **Details** tab.
4. Click the pencil icon to the right of the **Dedicated Resources** field to open the **Dedicated Resources** window.

5. Select **Schedule this workload with dedicated resources (guaranteed policy)**
6. Click **Save**.

9.4. DELETING A VIRTUAL MACHINE TEMPLATE

Red Hat virtual machine templates cannot be deleted. You can use the web console to delete:

- Virtual machine templates created from Red Hat templates
- Custom virtual machine templates that were created by using the **Create Virtual Machine Template** wizard.

9.4.1. Deleting a virtual machine template in the web console

Deleting a virtual machine template permanently removes it from the cluster.




NOTE

You can delete virtual machine templates that were created by using a Red Hat template or the **Create Virtual Machine Template** wizard. Preconfigured virtual machine templates that are provided by Red Hat cannot be deleted.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Templates** tab. Select the appropriate method to delete a virtual machine template:

- Click the Options menu  of the template to delete and select **Delete Template**.
- Click the template name to open the **Virtual Machine Template Details** screen and click **Actions** → **Delete Template**.

3. In the confirmation pop-up window, click **Delete** to permanently delete the template.

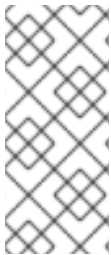
CHAPTER 10. LIVE MIGRATION

10.1. VIRTUAL MACHINE LIVE MIGRATION

10.1.1. Understanding live migration

Live migration is the process of moving a running virtual machine instance (VMI) to another node in the cluster without interrupting the virtual workload or access. If a VMI uses the **LiveMigrate** eviction strategy, it automatically migrates when the node that the VMI runs on is placed into maintenance mode. You can also manually start live migration by selecting a VMI to migrate.

Virtual machines must have a persistent volume claim (PVC) with a shared ReadWriteMany (RWX) access mode to be live migrated.



NOTE

Live migration is supported for virtual machines that are attached to an SR-IOV network interface only if the **sriovLiveMigration** feature gate is enabled in the HyperConverged Cluster custom resource (CR). When the **spec.featureGates.sriovLiveMigration** field is set to **true**, the **virt-launcher** pod runs with the **SYS_RESOURCE** capability. This might degrade its security.

10.1.2. Updating access mode for live migration

For live migration to function properly, you must use the ReadWriteMany (RWX) access mode. Use this procedure to update the access mode, if needed.

Procedure

- To set the RWX access mode, run the following **oc patch** command:

```
$ oc patch -n openshift-cnv \
  cm kubevirt-storage-class-defaults \
  -p '{"data":{"$<STORAGE_CLASS>'.accessMode':"ReadWriteMany"}}'
```

Additional resources:

- [Migrating a virtual machine instance to another node](#)
- [Live migration limiting](#)
- [Customizing the storage profile](#)

10.2. LIVE MIGRATION LIMITS AND TIMEOUTS

Apply live migration limits and timeouts so that migration processes do not overwhelm the cluster. Configure these settings by editing the **HyperConverged** custom resource (CR).

10.2.1. 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.

Procedure

- Edit the **HyperConverged** CR and add the necessary live migration parameters.

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

Example configuration file

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  liveMigrationConfig: 1
    bandwidthPerMigration: 64Mi
    completionTimeoutPerGiB: 800
    parallelMigrationsPerCluster: 5
    parallelOutboundMigrationsPerNode: 2
    progressTimeout: 150
```

- 1 In this example, the **spec.liveMigrationConfig** array contains the default values for each field.



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**.

10.2.2. Cluster-wide live migration limits and timeouts

Table 10.1. Migration parameters

Parameter	Description	Default
parallelMigrationsPerCluster	Number of migrations running in parallel in the cluster.	5
parallelOutboundMigrationsPerNode	Maximum number of outbound migrations per node.	2
bandwidthPerMigration	Bandwidth limit of each migration, in MiB/s.	0 [1]
completionTimeoutPerGiB	The migration is canceled if it has not completed in this time, in seconds per GiB of memory. For example, a virtual machine instance 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.	800

Parameter	Description	Default
progressTimeout	The migration is canceled if memory copy fails to make progress in this time, in seconds.	150

1. The default value of **0** is unlimited.

10.3. MIGRATING A VIRTUAL MACHINE INSTANCE TO ANOTHER NODE

Manually initiate a live migration of a virtual machine instance to another node using either the web console or the CLI.

10.3.1. Initiating live migration of a virtual machine instance in the web console

Migrate a running virtual machine instance to a different node in the cluster.




NOTE

The **Migrate Virtual Machine** action is visible to all users but only admin users can initiate a virtual machine migration.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. You can initiate the migration from this screen, which makes it easier to perform actions on multiple virtual machines in the one screen, or from the **Virtual Machine Overview** screen where you can view comprehensive details of the selected virtual machine:

- Click the Options menu  at the end of virtual machine and select **Migrate Virtual Machine**.
- Click the virtual machine name to open the **Virtual Machine Overview** screen and click **Actions** → **Migrate Virtual Machine**

4. Click **Migrate** to migrate the virtual machine to another node.

10.3.2. Initiating live migration of a virtual machine instance in the CLI

Initiate a live migration of a running virtual machine instance by creating a **VirtualMachineInstanceMigration** object in the cluster and referencing the name of the virtual machine instance.

Procedure

1. Create a **VirtualMachineInstanceMigration** configuration file for the virtual machine instance to migrate. For example, **vmi-migrate.yaml**:

```
apiVersion: kubevirt.io/v1
```

```
kind: VirtualMachineInstanceMigration
metadata:
  name: migration-job
spec:
  vmiName: vmi-fedora
```

2. Create the object in the cluster by running the following command:

```
$ oc create -f vmi-migrate.yaml
```

The **VirtualMachineInstanceMigration** object triggers a live migration of the virtual machine instance. This object exists in the cluster for as long as the virtual machine instance is running, unless manually deleted.

Additional resources:

- [Monitoring live migration of a virtual machine instance](#)
- [Cancelling the live migration of a virtual machine instance](#)

10.4. MONITORING LIVE MIGRATION OF A VIRTUAL MACHINE INSTANCE

You can monitor the progress of a live migration of a virtual machine instance from either the web console or the CLI.

10.4.1. Monitoring live migration of a virtual machine instance in the web console

For the duration of the migration, the virtual machine has a status of **Migrating**. This status is displayed in the **Virtual Machines** tab or in the **Virtual Machine Overview** screen for the migrating virtual machine.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.

10.4.2. Monitoring live migration of a virtual machine instance in the CLI

The status of the virtual machine migration is stored in the **Status** component of the **VirtualMachineInstance** configuration.

Procedure

- Use the **oc describe** command on the migrating virtual machine instance:

```
$ oc describe vmi vmi-fedora
```

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


```

10.5. CANCELLING THE LIVE MIGRATION OF A VIRTUAL MACHINE INSTANCE

Cancel the live migration so that the virtual machine instance remains on the original node.


You can cancel a live migration from either the web console or the CLI.

10.5.1. Cancelling live migration of a virtual machine instance in the web console

You can cancel a live migration of the virtual machine instance using the Options menu  found on each virtual machine in the **Virtualization** → **Virtual Machines** tab, or from the **Actions** menu available on all tabs in the **Virtual Machine Overview** screen.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. You can cancel the migration from this screen, which makes it easier to perform actions on multiple virtual machines, or from the **Virtual Machine Overview** screen where you can view comprehensive details of the selected virtual machine:

- Click the Options menu  at the end of virtual machine and select **Cancel Virtual Machine Migration**.
- Select a virtual machine name to open the **Virtual Machine Overview** screen and click **Actions** → **Cancel Virtual Machine Migration**

4. Click **Cancel Migration** to cancel the virtual machine live migration.

10.5.2. Cancelling live migration of a virtual machine instance in the CLI

Cancel the live migration of a virtual machine instance by deleting the **VirtualMachineInstanceMigration** object associated with the migration.

Procedure

- Delete the **VirtualMachineInstanceMigration** object that triggered the live migration, **migration-job** in this example:

```
$ oc delete vmim migration-job
```

10.6. CONFIGURING VIRTUAL MACHINE EVICTION STRATEGY

The **LiveMigrate** eviction strategy ensures that a virtual machine instance is not interrupted if the node is placed into maintenance or drained. Virtual machines instances with this eviction strategy will be live migrated to another node.

10.6.1. Configuring custom virtual machines with the LiveMigration eviction strategy

You only need to configure the **LiveMigration** eviction strategy on custom virtual machines. Common templates have this eviction strategy configured by default.

Procedure

1. Add the **evictionStrategy: LiveMigrate** option to the **spec.template.spec** section in the virtual machine configuration file. This example uses **oc edit** to update the relevant snippet of the **VirtualMachine** configuration file:

```
$ oc edit vm <custom-vm> -n <my-namespace>
```

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: custom-vm
spec:
  template:
    spec:
      evictionStrategy: LiveMigrate
  ...
```

2. Restart the virtual machine for the update to take effect:

```
$ virtctl restart <custom-vm> -n <my-namespace>
```

CHAPTER 11. NODE MAINTENANCE

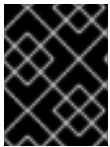
11.1. ABOUT NODE MAINTENANCE

11.1.1. Understanding node maintenance mode

Nodes can be placed into maintenance mode using the **oc adm** utility, or using **NodeMaintenance** custom resources (CRs).

Placing a node into maintenance marks the node as unschedulable and drains all the virtual machines and pods from it. Virtual machine instances that have a **LiveMigrate** eviction strategy are live migrated to another node without loss of service. This eviction strategy is configured by default in virtual machine created from common templates but must be configured manually for custom virtual machines.

Virtual machine instances without an eviction strategy are shut down. Virtual machines with a **RunStrategy** of **Running** or **RerunOnFailure** are recreated on another node. Virtual machines with a **RunStrategy** of **Manual** are not automatically restarted.



IMPORTANT

Virtual machines must have a persistent volume claim (PVC) with a shared **ReadWriteMany** (RWX) access mode to be live migrated.

When installed as part of OpenShift Virtualization, 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.

11.1.2. 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.

Additional resources:


- [About RunStrategies for virtual machines](#)
- [Virtual machine live migration](#)

- [Configuring virtual machine eviction strategy](#)

11.2. SETTING A NODE TO MAINTENANCE MODE


Place a node into maintenance from the web console, CLI, or using a **NodeMaintenance** custom resource.

11.2.1. Setting a node to maintenance mode in the web console

Set a node to maintenance mode using the Options menu  found on each node in the **Compute → Nodes** list, or using the **Actions** control of the **Node Details** screen.

Procedure

1. In the OpenShift Virtualization console, click **Compute → Nodes**.
2. You can set the node to maintenance from this screen, which makes it easier to perform actions on multiple nodes in the one screen or from the **Node Details** screen where you can view comprehensive details of the selected node:

- Click the Options menu  at the end of the node and select **Start Maintenance**.
- Click the node name to open the **Node Details** screen and click **Actions → Start Maintenance**.

3. Click **Start Maintenance** in the confirmation window.

The node will live migrate virtual machine instances that have the **LiveMigration** eviction strategy, and the node is no longer schedulable. All other pods and virtual machines on the node are deleted and recreated on another node.

11.2.2. Setting a node to maintenance mode in the CLI

Set a node to maintenance mode by marking it as unschedulable and using the **oc adm drain** command to evict or delete pods from the node.

Procedure

1. Mark the node as unschedulable. The node status changes to **NotReady,SchedulingDisabled**.

```
$ oc adm cordon <node1>
```

2. Drain the node in preparation for maintenance. The node live migrates virtual machine instances that have the **LiveMigratable** condition set to **True** and the **spec:evictionStrategy** field set to **LiveMigrate**. All other pods and virtual machines on the node are deleted and recreated on another node.

```
$ oc adm drain <node1> --delete-emptydir-data --ignore-daemonsets=true --force
```

- The **--delete-emptydir-data** flag removes any virtual machine instances on the node that use **emptyDir** volumes. Data in these volumes is ephemeral and is safe to be deleted after termination.
- The **--ignore-daemonsets=true** flag ensures that daemon sets are ignored and pod eviction can continue successfully.
- The **--force** flag is required to delete pods that are not managed by a replica set or daemon set controller.

11.2.3. Setting a node to maintenance mode with a NodeMaintenance custom resource

You can put a node into maintenance mode with a **NodeMaintenance** custom resource (CR). When you apply a **NodeMaintenance** CR, all allowed pods are evicted and the node is shut down. Evicted pods are queued to be moved to another node in the cluster.

Prerequisites

- Install the OpenShift Container Platform CLI **oc**.
- Log in to the cluster as a user with **cluster-admin** privileges.

Procedure

1. Create the following node maintenance CR, and save the file as **nodemaintenance-cr.yaml**:

```
apiVersion: nodemaintenance.kubevirt.io/v1beta1
kind: NodeMaintenance
metadata:
  name: maintenance-example 1
spec:
  nodeName: node-1.example.com 2
  reason: "Node maintenance" 3
```

- 1 Node maintenance CR name
- 2 The name of the node to be put into maintenance mode
- 3 Plain text description of the reason for maintenance

2. Apply the node maintenance schedule by running the following command:

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

3. Check the progress of the maintenance task by running the following command, replacing **<node-name>** with the name of your node:

```
$ oc describe node <node-name>
```

Example output

```
Events:
```

Type	Reason	Age	From	Message
Normal	NodeNotSchedulable	61m	kubelet	Node node-1.example.com status is now: NodeNotSchedulable

11.2.3.1. Checking status of current NodeMaintenance CR tasks

You can check the status of current **NodeMaintenance** CR tasks.

Prerequisites

- Install the OpenShift Container Platform CLI **oc**.
- Log in as a user with **cluster-admin** privileges.

Procedure

- Check the status of current node maintenance tasks by running the following command:

```
$ oc get NodeMaintenance -o yaml
```

Example output

```
apiVersion: v1
items:
- apiVersion: nodemaintenance.kubevirt.io/v1beta1
  kind: NodeMaintenance
  metadata:
  ...
  spec:
    nodeName: node-1.example.com
    reason: Node maintenance
  status:
    evictionPods: 3 1
    pendingPods:
    - pod-example-workload-0
    - httpd
    - httpd-manual
    phase: Running
    lastError: "Last failure message" 2
    totalpods: 5
  ...
```

1 **evictionPods** is the number of pods scheduled for eviction.

2 **lastError** records the latest eviction error, if any.

Additional resources:

- [Resuming a node from maintenance mode](#)
- [Virtual machine live migration](#)


- [Configuring virtual machine eviction strategy](#)

11.3. RESUMING A NODE FROM MAINTENANCE MODE

Resuming a node brings it out of maintenance mode and makes it schedulable again.


Resume a node from maintenance mode from the web console, CLI, or by deleting the **NodeMaintenance** custom resource.

11.3.1. Resuming a node from maintenance mode in the web console

Resume a node from maintenance mode using the Options menu  found on each node in the **Compute → Nodes** list, or using the **Actions** control of the **Node Details** screen.

Procedure

1. In the OpenShift Virtualization console, click **Compute → Nodes**.
2. You can resume the node from this screen, which makes it easier to perform actions on multiple nodes in the one screen, or from the **Node Details** screen where you can view comprehensive details of the selected node:

- Click the Options menu  at the end of the node and select **Stop Maintenance**.
- Click the node name to open the **Node Details** screen and click **Actions → Stop Maintenance**.

3. Click **Stop Maintenance** in the confirmation window.

The node becomes schedulable, but virtual machine instances that were running on the node prior to maintenance will not automatically migrate back to this node.

11.3.2. Resuming a node from maintenance mode in the CLI

Resume a node from maintenance mode by making it schedulable again.

Procedure

- Mark the node as schedulable. You can then resume scheduling new workloads on the node.

```
$ oc adm uncordon <node1>
```

11.3.3. Resuming a node from maintenance mode that was initiated with a NodeMaintenance CR

You can resume a node by deleting the **NodeMaintenance** CR.

Prerequisites

- Install the OpenShift Container Platform CLI **oc**.

- Log in to the cluster as a user with **cluster-admin** privileges.

Procedure

- When your node maintenance task is complete, delete the active **NodeMaintenance** CR:

```
$ oc delete -f nodemaintenance-cr.yaml
```

Example output

```
nodemaintenance.nodemaintenance.kubevirt.io "maintenance-example" deleted
```

11.4. AUTOMATIC RENEWAL OF TLS CERTIFICATES

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

11.4.1. TLS certificates 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

11.5. 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.

11.5.1. 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 11.1. 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.

11.5.2. About node labeling for CPU features

Through the process of iteration, the base CPU features in the minimum CPU model are eliminated from the list of labels generated for the node.

For example:

- An environment might have two supported CPU models: **Penryn** and **Haswell**.
- If **Penryn** is specified as the CPU model for **minCPU**, each base CPU feature for **Penryn** is compared to the list of CPU features supported by **Haswell**.

Example 11.2. CPU features supported by Penryn

```

apic
clflush
cmov
cx16
cx8
de
fpu
fxsr
lahf_lm
lm
mca
mce
mmx
msr
mtrr
nx
pae
pat
pge
pni
pse
pse36
sep
sse

```

```
sse2  
sse4.1  
ssse3  
syscall  
tsc
```

Example 11.3. CPU features supported by Haswell

```
aes  
apic  
avx  
avx2  
bmi1  
bmi2  
clflush  
cmov  
cx16  
cx8  
de  
erms  
fma  
fpu  
fsgsbase  
fxsr  
hle  
invpcid  
lahf_lm  
lm  
mca  
mce  
mmx  
movbe  
msr  
mtrr  
nx  
pae  
pat  
pcid  
pclmuldq  
pge  
pni  
popcnt  
pse  
pse36  
rdtscp  
rtm  
sep  
smep  
sse  
sse2  
sse4.1  
sse4.2  
ssse3  
syscall
```

```
tsc
tsc-deadline
x2apic
xsave
```

- If both **Penryn** and **Haswell** support a specific CPU feature, a label is not created for that feature. Labels are generated for CPU features that are supported only by **Haswell** and not by **Penryn**.

Example 11.4. Node labels created for CPU features after iteration

```
aes
avx
avx2
bmi1
bmi2
erms
fma
fsgsbase
hle
invpcid
movbe
pcid
pclmuldq
popcnt
rdtscp
rtm
sse4.2
tsc-deadline
x2apic
xsave
```

11.5.3. 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-cn
spec:
  obsoleteCPUs:
    cpuModels: 1
```

```
- "<obsolete_cpu_1>"  
- "<obsolete_cpu_2>"  
minCPUModel: "<minimum_cpu_model>" 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.
- 2 Replace this value with the minimum CPU model that you want to use for basic CPU features. If you do not specify a value, **Penryn** is used by default.

11.6. PREVENTING NODE RECONCILIATION

Use **skip-node** annotation to prevent the **node-labeller** from reconciling a node.

11.6.1. 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.

11.6.2. Additional resources

- [Managing node labeling for obsolete CPU models](#)

CHAPTER 12. NODE NETWORKING

12.1. OBSERVING NODE NETWORK STATE

Node network state is the network configuration for all nodes in the cluster.

12.1.1. About nmstate

OpenShift Virtualization uses **nmstate** to report on and configure the state of the node network. This makes it possible to modify network policy configuration, such as by creating a Linux bridge on all nodes, by applying a single configuration manifest to the cluster.

Node networking is monitored and updated by the following objects:

NodeNetworkState

Reports the state of the network on that node.

NodeNetworkConfigurationPolicy

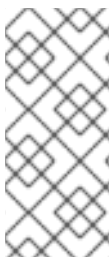
Describes the requested network configuration on nodes. You update the node network configuration, including adding and removing interfaces, by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

NodeNetworkConfigurationEnactment

Reports the network policies enacted upon each node.

OpenShift Virtualization supports the use of the following nmstate interface types:

- Linux Bridge
- VLAN
- Bond
- Ethernet



NOTE

If your OpenShift Container Platform cluster uses OVN-Kubernetes as the default Container Network Interface (CNI) provider, you cannot attach a Linux bridge or bonding to the default interface of a host because of a change in the host network topology of OVN-Kubernetes. As a workaround, you can use a secondary network interface connected to your host, or switch to the OpenShift SDN default CNI provider.

12.1.2. Viewing the network state of a node

A **NodeNetworkState** object exists on every node in the cluster. This object is periodically updated and captures the state of the network for that node.

Procedure

1. List all the **NodeNetworkState** objects in the cluster:

```
$ oc get nns
```

- Inspect a **NodeNetworkState** object to view the network on that node. The output in this example has been redacted for clarity:

```
$ oc get nns node01 -o yaml
```

Example output

```
apiVersion: nmstate.io/v1beta1
kind: NodeNetworkState
metadata:
  name: node01 1
status:
  currentState: 2
  dns-resolver:
  ...
  interfaces:
  ...
  route-rules:
  ...
  routes:
  ...
lastSuccessfulUpdateTime: "2020-01-31T12:14:00Z" 3
```

- The name of the **NodeNetworkState** object is taken from the node.
- The **currentState** contains the complete network configuration for the node, including DNS, interfaces, and routes.
- Timestamp of the last successful update. This is updated periodically as long as the node is reachable and can be used to evaluate the freshness of the report.

12.2. UPDATING NODE NETWORK CONFIGURATION

You can update the node network configuration, such as adding or removing interfaces from nodes, by applying **NodeNetworkConfigurationPolicy** manifests to the cluster.

12.2.1. About nmstate

OpenShift Virtualization uses **nmstate** to report on and configure the state of the node network. This makes it possible to modify network policy configuration, such as by creating a Linux bridge on all nodes, by applying a single configuration manifest to the cluster.

Node networking is monitored and updated by the following objects:

NodeNetworkState

Reports the state of the network on that node.

NodeNetworkConfigurationPolicy

Describes 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.

NodeNetworkConfigurationEnactment

Reports the network policies enacted upon each node.

OpenShift Virtualization supports the use of the following nmstate interface types:

- Linux Bridge
- VLAN
- Bond
- Ethernet



NOTE

If your OpenShift Container Platform cluster uses OVN-Kubernetes as the default Container Network Interface (CNI) provider, you cannot attach a Linux bridge or bonding to the default interface of a host because of a change in the host network topology of OVN-Kubernetes. As a workaround, you can use a secondary network interface connected to your host, or switch to the OpenShift SDN default CNI provider.

12.2.2. Creating an interface on nodes

Create an interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster. The manifest details the requested configuration for the interface.

By default, the manifest applies to all nodes in the cluster. To add the interface to specific nodes, add the **spec: nodeSelector** parameter and the appropriate **<key>:<value>** for your node selector.

Procedure

1. Create the **NodeNetworkConfigurationPolicy** manifest. The following example configures a Linux bridge on all worker nodes:

```
apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: <br1-eth1-policy> 1
spec:
  nodeSelector: 2
    node-role.kubernetes.io/worker: "" 3
desiredState:
  interfaces:
    - name: br1
      description: Linux bridge with eth1 as a port 4
      type: linux-bridge
      state: up
      ipv4:
        dhcp: true
        enabled: true
      bridge:
        options:
          stp:
            enabled: false
      port:
        - name: eth1
```

- 1 Name of the policy.
- 2 Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.
- 3 This example uses the **node-role.kubernetes.io/worker: ""** node selector to select all worker nodes in the cluster.
- 4 Optional: Human-readable description for the interface.

2. Create the node network policy:

```
$ oc apply -f <br1-eth1-policy.yaml> 1
```

- 1 File name of the node network configuration policy manifest.

Additional resources

- [Example policy configurations for different interfaces](#)
- [Example for creating multiple interfaces in the same policy](#)
- [Examples of different IP management methods in policies](#)

12.2.3. Confirming node network policy updates on nodes

A **NodeNetworkConfigurationPolicy** manifest describes your requested network configuration for nodes in the cluster. The node network policy includes your requested network configuration and the status of execution of the policy on the cluster as a whole.

When you apply a node network policy, a **NodeNetworkConfigurationEnactment** object is created for every node in the cluster. The node network configuration enactment is a read-only object that represents the status of execution of the policy on that node. If the policy fails to be applied on the node, the enactment for that node includes a traceback for troubleshooting.

Procedure

1. To confirm that a policy has been applied to the cluster, list the policies and their status:

```
$ oc get nncp
```

2. Optional: If a policy is taking longer than expected to successfully configure, you can inspect the requested state and status conditions of a particular policy:

```
$ oc get nncp <policy> -o yaml
```

3. Optional: If a policy is taking longer than expected to successfully configure on all nodes, you can list the status of the enactments on the cluster:

```
$ oc get nnce
```

4. Optional: To view the configuration of a particular enactment, including any error reporting for a failed configuration:


```
$ oc get nnce <node>.<policy> -o yaml
```

12.2.4. Removing an interface from nodes

You can remove an interface from one or more nodes in the cluster by editing the **NodeNetworkConfigurationPolicy** object and setting the **state** of the interface to **absent**.

Removing an interface from a node does not automatically restore the node network configuration to a previous state. If you want to restore the previous state, you will need to define that node network configuration in the policy.

If you remove a bridge or bonding interface, any node NICs in the cluster that were previously attached or subordinate to that bridge or bonding interface are placed in a **down** state and become unreachable. To avoid losing connectivity, configure the node NIC in the same policy so that it has a status of **up** and either DHCP or a static IP address.



NOTE

Deleting the node network policy that added an interface does not change the configuration of the policy on the node. Although a **NodeNetworkConfigurationPolicy** is an object in the cluster, it only represents the requested configuration. Similarly, removing an interface does not delete the policy.

Procedure

1. Update the **NodeNetworkConfigurationPolicy** manifest used to create the interface. The following example removes a Linux bridge and configures the **eth1** NIC with DHCP to avoid losing connectivity:

```
apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: <br1-eth1-policy> 1
spec:
  nodeSelector: 2
    node-role.kubernetes.io/worker: "" 3
  desiredState:
    interfaces:
      - name: br1
        type: linux-bridge
        state: absent 4
      - name: eth1 5
        type: ethernet 6
        state: up 7
        ipv4:
          dhcp: true 8
          enabled: true 9
```

- 1 Name of the policy.
- 2 Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.

- 3 This example uses the **node-role.kubernetes.io/worker: ""** node selector to select all worker nodes in the cluster.
- 4 Changing the state to **absent** removes the interface.
- 5 The name of the interface that is to be unattached from the bridge interface.
- 6 The type of interface. This example creates an Ethernet networking interface.
- 7 The requested state for the interface.
- 8 Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.
- 9 Enables **ipv4** in this example.

2. Update the policy on the node and remove the interface:

```
$ oc apply -f <br1-eth1-policy.yaml> 1
```

- 1 File name of the policy manifest.

12.2.5. Example policy configurations for different interfaces

12.2.5.1. Example: Linux bridge interface node network configuration policy

Create a Linux bridge interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

The following YAML file is an example of a manifest for a Linux bridge interface. It includes sample values that you must replace with your own information.

```
apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
      - name: br1 4
        description: Linux bridge with eth1 as a port 5
        type: linux-bridge 6
        state: up 7
        ipv4:
          dhcp: true 8
          enabled: true 9
        bridge:
          options:
            stp:
```

```

    enabled: false 10
  port:
    - name: eth1 11

```

- 1 Name of the policy.
- 2 Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.
- 3 This example uses a **hostname** node selector.
- 4 Name of the interface.
- 5 Optional: Human-readable description of the interface.
- 6 The type of interface. This example creates a bridge.
- 7 The requested state for the interface after creation.
- 8 Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.
- 9 Enables **ipv4** in this example.
- 10 Disables **stp** in this example.
- 11 The node NIC to which the bridge attaches.

12.2.5.2. Example: VLAN interface node network configuration policy

Create a VLAN interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

The following YAML file is an example of a manifest for a VLAN interface. It includes samples values that you must replace with your own information.

```

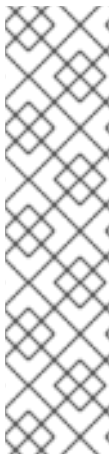
apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: vlan-eth1-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
      - name: eth1.102 4
        description: VLAN using eth1 5
        type: vlan 6
        state: up 7
        vlan:
          base-iface: eth1 8
          id: 102 9

```

- 1 Name of the policy.
- 2 Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.
- 3 This example uses a **hostname** node selector.
- 4 Name of the interface.
- 5 Optional: Human-readable description of the interface.
- 6 The type of interface. This example creates a VLAN.
- 7 The requested state for the interface after creation.
- 8 The node NIC to which the VLAN is attached.
- 9 The VLAN tag.

12.2.5.3. Example: Bond interface node network configuration policy

Create a bond interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.



NOTE

OpenShift Virtualization only supports the following bond modes:

- mode=1 active-backup
- mode=2 balance-xor
- mode=4 802.3ad
- mode=5 balance-tlb
- mode=6 balance-alb

The following YAML file is an example of a manifest for a bond interface. It includes sample values that you must replace with your own information.

```
apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond0-eth1-eth2-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
      - name: bond0 4
        description: Bond enslaving eth1 and eth2 5
        type: bond 6
        state: up 7
```

```

ipv4:
  dhcp: true 8
  enabled: true 9
link-aggregation:
  mode: active-backup 10
  options:
    miimon: '140' 11
  slaves: 12
    - eth1
    - eth2
  mtu: 1450 13

```

- 1 Name of the policy.
- 2 Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.
- 3 This example uses a **hostname** node selector.
- 4 Name of the interface.
- 5 Optional: Human-readable description of the interface.
- 6 The type of interface. This example creates a bond.
- 7 The requested state for the interface after creation.
- 8 Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.
- 9 Enables **ipv4** in this example.
- 10 The driver mode for the bond. This example uses an active backup mode.
- 11 Optional: This example uses **miimon** to inspect the bond link every 140ms.
- 12 The subordinate node NICs in the bond.
- 13 Optional: The maximum transmission unit (MTU) for the bond. If not specified, this value is set to **1500** by default.

12.2.5.4. Example: Ethernet interface node network configuration policy

Configure an Ethernet interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

The following YAML file is an example of a manifest for an Ethernet interface. It includes sample values that you must replace with your own information.

```

apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: eth1-policy 1
spec:

```

```

nodeSelector: 2
  kubernetes.io/hostname: <node01> 3
desiredState:
  interfaces:
  - name: eth1 4
    description: Configuring eth1 on node01 5
    type: ethernet 6
    state: up 7
    ipv4:
      dhcp: true 8
      enabled: true 9

```

- 1 Name of the policy.
- 2 Optional: If you do not include the **nodeSelector** parameter, the policy applies to all nodes in the cluster.
- 3 This example uses a **hostname** node selector.
- 4 Name of the interface.
- 5 Optional: Human-readable description of the interface.
- 6 The type of interface. This example creates an Ethernet networking interface.
- 7 The requested state for the interface after creation.
- 8 Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.
- 9 Enables **ipv4** in this example.

12.2.5.5. Example: Multiple interfaces in the same node network configuration policy

You can create multiple interfaces in the same node network configuration policy. These interfaces can reference each other, allowing you to build and deploy a network configuration by using a single policy manifest.

The following example snippet creates a bond that is named **bond10** across two NICs and a Linux bridge that is named **br1** that connects to the bond.

```

...
  interfaces:
  - name: bond10
    description: Bonding eth2 and eth3 for Linux bridge
    type: bond
    state: up
    link-aggregation:
      slaves:
      - eth2
      - eth3
  - name: br1
    description: Linux bridge on bond
    type: linux-bridge

```

```

state: up
bridge:
  port:
    - name: bond10
...

```

12.2.6. Examples: IP management

The following example configuration snippets demonstrate different methods of IP management.

These examples use the **ethernet** interface type to simplify the example while showing the related context in the policy configuration. These IP management examples can be used with the other interface types.

12.2.6.1. Static

The following snippet statically configures an IP address on the Ethernet interface:

```

...
interfaces:
  - name: eth1
    description: static IP on eth1
    type: ethernet
    state: up
    ipv4:
      dhcp: false
      address:
        - ip: 192.168.122.250 1
          prefix-length: 24
          enabled: true
...

```

1 Replace this value with the static IP address for the interface.

12.2.6.2. No IP address

The following snippet ensures that the interface has no IP address:

```

...
interfaces:
  - name: eth1
    description: No IP on eth1
    type: ethernet
    state: up
    ipv4:
      enabled: false
...

```

12.2.6.3. Dynamic host configuration

The following snippet configures an Ethernet interface that uses a dynamic IP address, gateway address, and DNS:

-

```
...
  interfaces:
  - name: eth1
    description: DHCP on eth1
    type: ethernet
    state: up
    ipv4:
      dhcp: true
      enabled: true
  ...
```

The following snippet configures an Ethernet interface that uses a dynamic IP address but does not use a dynamic gateway address or DNS:

```
...
  interfaces:
  - name: eth1
    description: DHCP without gateway or DNS on eth1
    type: ethernet
    state: up
    ipv4:
      dhcp: true
      auto-gateway: false
      auto-dns: false
      enabled: true
  ...
```

12.2.6.4. DNS

The following snippet sets DNS configuration on the host.

```
...
  interfaces:
  ...
  dns-resolver:
  config:
  search:
  - example.com
  - example.org
  server:
  - 8.8.8.8
  ...
```

12.2.6.5. Static routing

The following snippet configures a static route and a static IP on interface **eth1**.

```
...
  interfaces:
  - name: eth1
    description: Static routing on eth1
    type: ethernet
    state: up
    ipv4:
```



```

dhcp: false
address:
- ip: 192.0.2.251 1
  prefix-length: 24
enabled: true
routes:
config:
- destination: 198.51.100.0/24
  metric: 150
  next-hop-address: 192.0.2.1 2
  next-hop-interface: eth1
  table-id: 254
...

```

- 1** The static IP address for the Ethernet interface.
- 2** Next hop address for the node traffic. This must be in the same subnet as the IP address set for the Ethernet interface.

12.3. TROUBLESHOOTING NODE NETWORK CONFIGURATION

If the node network configuration encounters an issue, the policy is automatically rolled back and the enactments report failure. This includes issues such as:

- The configuration fails to be applied on the host.
- The host loses connection to the default gateway.
- The host loses connection to the API server.

12.3.1. Troubleshooting an incorrect node network configuration policy configuration

You can apply changes to the node network configuration across your entire cluster by applying a node network configuration policy. If you apply an incorrect configuration, you can use the following example to troubleshoot and correct the failed node network policy.

In this example, a Linux bridge policy is applied to an example cluster that has three control plane nodes (master) and three compute (worker) nodes. The policy fails to be applied because it references an incorrect interface. To find the error, investigate the available NMState resources. You can then update the policy with the correct configuration.

Procedure

1. Create a policy and apply it to your cluster. The following example creates a simple bridge on the **ens01** interface:

```

apiVersion: nmstate.io/v1beta1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ens01-bridge-testfail
spec:
  desiredState:
    interfaces:
      - name: br1

```

```

description: Linux bridge with the wrong port
type: linux-bridge
state: up
ipv4:
  dhcp: true
  enabled: true
bridge:
  options:
    stp:
      enabled: false
port:
  - name: ens01

```

```
$ oc apply -f ens01-bridge-testfail.yaml
```

Example output

```
nodenetworkconfigurationpolicy.nmstate.io/ens01-bridge-testfail created
```

2. Verify the status of the policy by running the following command:

```
$ oc get nncp
```

The output shows that the policy failed:

Example output

```

NAME                STATUS
ens01-bridge-testfail FailedToConfigure

```

However, the policy status alone does not indicate if it failed on all nodes or a subset of nodes.

3. List the node network configuration enactments to see if the policy was successful on any of the nodes. If the policy failed for only a subset of nodes, it suggests that the problem is with a specific node configuration. If the policy failed on all nodes, it suggests that the problem is with the policy.

```
$ oc get nnce
```

The output shows that the policy failed on all nodes:

Example output

```

NAME                STATUS
control-plane-1.ens01-bridge-testfail FailedToConfigure
control-plane-2.ens01-bridge-testfail FailedToConfigure
control-plane-3.ens01-bridge-testfail FailedToConfigure
compute-1.ens01-bridge-testfail FailedToConfigure
compute-2.ens01-bridge-testfail FailedToConfigure
compute-3.ens01-bridge-testfail FailedToConfigure

```

4. View one of the failed enactments and look at the traceback. The following command uses the output tool **jsonpath** to filter the output:

```
$ oc get nnce compute-1.ens01-bridge-testfail -o jsonpath='{.status.conditions[?(@.type=="Failing")].message}'
```

This command returns a large traceback that has been edited for brevity:

Example output

```
error reconciling NodeNetworkConfigurationPolicy at desired state apply: , failed to execute
nmstatectl set --no-commit --timeout 480: 'exit status 1' "
...
libnmstate.error.NmstateVerificationError:
desired
=====
---
name: br1
type: linux-bridge
state: up
bridge:
  options:
    group-forward-mask: 0
    mac-ageing-time: 300
    multicast-snooping: true
  stp:
    enabled: false
    forward-delay: 15
    hello-time: 2
    max-age: 20
    priority: 32768
  port:
    - name: ens01
description: Linux bridge with the wrong port
ipv4:
  address: []
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  dhcp: true
  enabled: true
ipv6:
  enabled: false
mac-address: 01-23-45-67-89-AB
mtu: 1500

current
=====
---
name: br1
type: linux-bridge
state: up
bridge:
  options:
    group-forward-mask: 0
    mac-ageing-time: 300
    multicast-snooping: true
  stp:
```

```

    enabled: false
    forward-delay: 15
    hello-time: 2
    max-age: 20
    priority: 32768
    port: []
description: Linux bridge with the wrong port
ipv4:
  address: []
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  dhcp: true
  enabled: true
ipv6:
  enabled: false
mac-address: 01-23-45-67-89-AB
mtu: 1500

difference
=====
--- desired
+++ current
@@ -13,8 +13,7 @@
     hello-time: 2
     max-age: 20
     priority: 32768
- port:
- - name: ens01
+ port: []
description: Linux bridge with the wrong port
ipv4:
  address: []
line 651, in _assert_interfaces_equal\n
current_state.interfaces[ifname],\nlibnmstate.error.NmstateVerificationError:

```

The **NmstateVerificationError** lists the **desired** policy configuration, the **current** configuration of the policy on the node, and the **difference** highlighting the parameters that do not match. In this example, the **port** is included in the **difference**, which suggests that the problem is the port configuration in the policy.

- To ensure that the policy is configured properly, view the network configuration for one or all of the nodes by requesting the **NodeNetworkState** object. The following command returns the network configuration for the **control-plane-1** node:

```
$ oc get nns control-plane-1 -o yaml
```

The output shows that the interface name on the nodes is **ens1** but the failed policy incorrectly uses **ens01**:

Example output

```

- ipv4:
...
  name: ens1

```

```
state: up
type: ethernet
```

6. Correct the error by editing the existing policy:

```
$ oc edit nncp ens01-bridge-testfail
```

```
...
  port:
    - name: ens1
```

Save the policy to apply the correction.

7. Check the status of the policy to ensure it updated successfully:

```
$ oc get nncp
```

Example output

```
NAME                STATUS
ens01-bridge-testfail SuccessfullyConfigured
```

The updated policy is successfully configured on all nodes in the cluster.

CHAPTER 13. LOGGING, EVENTS, AND MONITORING

13.1. VIEWING VIRTUAL MACHINE LOGS

13.1.1. Understanding virtual machine logs

Logs are collected for OpenShift Container Platform builds, deployments, and pods. In OpenShift Virtualization, virtual machine logs can be retrieved from the virtual machine launcher pod in either the web console or the CLI.

The **-f** option follows the log output in real time, which is useful for monitoring progress and error checking.

If the launcher pod is failing to start, use the **--previous** option to see the logs of the last attempt.



WARNING

ErrImagePull and **ImagePullBackOff** errors can be caused by an incorrect deployment configuration or problems with the images that are referenced.

13.1.2. Viewing virtual machine logs in the CLI

Get virtual machine logs from the virtual machine launcher pod.

Procedure

- Use the following command:

```
$ oc logs <virt-launcher-name>
```

13.1.3. Viewing virtual machine logs in the web console

Get virtual machine logs from the associated virtual machine launcher pod.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. In the **Details** tab, click the **virt-launcher-<name>** pod in the **Pod** section.
5. Click **Logs**.

13.2. VIEWING EVENTS

13.2.1. Understanding virtual machine events

OpenShift Container Platform events are records of important life-cycle information in a namespace and are useful for monitoring and troubleshooting resource scheduling, creation, and deletion issues.

OpenShift Virtualization adds events for virtual machines and virtual machine instances. These can be viewed from either the web console or the CLI.

See also: [Viewing system event information in an OpenShift Container Platform cluster](#) .

13.2.2. Viewing the events for a virtual machine in the web console

You can view the stream events for a running a virtual machine from the **Virtual Machine Overview** panel of the web console.

The **⏸** button pauses the events stream.

The **▶** button continues a paused events stream.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click **Events** to view all events for the virtual machine.

13.2.3. Viewing namespace events in the CLI

Use the OpenShift Container Platform client to get the events for a namespace.

Procedure

- In the namespace, use the **oc get** command:

```
┆ $ oc get events
```

13.2.4. Viewing resource events in the CLI

Events are included in the resource description, which you can get using the OpenShift Container Platform client.

Procedure

- In the namespace, use the **oc describe** command. The following example shows how to get the events for a virtual machine, a virtual machine instance, and the virt-launcher pod for a virtual machine:

```
┆ $ oc describe vm <vm>
```

```
┆ $ oc describe vmi <vmi>
```

```
$ oc describe pod virt-launcher-<name>
```

13.3. DIAGNOSING DATA VOLUMES USING EVENTS AND CONDITIONS

Use the **oc describe** command to analyze and help resolve issues with data volumes.

13.3.1. About conditions and events

Diagnose data volume issues by examining the output of the **Conditions** and **Events** sections generated by the command:

```
$ oc describe dv <DataVolume>
```

There are three **Types** in the **Conditions** section that display:

- **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 either **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.

13.3.2. Analyzing data volumes using 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

```
---
- name: vm1
  status:
    conditions:
      - lastHeartBeatTime: 2020-07-15T04:31:39Z
        lastTransitionTime: 2020-07-15T04:31:39Z
        status: True
        type: Ready
```

13.4. VIEWING INFORMATION ABOUT VIRTUAL MACHINE WORKLOADS

You can view high-level information about your virtual machines by using the **Virtual Machines** dashboard in the OpenShift Container Platform web console.

13.4.1. About the Virtual Machines dashboard

Access virtual machines from the OpenShift Container Platform web console by navigating to the **Workloads** → **Virtualization** page. The **Workloads** → **Virtualization** page contains two tabs:

- **Virtual Machines**
- **Virtual Machine Templates**

The following cards describe each virtual machine:

- **Details** provides identifying information about the virtual machine, including:
 - Name
 - Namespace
 - Date of creation
 - Node name
 - IP address
- **Inventory** lists the virtual machine's resources, including:
 - Network interface controllers (NICs)
 - Disks
- **Status** includes:
 - The current status of the virtual machine
 - A note indicating whether or not the QEMU guest agent is installed on the virtual machine
- **Utilization** includes charts that display usage data for:
 - CPU
 - Memory
 - Filesystem

- Network transfer



NOTE

Use the drop-down list to choose a duration for the utilization data. The available options are **1 Hour**, **6 Hours**, and **24 Hours**.

- **Events** lists messages about virtual machine activity over the past hour. To view additional events, click **View all**.

13.5. MONITORING VIRTUAL MACHINE HEALTH

A virtual machine instance (VMI) can become unhealthy due to transient issues such as connectivity loss, deadlocks, or problems with external dependencies. A health check periodically performs diagnostics on a VMI by using any combination of the readiness and liveness probes.

13.5.1. About readiness and liveness probes

Use readiness and liveness probes to detect and handle unhealthy virtual machine instances (VMIs). You can include one or more probes in the specification of the VMI to ensure that traffic does not reach a VMI that is not ready for it and that a new instance is created when a VMI becomes unresponsive.

A *readiness probe* determines whether a VMI is ready to accept service requests. If the probe fails, the VMI is removed from the list of available endpoints until the VMI is ready.

A *liveness probe* determines whether a VMI is responsive. If the probe fails, the VMI is deleted and a new instance is created to restore responsiveness.

You can configure readiness and liveness probes by setting the **spec.readinessProbe** and the **spec.livenessProbe** fields of the **VirtualMachineInstance** object. These fields support the following tests:

HTTP GET

The probe determines the health of the VMI 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 VMI. The VMI 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.

13.5.2. Defining an HTTP readiness probe

Define an HTTP readiness probe by setting the **spec.readinessProbe.httpGet** field of the virtual machine instance (VMI) configuration.

Procedure

1. Include details of the readiness probe in the VMI configuration file.

Sample readiness probe with an HTTP GET test

```
# ...
```

```
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 VMI.
- 2 The port of the VMI 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 VMI is considered to be healthy. If the handler returns a failure code, the VMI is removed from the list of available endpoints.
- 4 The time, in seconds, after the VMI 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 VMI 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 VMI by running the following command:

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

13.5.3. Defining a TCP readiness probe

Define a TCP readiness probe by setting the **spec.readinessProbe.tcpSocket** field of the virtual machine instance (VMI) configuration.

Procedure

1. Include details of the TCP readiness probe in the VMI configuration file.

Sample readiness probe with a TCP socket test

```

...
spec:
  readinessProbe:
    initialDelaySeconds: 120 1
    periodSeconds: 20 2
    tcpSocket: 3
      port: 1500 4
    timeoutSeconds: 10 5
...

```

- 1 The time, in seconds, after the VMI 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 VMI that the probe queries.
- 5 The number of seconds of inactivity after which the probe times out and the VMI is assumed to have failed. The default value is 1. This value must be lower than **periodSeconds**.

2. Create the VMI by running the following command:

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

13.5.4. Defining an HTTP liveness probe

Define an HTTP liveness probe by setting the **spec.livenessProbe.httpGet** field of the virtual machine instance (VMI) 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.

Procedure

1. Include details of the HTTP liveness probe in the VMI configuration file.

Sample liveness probe with an HTTP GET test

```

# ...
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 VMI 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 VMI.
- 4 The port of the VMI that the probe queries. In the above example, the probe queries port 1500. The VMI 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 VMI is considered to be healthy. If the handler returns a failure code, the VMI is deleted and a new instance is created.
- 6 The number of seconds of inactivity after which the probe times out and the VMI is assumed to have failed. The default value is 1. This value must be lower than **periodSeconds**.

2. Create the VMI by running the following command:

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

13.5.5. Template: Virtual machine instance configuration file for defining health checks

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
metadata:
  labels:
    special: vmi-fedora
  name: vmi-fedora
spec:
  domain:
    devices:
      disks:
        - disk:
            bus: virtio
            name: containerdisk
        - disk:
            bus: virtio
            name: cloudinitdisk
  resources:
    requests:
      memory: 1024M
  readinessProbe:
    httpGet:
      port: 1500
    initialDelaySeconds: 120
    periodSeconds: 20
    timeoutSeconds: 10
    failureThreshold: 3
    successThreshold: 3
  terminationGracePeriodSeconds: 0
  volumes:
```

```

- name: containerdisk
  containerDisk:
    image: kubevirt/fedora-cloud-registry-disk-demo
- cloudInitNoCloud:
  userData: |-
    #cloud-config
    password: fedora
    chpasswd: { expire: False }
    bootcmd:
      - setenforce 0
      - dnf install -y nmap-ncat
      - systemd-run --unit=httpserver nc -klp 1500 -e '/usr/bin/echo -e HTTP/1.1 200 OK\n\nHello
World!'
    name: cloudinitdisk

```

13.5.6. Additional resources

- [Monitoring application health by using health checks](#)

13.6. USING THE OPENSIFT CONTAINER PLATFORM DASHBOARD TO GET CLUSTER INFORMATION

Access the OpenShift Container Platform dashboard, which captures high-level information about the cluster, by clicking **Home > Dashboards > Overview** from the OpenShift Container Platform web console.

The OpenShift Container Platform dashboard provides various cluster information, captured in individual dashboard *cards*.

13.6.1. About the OpenShift Container Platform dashboards page

The OpenShift Container Platform dashboard consists of the following cards:

- **Details** provides a brief overview of informational cluster details. Status include **ok**, **error**, **warning**, **in progress**, and **unknown**. Resources can add custom status names.
 - Cluster ID
 - Provider
 - Version
- **Cluster Inventory** details number of resources and associated statuses. It is helpful when intervention is required to resolve problems, including information about:
 - Number of nodes
 - Number of pods
 - Persistent storage volume claims
 - Virtual machines (available if OpenShift Virtualization is installed)

- Bare metal hosts in the cluster, listed according to their state (only available in **metal3** environment).
- **Cluster Health** summarizes the current health of the cluster as a whole, including relevant alerts and descriptions. If OpenShift Virtualization is installed, the overall health of OpenShift Virtualization is diagnosed as well. If more than one subsystem is present, click **See All** to view the status of each subsystem.
- **Cluster Capacity** charts help administrators understand when additional resources are required in the cluster. The charts contain an inner ring that displays current consumption, while an outer ring displays thresholds configured for the resource, including information about:
 - CPU time
 - Memory allocation
 - Storage consumed
 - Network resources consumed
- **Cluster Utilization** shows the capacity of various resources over a specified period of time, to help administrators understand the scale and frequency of high resource consumption.
- **Events** lists messages related to recent activity in the cluster, such as pod creation or virtual machine migration to another host.
- **Top Consumers** helps administrators understand how cluster resources are consumed. Click on a resource to jump to a detailed page listing pods and nodes that consume the largest amount of the specified cluster resource (CPU, memory, or storage).

13.7. OPENSIFT CONTAINER PLATFORM CLUSTER MONITORING, LOGGING, AND TELEMETRY

OpenShift Container Platform provides various resources for monitoring at the cluster level.

13.7.1. About OpenShift Container Platform monitoring

OpenShift Container Platform includes a pre-configured, pre-installed, and self-updating monitoring stack that provides **monitoring for core platform components**. OpenShift Container Platform delivers monitoring best practices out of the box. A set of alerts are included by default that immediately notify cluster administrators about issues with a cluster. Default dashboards in the OpenShift Container Platform web console include visual representations of cluster metrics to help you to quickly understand the state of your cluster.

After installing OpenShift Container Platform 4.8, cluster administrators can optionally enable **monitoring for user-defined projects**. By using this feature, cluster administrators, developers, and other users can specify how services and pods are monitored in their own projects. You can then query metrics, review dashboards, and manage alerting rules and silences for your own projects in the OpenShift Container Platform web console.



NOTE

Cluster administrators can grant developers and other users permission to monitor their own projects. Privileges are granted by assigning one of the predefined monitoring roles.

13.7.2. About OpenShift Logging components

The OpenShift Logging components include a collector deployed to each node in the OpenShift Container Platform cluster that collects all node and container logs and writes them to a log store. You can use a centralized web UI to create rich visualizations and dashboards with the aggregated data.

The major components of OpenShift Logging are:

- collection - This is the component that collects logs from the cluster, formats them, and forwards them to the log store. The current implementation is Fluentd.
- log store - This is where the logs are stored. The default implementation is Elasticsearch. You can use the default Elasticsearch log store or forward logs to external log stores. The default log store is optimized and tested for short-term storage.
- visualization - This is the UI component you can use to view logs, graphs, charts, and so forth. The current implementation is Kibana.

For more information on OpenShift Logging, see the [OpenShift Logging](#) documentation.

13.7.3. About Telemetry

Telemetry sends a carefully chosen subset of the cluster monitoring metrics to Red Hat. The Telemeter Client fetches the metrics values every four minutes and thirty seconds and uploads the data to Red Hat. These metrics are described in this document.

This stream of data is used by Red Hat to monitor the clusters in real-time and to react as necessary to problems that impact our customers. It also allows Red Hat to roll out OpenShift Container Platform upgrades to customers to minimize service impact and continuously improve the upgrade experience.

This debugging information is available to Red Hat Support and Engineering teams with the same restrictions as accessing data reported through support cases. All connected cluster information is used by Red Hat to help make OpenShift Container Platform better and more intuitive to use.

13.7.3.1. Information collected by Telemetry

The following information is collected by Telemetry:

- The unique random identifier that is generated during an installation
- Version information, including the OpenShift Container Platform cluster version and installed update details that are used to determine update version availability
- Update information, including the number of updates available per cluster, the channel and image repository used for an update, update progress information, and the number of errors that occur in an update
- The name of the provider platform that OpenShift Container Platform is deployed on and the data center location
- Sizing information about clusters, machine types, and machines, including the number of CPU cores and the amount of RAM used for each
- The number of running virtual machine instances in a cluster
- The number of etcd members and the number of objects stored in the etcd cluster

- The OpenShift Container Platform framework components installed in a cluster and their condition and status
- Usage information about components, features, and extensions
- Usage details about Technology Previews and unsupported configurations
- Information about degraded software
- Information about nodes that are marked as **NotReady**
- Events for all namespaces listed as "related objects" for a degraded Operator
- Configuration details that help Red Hat Support to provide beneficial support for customers, including node configuration at the cloud infrastructure level, hostnames, IP addresses, Kubernetes pod names, namespaces, and services
- Information about the validity of certificates
- Number of application builds by build strategy type

Telemetry does not collect identifying information such as user names or passwords. Red Hat does not intend to collect personal information. If Red Hat discovers that personal information has been inadvertently received, Red Hat will delete such information. To the extent that any telemetry data constitutes personal data, please refer to the [Red Hat Privacy Statement](#) for more information about Red Hat's privacy practices.

13.7.4. CLI troubleshooting and debugging commands

For a list of the **oc** client troubleshooting and debugging commands, see the [OpenShift Container Platform CLI tools](#) documentation.

13.8. PROMETHEUS QUERIES FOR VIRTUAL RESOURCES

OpenShift Virtualization provides metrics for monitoring how infrastructure resources are consumed in the cluster. The metrics cover the following resources:

- vCPU
- Network
- Storage
- Guest memory swapping

Use the OpenShift Container Platform monitoring dashboard to query virtualization metrics.

13.8.1. 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. See the [OpenShift Container Platform machine configuration tasks](#) documentation for more information on applying a kernel argument.

- For guest memory swapping queries to return data, memory swapping must be enabled on the virtual guests.

13.8.2. Querying metrics

The OpenShift Container Platform monitoring dashboard enables you 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**, you can query metrics for all core OpenShift Container Platform and user-defined projects.

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.

13.8.2.1. Querying metrics for all projects as a cluster administrator

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.





NOTE

Only cluster administrators have access to the third-party UIs provided with OpenShift Container Platform Monitoring.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role or with view permissions for all projects.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. In the **Administrator** perspective within the OpenShift Container Platform web console, select **Monitoring** → **Metrics**.
2. Select **Insert Metric at Cursor** to view a list of predefined queries.
3. To create a custom query, add your Prometheus Query Language (PromQL) query to the **Expression** field.
4. To add multiple queries, select **Add Query**.
5. To delete a query, select  next to the query, then choose **Delete query**.
6. To disable a query from being run, select  next to the query and choose **Disable query**.
7. Select **Run Queries** to run the queries that you have created. The metrics from the queries are visualized on the plot. If a query is invalid, the UI shows an error message.

**NOTE**

Queries that operate on large amounts of data might time out or overload the browser when drawing time series graphs. To avoid this, select **Hide graph** and calibrate your query using only the metrics table. Then, after finding a feasible query, enable the plot to draw the graphs.

- Optional: The page URL now contains the queries you ran. To use this set of queries again in the future, save this URL.

Additional resources

- See the [Prometheus query documentation](#) for more information about creating PromQL queries.

13.8.2.2. Querying metrics for user-defined projects as a developer

You can access metrics for a user-defined project as a developer or as a user with view permissions for the project.

In the **Developer** perspective, the Metrics UI includes some predefined CPU, memory, bandwidth, and network packet queries for the selected project. You can also run custom Prometheus Query Language (PromQL) queries for CPU, memory, bandwidth, network packet and application metrics for the project.

**NOTE**

Developers can only use the **Developer** perspective and not the **Administrator** perspective. As a developer, you can only query metrics for one project at a time. Developers cannot access the third-party UIs provided with OpenShift Container Platform monitoring that are for core platform components. Instead, use the Metrics UI for your user-defined 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

- From the **Developer** perspective in the OpenShift Container Platform web console, select **Monitoring** → **Metrics**.
- Select the project that you want to view metrics for in the **Project:** list.
- Choose a query from the **Select Query** list, or run a custom PromQL query by selecting **Show PromQL**.

**NOTE**

In the **Developer** perspective, you can only run one query at a time.

Additional resources

- See the [Prometheus query documentation](#) for more information about creating PromQL queries.

13.8.3. Virtualization metrics

The following metric descriptions include example Prometheus Query Language (PromQL) queries. These metrics are not an API and might change between versions.

**NOTE**

The following examples use **topk** queries that specify a time period. If virtual machines are deleted during that time period, they can still appear in the query output.

13.8.3.1. vCPU metrics

The following query can identify virtual machines that are waiting for Input/Output (I/O):

kubevirt_vmi_vcpu_wait_seconds

Returns the wait time (in seconds) for a virtual machine's vCPU.

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.

Example vCPU wait time query

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_vcpu_wait_seconds[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.

13.8.3.2. 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.

kubevirt_vmi_network_transmit_bytes_total

Returns the total amount of traffic transmitted (in bytes) on the virtual machine's network.

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.

13.8.3.3. Storage metrics

13.8.3.3.1. 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.

kubevirt_vmi_storage_write_traffic_bytes_total

Returns the total amount of storage writes (in bytes) of the virtual machine's storage-related traffic.

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.

13.8.3.3.2. 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.

kubevirt_vmi_storage_iops_write_total

Returns the amount of read I/O operations the virtual machine is performing per second.

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.

13.8.3.4. 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_total

Returns the total amount (in bytes) of memory the virtual guest is swapping in.

kubevirt_vmi_memory_swap_out_traffic_bytes_total

Returns the total amount (in bytes) of memory the virtual guest is swapping out.

Example memory swapping query

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_memory_swap_in_traffic_bytes_total[6m])) +
sum by (name, namespace) (rate(kubevirt_vmi_memory_swap_out_traffic_bytes_total[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.

13.8.4. Additional resources

- [Monitoring overview](#)

13.9. 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.

13.9.1. 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 them 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 by using the default **must-gather** image.
2. Collect **must-gather** data for Red Hat OpenShift Container Storage, if necessary.
3. Collect **must-gather** data for OpenShift Virtualization by using the OpenShift Virtualization **must-gather** image.
4. Collect Prometheus metrics for the cluster.

13.9.1.1. Additional resources

- Configuring the [retention time](#) for Prometheus metrics data
- Configuring the Alertmanager to send [alert notifications](#) to external systems
- Collecting **must-gather** data for [OpenShift Container Platform](#)
- Collecting **must-gather** data for [OpenShift Virtualization](#)
- Collecting Prometheus metrics for [all projects](#) as a cluster administrator

13.9.2. Collecting data about virtual machines

Collecting data about malfunctioning virtual machines (VMs) minimizes the time required to analyze and determine the root cause.

Prerequisites

- Windows VMs:
 - Record the Windows patch update details for Red Hat Support.
 - Install the latest version of the VirtIO drivers. The VirtIO drivers include the QEMU guest agent.
 - If Remote Desktop Protocol (RDP) is enabled, try to connect to the VMs with RDP to determine whether there is a problem with the connection software.

Procedure

1. Collect detailed **must-gather** data about the malfunctioning VMs.
2. Collect screenshots of VMs that have crashed before you restart them.
3. Record factors that the malfunctioning VMs have in common. For example, the VMs have the same host or network.

13.9.2.1. Additional resources

- Installing [VirtIO drivers](#) on Windows VMs
- Downloading and installing [VirtIO drivers](#) on Windows VMs without host access
- Connecting to Windows VMs with RDP using the [web console](#) or the [command line](#)

- Collecting **must-gather** data about [virtual machines](#)

13.9.3. 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

Procedure

- Run the following command to collect data about OpenShift Virtualization:

```
$ oc adm must-gather --image-stream=openshift/must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-
  rhel8:v{HCOVersion}
```

13.9.3.1. must-gather tool options

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 and image stream information
- Limiting the maximum number of parallel processes used by the **must-gather** tool

13.9.3.1.1. 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 only compatible with certain environment variable combinations.

gather_vms_details

Collect VM log files, VM definitions, and namespaces (and their child objects) that belong to OpenShift Virtualization resources. 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.

gather

Use the default **must-gather** script, which collects cluster data from all namespaces and includes only basic VM information. This script is only compatible with the **PROS** variable.

gather_images

Collect image and image stream custom resource information. This script is only compatible with the **PROS** variable.

13.9.3.1.2. Usage and examples

Environment variables are optional. You can run a script by itself or with one or more compatible environment variables.

Table 13.1. Compatible parameters

Script	Compatible environment variable
gather_vms_details	<ul style="list-style-type: none"> • For a namespace: NS=<namespace_name> • For a VM: VM=<vm_name> NS=<namespace_name> • PROS=<number_of_processes>
gather	<ul style="list-style-type: none"> • PROS=<number_of_processes>
gather_images	<ul style="list-style-type: none"> • PROS=<number_of_processes>

To customize the data that **must-gather** collects, you append a double dash (**--**) to the command, followed by a space and one or more compatible parameters.

Syntax

```
$ oc adm must-gather \
--image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.8.7 \
-- <environment_variable_1> <environment_variable_2> <script_name>
```

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-rhel8:v4.8.7 \
-- NS=mynamespace VM=my-vm gather_vms_details 1
```

1 The **NS** environment variable is mandatory if you use the **VM** environment variable.

Default data collection limited to three parallel processes

The following command collects default **must-gather** information by using a maximum of three parallel processes:

```
$ oc adm must-gather \
--image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.8.7 \
-- PROS=3 gather
```

Image and image stream information

The following command collects image and image stream information from the cluster:

```
$ oc adm must-gather \
--image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.8.7 \
-- gather_images
```

13.9.3.2. Additional resources

- [About the **must-gather** tool](#)