



OpenShift Container Platform 4.11

Virtualization

OpenShift Virtualization installation, usage, and release notes

OpenShift Container Platform 4.11 Virtualization

OpenShift Virtualization installation, usage, and release notes

Legal Notice

Copyright © 2022 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	16
1.1. WHAT YOU CAN DO WITH OPENSIFT VIRTUALIZATION	16
1.1.1. OpenShift Virtualization supported cluster version	16
CHAPTER 2. OPENSIFT VIRTUALIZATION ARCHITECTURE	17
2.1. HOW OPENSIFT VIRTUALIZATION ARCHITECTURE WORKS	17
2.2. ABOUT THE HCO-OPERATOR	18
2.3. ABOUT THE CDI-OPERATOR	19
2.4. ABOUT THE CLUSTER-NETWORK-ADDONS-OPERATOR	20
2.5. ABOUT THE HOSTPATH-PROVISIONER-OPERATOR	21
2.6. ABOUT THE SSP-OPERATOR	22
2.7. ABOUT THE TEKTON-TASKS-OPERATOR	22
2.8. ABOUT THE VIRT-OPERATOR	22
CHAPTER 3. GETTING STARTED WITH OPENSIFT VIRTUALIZATION	24
3.1. BEFORE YOU BEGIN	24
3.1.1. Additional resources	24
3.2. GETTING STARTED	24
3.3. NEXT STEPS	25
3.3.1. Additional resources	25
CHAPTER 4. OPENSIFT VIRTUALIZATION RELEASE NOTES	26
4.1. MAKING OPEN SOURCE MORE INCLUSIVE	26
4.2. ABOUT RED HAT OPENSIFT VIRTUALIZATION	26
4.2.1. OpenShift Virtualization supported cluster version	26
4.2.2. Supported guest operating systems	26
4.3. NEW AND CHANGED FEATURES	26
4.3.1. Quick starts	27
4.3.2. Storage	27
4.3.3. Web console	27
4.4. DEPRECATED AND REMOVED FEATURES	27
4.4.1. Deprecated features	27
4.4.2. Removed features	28
4.5. TECHNOLOGY PREVIEW FEATURES	28
4.6. BUG FIXES	29
4.7. KNOWN ISSUES	29
CHAPTER 5. INSTALLING	33
5.1. PREPARING YOUR CLUSTER FOR OPENSIFT VIRTUALIZATION	33
5.1.1. Hardware and operating system requirements	33
5.1.2. Physical resource overhead requirements	34
5.1.2.1. Memory overhead	35
5.1.2.2. CPU overhead	35
5.1.2.3. Storage overhead	36
5.1.2.4. Example	36
5.1.3. Object maximums	36
5.1.4. Restricted network environments	36
5.1.5. Live migration	36
5.1.6. Snapshots and cloning	37
5.1.7. Cluster high-availability options	37
5.2. SPECIFYING NODES FOR OPENSIFT VIRTUALIZATION COMPONENTS	37
5.2.1. About node placement for virtualization components	38

5.2.1.1. How to apply node placement rules to virtualization components	38
5.2.1.2. Node placement in the OLM Subscription object	39
5.2.1.3. Node placement in the HyperConverged object	39
5.2.1.4. Node placement in the HostPathProvisioner object	39
5.2.1.5. Additional resources	40
5.2.2. Example manifests	40
5.2.2.1. Operator Lifecycle Manager Subscription object	40
5.2.2.1.1. Example: Node placement with nodeSelector in the OLM Subscription object	40
5.2.2.1.2. Example: Node placement with tolerations in the OLM Subscription object	41
5.2.2.2. HyperConverged object	41
5.2.2.2.1. Example: Node placement with nodeSelector in the HyperConverged Cluster CR	41
5.2.2.2.2. Example: Node placement with affinity in the HyperConverged Cluster CR	42
5.2.2.2.3. Example: Node placement with tolerations in the HyperConverged Cluster CR	42
5.2.2.3. HostPathProvisioner object	43
5.2.2.3.1. Example: Node placement with nodeSelector in the HostPathProvisioner object	43
5.3. INSTALLING OPENSIFT VIRTUALIZATION USING THE WEB CONSOLE	43
5.3.1. Installing the OpenShift Virtualization Operator	43
5.3.2. Next steps	45
5.4. INSTALLING OPENSIFT VIRTUALIZATION USING THE CLI	45
5.4.1. Prerequisites	45
5.4.2. Subscribing to the OpenShift Virtualization catalog by using the CLI	45
5.4.3. Deploying the OpenShift Virtualization Operator by using the CLI	46
5.4.4. Next steps	47
5.5. ENABLING THE VIRTCTL CLIENT	47
5.5.1. Downloading and installing the virtctl client	47
5.5.1.1. Downloading the virtctl client	47
5.5.1.2. Installing the virtctl client	47
5.5.2. Installing the virtctl RPM package	48
5.5.2.1. Enabling OpenShift Virtualization repositories	48
5.5.2.2. Installing the virtctl client using the yum utility	49
5.5.3. Additional resources	49
5.6. UNINSTALLING OPENSIFT VIRTUALIZATION USING THE WEB CONSOLE	49
5.6.1. Prerequisites	49
5.6.2. Deleting the OpenShift Virtualization Operator Deployment custom resource	49
5.6.3. Deleting the OpenShift Virtualization catalog subscription	50
5.6.4. Deleting a namespace using the web console	50
5.7. UNINSTALLING OPENSIFT VIRTUALIZATION USING THE CLI	51
5.7.1. Prerequisites	51
5.7.2. Deleting OpenShift Virtualization	51
CHAPTER 6. UPDATING OPENSIFT VIRTUALIZATION	53
6.1. ABOUT UPDATING OPENSIFT VIRTUALIZATION	53
6.2. CONFIGURING AUTOMATIC WORKLOAD UPDATES	54
6.2.1. About workload updates	54
Migration attempts and timeouts	54
6.2.2. Configuring workload update methods	55
6.3. APPROVING PENDING OPERATOR UPDATES	56
6.3.1. Manually approving a pending Operator upgrade	56
6.4. MONITORING UPDATE STATUS	56
6.4.1. Monitoring OpenShift Virtualization upgrade status	57
6.4.2. Viewing outdated OpenShift Virtualization workloads	57
6.5. ADDITIONAL RESOURCES	58

CHAPTER 7. SECURITY POLICIES	59
7.1. ABOUT WORKLOAD SECURITY	59
7.2. EXTENDED SELINUX POLICIES FOR VIRT-LAUNCHER PODS	59
7.3. ADDITIONAL OPENSIFT CONTAINER PLATFORM SECURITY CONTEXT CONSTRAINTS AND LINUX CAPABILITIES FOR THE KUBEVIRT-CONTROLLER SERVICE ACCOUNT	59
7.3.1. Additional SCCs granted to the kubevirt-controller service account	60
7.3.2. Viewing the SCC and RBAC definitions for the kubevirt-controller	60
7.4. ADDITIONAL RESOURCES	60
CHAPTER 8. USING THE CLI TOOLS	61
8.1. PREREQUISITES	61
8.2. OPENSIFT CONTAINER PLATFORM CLIENT COMMANDS	61
8.3. VIRTCTL CLIENT COMMANDS	61
8.4. CREATING A CONTAINER USING VIRTCTL GUESTFS	63
8.5. LIBGUESTFS TOOLS AND VIRTCTL GUESTFS	63
8.6. ADDITIONAL RESOURCES	65
CHAPTER 9. VIRTUAL MACHINES	66
9.1. CREATING VIRTUAL MACHINES	66
9.1.1. Using a Quick Start to create a virtual machine	66
9.1.2. Quick creating a virtual machine	67
9.1.3. Creating a virtual machine from a customized template	67
9.1.3.1. Virtual machine fields	68
9.1.3.1.1. Networking fields	69
9.1.3.1.2. Storage fields	69
Advanced storage settings	70
9.1.3.1.3. Cloud-init fields	71
9.1.3.2. Pasting in a pre-configured YAML file to create a virtual machine	71
9.1.4. Using the CLI to create a virtual machine	72
9.1.5. Virtual machine storage volume types	74
9.1.6. About RunStrategies for virtual machines	75
9.1.7. Additional resources	76
9.2. EDITING VIRTUAL MACHINES	77
9.2.1. Editing a virtual machine in the web console	77
9.2.1.1. Virtual machine fields	78
9.2.2. Editing a virtual machine YAML configuration using the web console	79
9.2.3. Editing a virtual machine YAML configuration using the CLI	79
9.2.4. Adding a virtual disk to a virtual machine	80
9.2.4.1. Editing CD-ROMs for VirtualMachines	80
9.2.4.2. Storage fields	81
Advanced storage settings	82
9.2.5. Adding a network interface to a virtual machine	83
9.2.5.1. Networking fields	83
9.2.6. Additional resources	84
9.3. EDITING BOOT ORDER	84
9.3.1. Adding items to a boot order list in the web console	84
9.3.2. Editing a boot order list in the web console	85
9.3.3. Editing a boot order list in the YAML configuration file	85
9.3.4. Removing items from a boot order list in the web console	86
9.4. DELETING VIRTUAL MACHINES	87
9.4.1. Deleting a virtual machine using the web console	87
9.4.2. Deleting a virtual machine by using the CLI	87
9.5. MANAGING VIRTUAL MACHINE INSTANCES	88

9.5.1. About virtual machine instances	88
9.5.2. Listing all virtual machine instances using the CLI	88
9.5.3. Listing standalone virtual machine instances using the web console	89
9.5.4. Editing a standalone virtual machine instance using the web console	89
9.5.5. Deleting a standalone virtual machine instance using the CLI	89
9.5.6. Deleting a standalone virtual machine instance using the web console	89
9.6. CONTROLLING VIRTUAL MACHINE STATES	90
9.6.1. Starting a virtual machine	90
9.6.2. Restarting a virtual machine	90
9.6.3. Stopping a virtual machine	91
9.6.4. Unpausing a virtual machine	91
9.7. ACCESSING VIRTUAL MACHINE CONSOLES	92
9.7.1. Accessing virtual machine consoles in the OpenShift Container Platform web console	92
9.7.1.1. Connecting to the serial console	92
9.7.1.2. Connecting to the VNC console	93
9.7.1.3. Connecting to a Windows virtual machine with RDP	93
9.7.1.4. Switching between virtual machine displays	94
9.7.2. Accessing virtual machine consoles by using CLI commands	94
9.7.2.1. Accessing a virtual machine instance via SSH	94
9.7.2.2. Accessing a virtual machine via SSH with YAML configurations	95
9.7.2.3. Accessing the serial console of a virtual machine instance	98
9.7.2.4. Accessing the graphical console of a virtual machine instances with VNC	98
9.7.2.5. Connecting to a Windows virtual machine with an RDP console	99
9.8. AUTOMATING WINDOWS INSTALLATION WITH SYSPREP	100
9.8.1. Using a Windows DVD to create a VM disk image	100
9.8.2. Using a disk image to install Windows	100
9.8.3. Generalizing a Windows VM using sysprep	101
9.8.4. Specializing a Windows virtual machine	102
9.8.5. Additional resources	102
9.9. TRIGGERING VIRTUAL MACHINE FAILOVER BY RESOLVING A FAILED NODE	102
9.9.1. Prerequisites	103
9.9.2. Deleting nodes from a bare metal cluster	103
9.9.3. Verifying virtual machine failover	103
9.9.3.1. Listing all virtual machine instances using the CLI	104
9.10. INSTALLING THE QEMU GUEST AGENT ON VIRTUAL MACHINES	104
9.10.1. Installing QEMU guest agent on a Linux virtual machine	104
9.10.2. Installing QEMU guest agent on a Windows virtual machine	105
9.10.2.1. Installing VirtIO drivers on an existing Windows virtual machine	105
9.10.2.2. Installing VirtIO drivers during Windows installation	106
9.11. VIEWING THE QEMU GUEST AGENT INFORMATION FOR VIRTUAL MACHINES	106
9.11.1. Prerequisites	106
9.11.2. About the QEMU guest agent information in the web console	106
9.11.3. Viewing the QEMU guest agent information in the web console	107
9.12. MANAGING CONFIG MAPS, SECRETS, AND SERVICE ACCOUNTS IN VIRTUAL MACHINES	107
9.12.1. Adding a secret, config map, or service account to a virtual machine	107
9.12.2. Removing a secret, config map, or service account from a virtual machine	108
9.12.3. Additional resources	109
9.13. INSTALLING VIRTIO DRIVER ON AN EXISTING WINDOWS VIRTUAL MACHINE	109
9.13.1. About VirtIO drivers	109
9.13.2. Supported VirtIO drivers for Microsoft Windows virtual machines	109
9.13.3. Adding VirtIO drivers container disk to a virtual machine	109
9.13.4. Installing VirtIO drivers on an existing Windows virtual machine	110
9.13.5. Removing the VirtIO container disk from a virtual machine	111

9.14. INSTALLING VIRTIO DRIVER ON A NEW WINDOWS VIRTUAL MACHINE	112
9.14.1. Prerequisites	112
9.14.2. About VirtIO drivers	112
9.14.3. Supported VirtIO drivers for Microsoft Windows virtual machines	112
9.14.4. Adding VirtIO drivers container disk to a virtual machine	112
9.14.5. Installing VirtIO drivers during Windows installation	113
9.14.6. Removing the VirtIO container disk from a virtual machine	114
9.15. USING VIRTUAL TRUSTED PLATFORM MODULE DEVICES	114
9.15.1. About vTPM devices	114
9.15.2. Adding a vTPM device to a virtual machine	115
9.16. ADVANCED VIRTUAL MACHINE MANAGEMENT	115
9.16.1. Working with resource quotas for virtual machines	115
9.16.1.1. Setting resource quota limits for virtual machines	115
9.16.1.2. Additional resources	116
9.16.2. Specifying nodes for virtual machines	116
9.16.2.1. About node placement for virtual machines	116
9.16.2.2. Node placement examples	117
9.16.2.2.1. Example: VM node placement with nodeSelector	117
9.16.2.2.2. Example: VM node placement with pod affinity and pod anti-affinity	118
9.16.2.2.3. Example: VM node placement with node affinity	118
9.16.2.2.4. Example: VM node placement with tolerations	119
9.16.2.3. Additional resources	120
9.16.3. Configuring certificate rotation	120
9.16.3.1. Configuring certificate rotation	120
9.16.3.2. Troubleshooting certificate rotation parameters	121
9.16.4. Automating management tasks	121
9.16.4.1. About Red Hat Ansible Automation	121
9.16.4.2. Automating virtual machine creation	121
9.16.4.3. Example: Ansible Playbook for creating virtual machines	123
9.16.5. Using UEFI mode for virtual machines	124
9.16.5.1. About UEFI mode for virtual machines	124
9.16.5.2. Booting virtual machines in UEFI mode	124
9.16.6. Configuring PXE booting for virtual machines	125
9.16.6.1. Prerequisites	125
9.16.6.2. PXE booting with a specified MAC address	125
9.16.6.3. Template: Virtual machine configuration file for PXE booting	128
9.16.6.4. OpenShift Virtualization networking glossary	129
9.16.7. Using huge pages with virtual machines	129
9.16.7.1. Prerequisites	129
9.16.7.2. What huge pages do	129
9.16.7.3. Configuring huge pages for virtual machines	130
9.16.8. Enabling dedicated resources for virtual machines	131
9.16.8.1. About dedicated resources	131
9.16.8.2. Prerequisites	131
9.16.8.3. Enabling dedicated resources for a virtual machine	131
9.16.9. Scheduling virtual machines	131
9.16.9.1. Policy attributes	131
9.16.9.2. Setting a policy attribute and CPU feature	132
9.16.9.3. Scheduling virtual machines with the supported CPU model	133
9.16.9.4. Scheduling virtual machines with the host model	133
9.16.10. Configuring PCI passthrough	133
9.16.10.1. About preparing a host device for PCI passthrough	134
9.16.10.1.1. Adding kernel arguments to enable the IOMMU driver	134

9.16.10.1.2. Binding PCI devices to the VFIO driver	135
9.16.10.1.3. Exposing PCI host devices in the cluster using the CLI	137
9.16.10.1.4. Removing PCI host devices from the cluster using the CLI	138
9.16.10.2. Configuring virtual machines for PCI passthrough	140
9.16.10.2.1. Assigning a PCI device to a virtual machine	140
9.16.10.3. Additional resources	140
9.16.11. Configuring vGPU passthrough	141
9.16.11.1. Assigning vGPU passthrough devices to a virtual machine	141
9.16.11.2. Additional resources	142
9.16.12. Configuring mediated devices	142
9.16.12.1. Prerequisites	142
9.16.12.2. About using virtual GPUs with OpenShift Virtualization	142
9.16.12.2.1. Configuration overview	142
9.16.12.2.2. How vGPUs are assigned to nodes	143
9.16.12.2.3. About changing and removing mediated devices	144
9.16.12.3. Preparing hosts for mediated devices	145
9.16.12.3.1. Adding kernel arguments to enable the IOMMU driver	145
9.16.12.4. Adding and removing mediated devices	146
9.16.12.4.1. Creating and exposing mediated devices	146
9.16.12.4.2. Removing mediated devices from the cluster using the CLI	147
9.16.12.5. Assigning a mediated device to a virtual machine	148
9.16.12.6. Additional resources	148
9.16.13. Configuring a watchdog	148
9.16.13.1. Prerequisites	148
9.16.13.2. Defining a watchdog device	149
9.16.13.3. Installing a watchdog device	150
9.16.13.4. Additional resources	150
9.16.14. Automatic importing and updating of pre-defined boot sources	150
9.16.14.1. Enabling automatic boot source updates	151
9.16.14.2. Disabling automatic boot source updates	151
9.16.14.3. Re-enabling automatic boot source updates	151
9.16.14.4. Configuring a storage class for user-defined boot source updates	152
9.16.14.5. Enabling automatic updates for user-defined boot sources	152
9.16.14.6. Disabling an automatic update for a system-defined or user-defined boot source	153
9.16.14.7. Verifying the status of a boot source	154
9.16.15. Enabling descheduler evictions on virtual machines	155
9.16.15.1. Descheduler profiles	156
9.16.15.2. Installing the descheduler	156
9.16.15.3. Enabling descheduler evictions on a virtual machine (VM)	157
9.16.15.4. Additional resources	158
9.17. IMPORTING VIRTUAL MACHINES	158
9.17.1. TLS certificates for data volume imports	158
9.17.1.1. Adding TLS certificates for authenticating data volume imports	158
9.17.1.2. Example: Config map created from a TLS certificate	159
9.17.2. Importing virtual machine images with data volumes	159
9.17.2.1. Prerequisites	159
9.17.2.2. CDI supported operations matrix	159
9.17.2.3. About data volumes	160
9.17.2.4. Importing a virtual machine image into storage by using a data volume	160
9.17.2.5. Additional resources	163
9.17.3. Importing virtual machine images into block storage with data volumes	163
9.17.3.1. Prerequisites	163
9.17.3.2. About data volumes	164

9.17.3.3. About block persistent volumes	164
9.17.3.4. Creating a local block persistent volume	164
9.17.3.5. Importing a virtual machine image into block storage by using a data volume	165
9.17.3.6. CDI supported operations matrix	166
9.17.3.7. Additional resources	167
9.18. CLONING VIRTUAL MACHINES	167
9.18.1. Enabling user permissions to clone data volumes across namespaces	167
9.18.1.1. Prerequisites	167
9.18.1.2. About data volumes	167
9.18.1.3. Creating RBAC resources for cloning data volumes	168
9.18.2. Cloning a virtual machine disk into a new data volume	169
9.18.2.1. Prerequisites	169
9.18.2.2. About data volumes	169
9.18.2.3. Cloning the persistent volume claim of a virtual machine disk into a new data volume	169
9.18.2.4. Template: Data volume clone configuration file	170
9.18.2.5. CDI supported operations matrix	171
9.18.3. Cloning a virtual machine by using a data volume template	171
9.18.3.1. Prerequisites	172
9.18.3.2. About data volumes	172
9.18.3.3. Creating a new virtual machine from a cloned persistent volume claim by using a data volume template	172
9.18.3.4. Template: Data volume virtual machine configuration file	174
9.18.3.5. CDI supported operations matrix	174
9.18.4. Cloning a virtual machine disk into a new block storage data volume	175
9.18.4.1. Prerequisites	175
9.18.4.2. About data volumes	175
9.18.4.3. About block persistent volumes	176
9.18.4.4. Creating a local block persistent volume	176
9.18.4.5. Cloning the persistent volume claim of a virtual machine disk into a new data volume	177
9.18.4.6. CDI supported operations matrix	178
9.19. VIRTUAL MACHINE NETWORKING	179
9.19.1. Configuring the virtual machine for the default pod network	179
9.19.1.1. Configuring masquerade mode from the command line	179
9.19.1.2. Configuring masquerade mode with dual-stack (IPv4 and IPv6)	180
9.19.2. Creating a service to expose a virtual machine	181
9.19.2.1. About services	181
9.19.2.1.1. Dual-stack support	182
9.19.2.2. Exposing a virtual machine as a service	182
9.19.2.3. Additional resources	184
9.19.3. Attaching a virtual machine to a Linux bridge network	185
9.19.3.1. Connecting to the network through the network attachment definition	185
9.19.3.1.1. Creating a Linux bridge node network configuration policy	185
9.19.3.2. Creating a Linux bridge network attachment definition	186
9.19.3.2.1. Creating a Linux bridge network attachment definition in the web console	186
9.19.3.2.2. Creating a Linux bridge network attachment definition in the CLI	187
9.19.3.3. Configuring the virtual machine for a Linux bridge network	188
9.19.3.3.1. Creating a NIC for a virtual machine in the web console	188
9.19.3.3.2. Networking fields	189
9.19.3.3.3. Attaching a virtual machine to an additional network in the CLI	189
9.19.4. Configuring IP addresses for virtual machines	190
9.19.4.1. Configuring an IP address for a new virtual machine using cloud-init	190
9.19.5. Configuring an SR-IOV network device for virtual machines	191
9.19.5.1. Prerequisites	191

9.19.5.2. Automated discovery of SR-IOV network devices	192
9.19.5.2.1. Example SrioNetworkNodeState object	192
9.19.5.3. Configuring SR-IOV network devices	193
9.19.5.4. Next steps	195
9.19.6. Connecting virtual machines to a service mesh	195
9.19.6.1. Prerequisites	195
9.19.6.2. Configuring a virtual machine for the service mesh	196
9.19.7. Defining an SR-IOV network	197
9.19.7.1. Prerequisites	197
9.19.7.2. Configuring SR-IOV additional network	198
9.19.7.3. Next steps	199
9.19.8. Attaching a virtual machine to an SR-IOV network	199
9.19.8.1. Prerequisites	200
9.19.8.2. Attaching a virtual machine to an SR-IOV network	200
9.19.9. Viewing the IP address of NICs on a virtual machine	201
9.19.9.1. Prerequisites	201
9.19.9.2. Viewing the IP address of a virtual machine interface in the CLI	201
9.19.9.3. Viewing the IP address of a virtual machine interface in the web console	202
9.19.10. Using a MAC address pool for virtual machines	202
9.19.10.1. About KubeMacPool	202
9.19.10.2. Disabling a MAC address pool for a namespace in the CLI	202
9.19.10.3. Re-enabling a MAC address pool for a namespace in the CLI	202
9.20. VIRTUAL MACHINE DISKS	203
9.20.1. Storage features	203
9.20.1.1. OpenShift Virtualization storage feature matrix	203
9.20.2. Configuring local storage for virtual machines	204
9.20.2.1. About the hostpath provisioner	204
9.20.2.2. Creating a hostpath provisioner with a basic storage pool	204
9.20.2.3. About creating storage classes	205
9.20.2.3.1. Creating a storage class for the CSI driver with the storagePools stanza	206
9.20.2.3.2. Creating a storage class for the legacy hostpath provisioner	206
9.20.2.4. About storage pools created with PVC templates	207
9.20.2.4.1. Creating a storage pool with a PVC template	208
9.20.3. Creating data volumes	209
9.20.3.1. Creating data volumes using the storage API	209
9.20.3.2. Creating data volumes using the PVC API	210
9.20.3.3. Customizing the storage profile	212
9.20.3.3.1. Setting a default cloning strategy using a storage profile	213
9.20.3.4. Additional resources	214
9.20.4. Reserving PVC space for file system overhead	214
9.20.4.1. How file system overhead affects space for virtual machine disks	214
9.20.4.2. Overriding the default file system overhead value	215
9.20.5. Configuring CDI to work with namespaces that have a compute resource quota	215
9.20.5.1. About CPU and memory quotas in a namespace	216
9.20.5.2. Overriding CPU and memory defaults	216
9.20.5.3. Additional resources	216
9.20.6. Managing data volume annotations	216
9.20.6.1. Example: Data volume annotations	217
9.20.7. Using preallocation for data volumes	217
9.20.7.1. About preallocation	217
9.20.7.2. Enabling preallocation for a data volume	218
9.20.8. Uploading local disk images by using the web console	218
9.20.8.1. Prerequisites	218

9.20.8.2. CDI supported operations matrix	218
9.20.8.3. Uploading an image file using the web console	219
9.20.8.4. Additional resources	220
9.20.9. Uploading local disk images by using the virtctl tool	220
9.20.9.1. Prerequisites	220
9.20.9.2. About data volumes	220
9.20.9.3. Creating an upload data volume	220
9.20.9.4. Uploading a local disk image to a data volume	221
9.20.9.5. CDI supported operations matrix	222
9.20.9.6. Additional resources	223
9.20.10. Uploading a local disk image to a block storage data volume	223
9.20.10.1. Prerequisites	223
9.20.10.2. About data volumes	223
9.20.10.3. About block persistent volumes	223
9.20.10.4. Creating a local block persistent volume	224
9.20.10.5. Creating an upload data volume	225
9.20.10.6. Uploading a local disk image to a data volume	225
9.20.10.7. CDI supported operations matrix	227
9.20.10.8. Additional resources	227
9.20.11. Managing virtual machine snapshots	227
9.20.11.1. About virtual machine snapshots	228
9.20.11.1.1. Virtual machine snapshot controller and custom resource definitions (CRDs)	228
9.20.11.2. Installing QEMU guest agent on a Linux virtual machine	229
9.20.11.3. Installing QEMU guest agent on a Windows virtual machine	229
9.20.11.3.1. Installing VirtIO drivers on an existing Windows virtual machine	230
9.20.11.3.2. Installing VirtIO drivers during Windows installation	230
9.20.11.4. Creating a virtual machine snapshot in the web console	231
9.20.11.5. Creating an virtual machine snapshot in the CLI	232
9.20.11.6. Verifying online snapshot creation with snapshot indications	234
9.20.11.7. Restoring a virtual machine from a snapshot in the web console	235
9.20.11.8. Restoring a virtual machine from a snapshot in the CLI	235
9.20.11.9. Deleting a virtual machine snapshot in the web console	237
9.20.11.10. Deleting a virtual machine snapshot in the CLI	238
9.20.11.11. Additional resources	238
9.20.12. Moving a local virtual machine disk to a different node	238
9.20.12.1. Cloning a local volume to another node	239
9.20.13. Expanding virtual storage by adding blank disk images	241
9.20.13.1. About data volumes	241
9.20.13.2. Creating a blank disk image with data volumes	241
9.20.13.3. Template: Data volume configuration file for blank disk images	242
9.20.13.4. Additional resources	242
9.20.14. Cloning a data volume using smart-cloning	242
9.20.14.1. About smart-cloning	243
9.20.14.2. Cloning a data volume	243
9.20.14.3. Additional resources	244
9.20.15. Creating and using boot sources	244
9.20.15.1. About virtual machines and boot sources	244
9.20.15.2. Importing a RHEL image as a boot source	245
9.20.15.3. Adding a boot source for a virtual machine template	246
9.20.15.4. Creating a virtual machine from a template with an attached boot source	247
9.20.15.5. Additional resources	247
9.20.16. Hot-plugging virtual disks	247
9.20.16.1. Hot-plugging a virtual disk using the CLI	248

9.20.16.2. Hot-unplugging a virtual disk using the CLI	248
9.20.16.3. Hot-plugging a virtual disk using the web console	248
9.20.16.4. Hot-unplugging a virtual disk using the web console	249
9.20.17. Using container disks with virtual machines	249
9.20.17.1. About container disks	250
9.20.17.1.1. Importing a container disk into a PVC by using a data volume	250
9.20.17.1.2. Attaching a container disk to a virtual machine as a containerDisk volume	250
9.20.17.2. Preparing a container disk for virtual machines	250
9.20.17.3. Disabling TLS for a container registry to use as insecure registry	251
9.20.17.4. Next steps	252
9.20.18. Preparing CDI scratch space	252
9.20.18.1. About data volumes	252
9.20.18.2. About scratch space	252
Manual provisioning	252
9.20.18.3. CDI operations that require scratch space	253
9.20.18.4. Defining a storage class	253
9.20.18.5. CDI supported operations matrix	254
9.20.18.6. Additional resources	254
9.20.19. Re-using persistent volumes	254
9.20.19.1. About reclaiming statically provisioned persistent volumes	254
9.20.19.2. Reclaiming statically provisioned persistent volumes	255
9.20.20. Expanding a virtual machine disk	256
9.20.20.1. Enlarging a virtual machine disk	256
9.20.20.2. Additional resources	257
9.20.21. Deleting data volumes	257
9.20.21.1. About data volumes	257
9.20.21.2. Listing all data volumes	257
9.20.21.3. Deleting a data volume	257
CHAPTER 10. VIRTUAL MACHINE TEMPLATES	259
10.1. CREATING VIRTUAL MACHINE TEMPLATES	259
10.1.1. About virtual machine templates	259
10.1.2. About virtual machines and boot sources	259
10.1.3. Creating a virtual machine template in the web console	260
10.1.4. Adding a boot source for a virtual machine template	260
10.1.4.1. Virtual machine template fields for adding a boot source	261
10.1.5. Marking virtual machine templates as favorites	263
10.1.6. Additional resources	263
10.2. EDITING VIRTUAL MACHINE TEMPLATES	263
10.2.1. Editing a virtual machine template in the web console	263
10.2.1.1. Virtual machine template fields	264
10.2.1.2. Adding a network interface to a virtual machine template	265
10.2.1.3. Adding a virtual disk to a virtual machine template	265
10.2.1.4. Editing CD-ROMs for Templates	266
10.3. ENABLING DEDICATED RESOURCES FOR VIRTUAL MACHINE TEMPLATES	266
10.3.1. About dedicated resources	266
10.3.2. Prerequisites	266
10.3.3. Enabling dedicated resources for a virtual machine template	266
10.4. DEPLOYING A VIRTUAL MACHINE TEMPLATE TO A CUSTOM NAMESPACE	267
10.4.1. Creating a custom namespace for templates	267
10.4.2. Adding templates to a custom namespace	267
10.4.2.1. Deleting templates from a custom namespace	268
10.4.2.2. Additional resources	269

10.5. DELETING VIRTUAL MACHINE TEMPLATES	269
10.5.1. Deleting a virtual machine template in the web console	269
CHAPTER 11. LIVE MIGRATION	270
11.1. VIRTUAL MACHINE LIVE MIGRATION	270
11.1.1. About live migration	270
11.1.2. Updating access mode for live migration	270
11.2. LIVE MIGRATION LIMITS AND TIMEOUTS	270
11.2.1. Configuring live migration limits and timeouts	270
11.2.2. Cluster-wide live migration limits and timeouts	271
11.3. MIGRATING A VIRTUAL MACHINE INSTANCE TO ANOTHER NODE	272
11.3.1. Initiating live migration of a virtual machine instance in the web console	272
11.3.2. Initiating live migration of a virtual machine instance in the CLI	272
11.4. MIGRATING A VIRTUAL MACHINE OVER A DEDICATED ADDITIONAL NETWORK	273
11.4.1. Configuring a dedicated secondary network for virtual machine live migration	273
11.4.2. Additional resources	275
11.5. MONITORING LIVE MIGRATION OF A VIRTUAL MACHINE INSTANCE	275
11.5.1. Monitoring live migration of a virtual machine instance in the web console	275
11.5.2. Monitoring live migration of a virtual machine instance in the CLI	275
11.6. CANCELLING THE LIVE MIGRATION OF A VIRTUAL MACHINE INSTANCE	276
11.6.1. Cancelling live migration of a virtual machine instance in the web console	276
11.6.2. Cancelling live migration of a virtual machine instance in the CLI	276
11.7. CONFIGURING VIRTUAL MACHINE EVICTION STRATEGY	276
11.7.1. Configuring custom virtual machines with the LiveMigration eviction strategy	276
11.8. CONFIGURING LIVE MIGRATION POLICIES	277
11.8.1. Configuring a live migration policy	277
CHAPTER 12. NODE MAINTENANCE	279
12.1. ABOUT NODE MAINTENANCE	279
12.1.1. About node maintenance mode	279
12.1.2. Maintaining bare metal nodes	279
12.1.3. Additional resources	280
12.2. AUTOMATIC RENEWAL OF TLS CERTIFICATES	280
12.2.1. TLS certificates automatic renewal schedules	280
12.3. MANAGING NODE LABELING FOR OBSOLETE CPU MODELS	280
12.3.1. About node labeling for obsolete CPU models	280
12.3.2. About node labeling for CPU features	281
12.3.3. Configuring obsolete CPU models	283
12.4. PREVENTING NODE RECONCILIATION	284
12.4.1. Using skip-node annotation	284
12.4.2. Additional resources	284
CHAPTER 13. LOGGING, EVENTS, AND MONITORING	285
13.1. REVIEWING VIRTUALIZATION OVERVIEW	285
13.1.1. Prerequisites	285
13.1.2. Resources monitored actively in the Virtualization Overview page	285
13.1.3. Resources monitored for top consumption	286
13.1.4. Reviewing top consumers for projects, virtual machines, and nodes	287
13.1.5. Additional resources	287
13.2. VIEWING OPENSIFT VIRTUALIZATION LOGS	287
13.2.1. Viewing OpenShift Virtualization logs with the CLI	287
13.2.2. Viewing virtual machine logs in the web console	289
13.2.3. Common error messages	289
13.3. VIEWING EVENTS	289

13.3.1. About virtual machine events	289
13.3.2. Viewing the events for a virtual machine in the web console	290
13.3.3. Viewing namespace events in the CLI	290
13.3.4. Viewing resource events in the CLI	290
13.4. DIAGNOSING DATA VOLUMES USING EVENTS AND CONDITIONS	290
13.4.1. About conditions and events	291
13.4.2. Analyzing data volumes using conditions and events	291
13.5. VIEWING INFORMATION ABOUT VIRTUAL MACHINE WORKLOADS	293
13.5.1. The Virtual Machines dashboard	293
13.6. MONITORING VIRTUAL MACHINE HEALTH	294
13.6.1. About readiness and liveness probes	294
13.6.2. Defining an HTTP readiness probe	295
13.6.3. Defining a TCP readiness probe	296
13.6.4. Defining an HTTP liveness probe	296
13.6.5. Template: Virtual machine configuration file for defining health checks	297
13.6.6. Additional resources	298
13.7. USING THE OPENSIFT CONTAINER PLATFORM DASHBOARD TO GET CLUSTER INFORMATION	298
13.7.1. About the OpenShift Container Platform dashboards page	299
13.8. REVIEWING RESOURCE USAGE BY VIRTUAL MACHINES	300
13.8.1. About reviewing top consumers	300
13.8.2. Reviewing top consumers	300
13.8.3. Additional resources	301
13.9. OPENSIFT CONTAINER PLATFORM CLUSTER MONITORING, LOGGING, AND TELEMETRY	301
13.9.1. About OpenShift Container Platform monitoring	301
13.9.2. About logging subsystem components	301
13.9.3. About Telemetry	302
13.9.3.1. Information collected by Telemetry	302
13.9.4. CLI troubleshooting and debugging commands	303
13.10. RUNNING CLUSTER CHECKUPS	303
13.10.1. About the OpenShift Container Platform cluster checkup framework	303
13.10.2. Checking network connectivity and latency for virtual machines on a secondary network	304
13.10.3. Additional resources	308
13.11. PROMETHEUS QUERIES FOR VIRTUAL RESOURCES	308
13.11.1. Prerequisites	309
13.11.2. About querying metrics	309
13.11.2.1. Querying metrics for all projects as a cluster administrator	309
13.11.2.2. Querying metrics for user-defined projects as a developer	310
13.11.3. Virtualization metrics	311
13.11.3.1. vCPU metrics	311
13.11.3.2. Network metrics	312
13.11.3.3. Storage metrics	312
13.11.3.3.1. Storage-related traffic	312
13.11.3.3.2. Storage snapshot data	312
13.11.3.3.3. I/O performance	313
13.11.3.4. Guest memory swapping metrics	313
13.11.4. Additional resources	314
13.12. EXPOSING CUSTOM METRICS FOR VIRTUAL MACHINES	314
13.12.1. Configuring the node exporter service	314
13.12.2. Configuring a virtual machine with the node exporter service	315
13.12.3. Creating a custom monitoring label for virtual machines	316
13.12.3.1. Querying the node-exporter service for metrics	317
13.12.4. Creating a ServiceMonitor resource for the node exporter service	318
13.12.4.1. Accessing the node exporter service outside the cluster	319

13.12.5. Additional resources	320
13.13. OPENSIFT VIRTUALIZATION CRITICAL ALERTS	320
13.13.1. Network alerts	320
13.13.1.1. KubeMacPoolDown alert	320
13.13.2. SSP alerts	321
13.13.2.1. SSPFailingToReconcile alert	321
13.13.2.2. SSPOperatorDown alert	322
13.13.2.3. SSPTemplateValidatorDown alert	322
13.13.3. Virt alerts	323
13.13.3.1. NoLeadingVirtOperator alert	323
13.13.3.2. NoReadyVirtController alert	324
13.13.3.3. NoReadyVirtOperator alert	325
13.13.3.4. VirtAPIDown alert	326
13.13.3.5. VirtApiRESTErrorsBurst alert	327
13.13.3.6. VirtControllerDown alert	328
13.13.3.7. VirtControllerRESTErrorsBurst alert	329
13.13.3.8. VirtHandlerRESTErrorsBurst alert	330
13.13.3.9. VirtOperatorDown alert	330
13.13.3.10. VirtOperatorRESTErrorsBurst alert	332
13.13.4. Additional resources	332
13.14. COLLECTING DATA FOR RED HAT SUPPORT	332
13.14.1. Collecting data about your environment	333
13.14.1.1. Additional resources	333
13.14.2. Collecting data about virtual machines	333
13.14.2.1. Additional resources	334
13.14.3. Using the must-gather tool for OpenShift Virtualization	334
13.14.3.1. must-gather tool options	334
13.14.3.1.1. Parameters	335
13.14.3.1.2. Usage and examples	335
13.14.3.2. Additional resources	337
CHAPTER 14. BACKUP AND RESTORE	338
14.1. INSTALLING AND CONFIGURING OADP	338
14.1.1. Installing the OADP Operator	338
14.1.2. About backup and snapshot locations and their secrets	338
Backup locations	338
Snapshot locations	338
Secrets	339
14.1.2.1. Creating a default Secret	339
14.1.3. Configuring the Data Protection Application	339
14.1.3.1. Setting Velero CPU and memory resource allocations	340
14.1.3.2. Enabling self-signed CA certificates	340
14.1.4. Installing the Data Protection Application	341
14.1.4.1. Enabling CSI in the DataProtectionApplication CR	343
14.1.5. Uninstalling OADP	344
14.2. BACKING UP AND RESTORING VIRTUAL MACHINES	344
14.2.1. Additional resources	345
14.3. BACKING UP VIRTUAL MACHINES	345
14.3.1. Creating a Backup CR	345
14.3.1.1. Backing up persistent volumes with CSI snapshots	347
14.3.1.2. Backing up applications with Restic	347
14.3.1.3. Creating backup hooks	348
14.3.2. Scheduling backups	349

14.3.3. Additional resources	351
14.4. RESTORING VIRTUAL MACHINES	351
14.4.1. Creating a Restore CR	351
14.4.1.1. Creating restore hooks	352

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 by using Kubernetes custom resources to enable virtualization tasks. These tasks include:

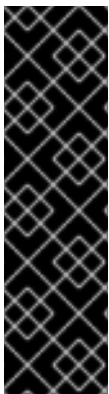
- Creating and managing Linux and Windows virtual machines
- Connecting to virtual machines through a variety of consoles and CLI tools
- Importing and cloning existing virtual machines
- Managing network interface controllers and storage disks attached to virtual machines
- Live migrating virtual machines 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 designed and tested to work well with Red Hat OpenShift Data Foundation features.

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 Plug-ins](#).

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



IMPORTANT

OpenShift Virtualization integration with the Compliance Operator 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/>.

1.1.1. OpenShift Virtualization supported cluster version

OpenShift Virtualization 4.11 is supported for use on OpenShift Container Platform 4.11 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. OPENSIFT VIRTUALIZATION ARCHITECTURE

Learn about OpenShift Virtualization architecture.

2.1. HOW OPENSIFT VIRTUALIZATION ARCHITECTURE WORKS

After you install OpenShift Virtualization, the Operator Lifecycle Manager (OLM) deploys operator pods for each component of OpenShift Virtualization:

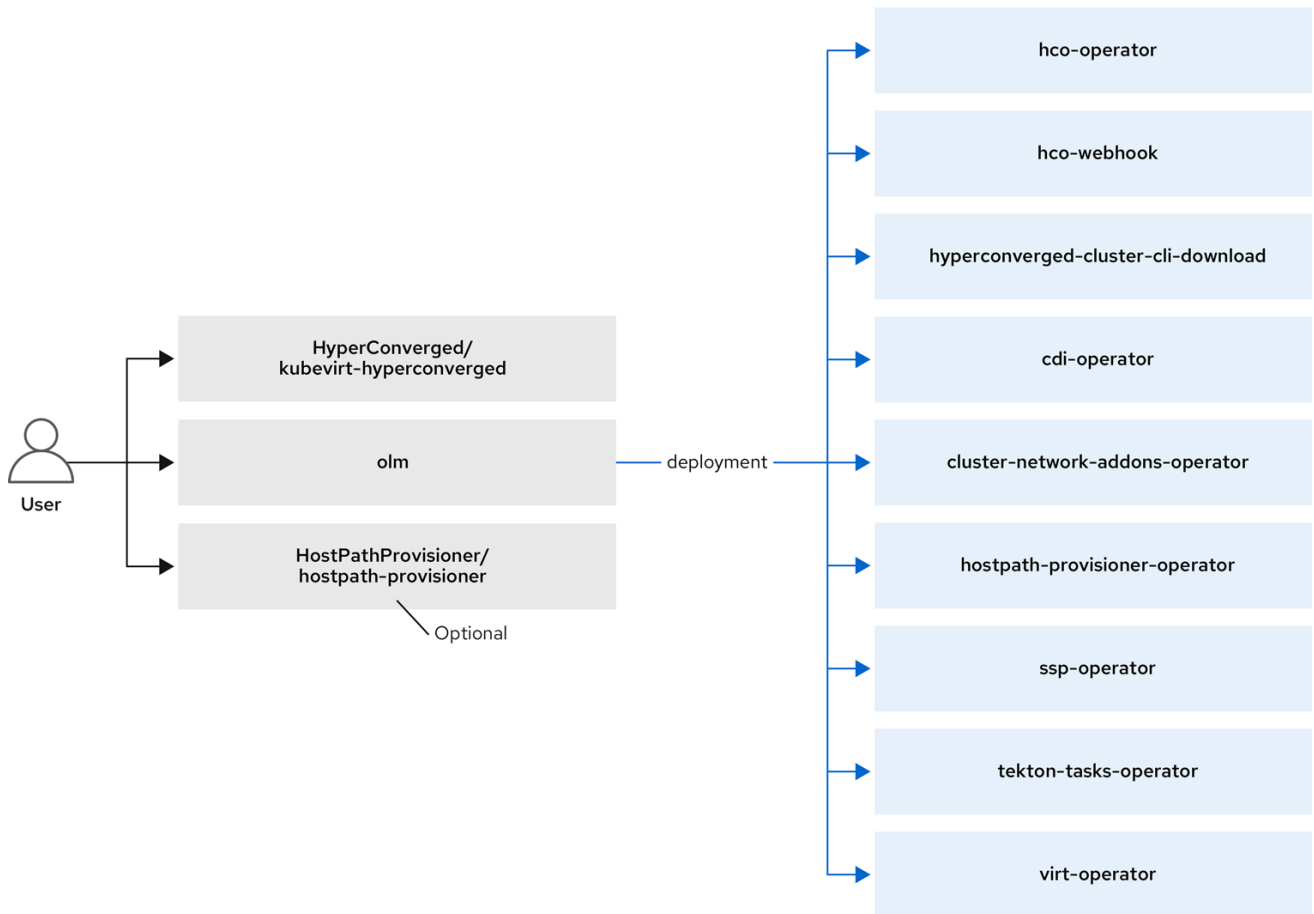
- Compute: **virt-operator**
- Storage: **cdi-operator**
- Network: **cluster-network-addons-operator**
- Scaling: **ssp-operator**
- Templating: **tekton-tasks-operator**

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

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

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

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



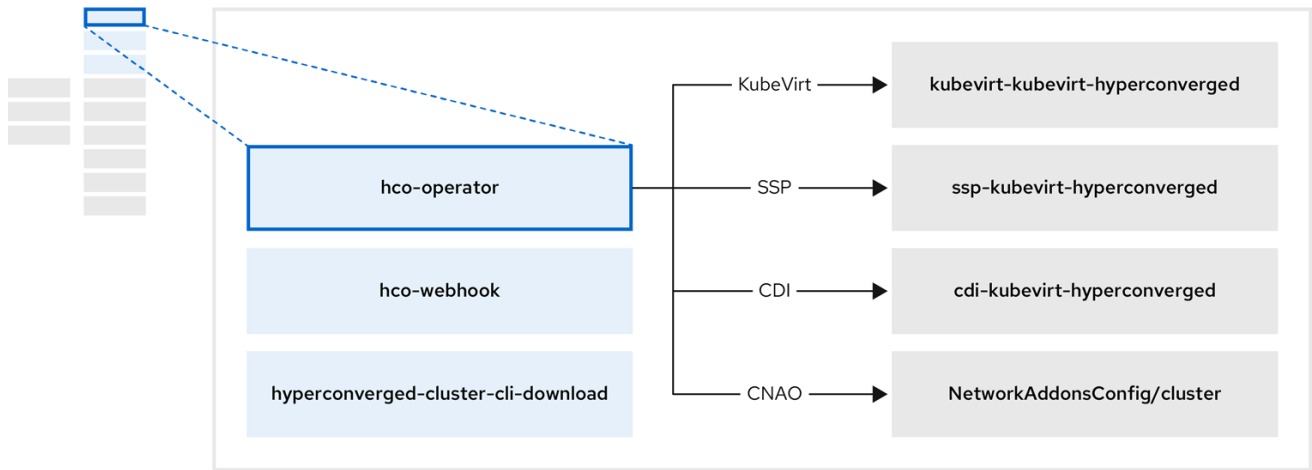
220_OpenShift_0722

Additional resources

- [HyperConverged CR configuration](#)
- [Virtctl client commands](#)
- [About the hostpath provisioner](#)

2.2. ABOUT THE HCO-OPERATOR

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



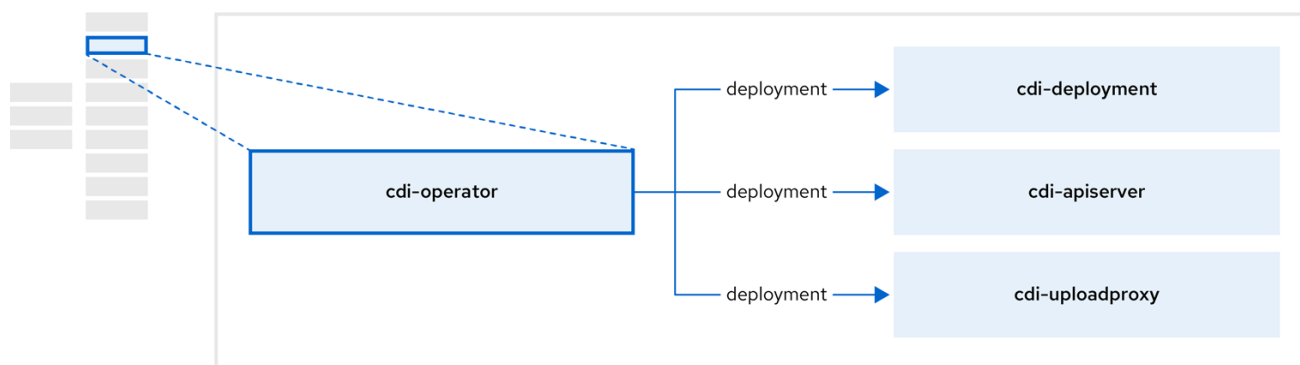
220_OpenShift_0722

Table 2.1. hco-operator components

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

2.3. ABOUT THE CDI-OPERATOR

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



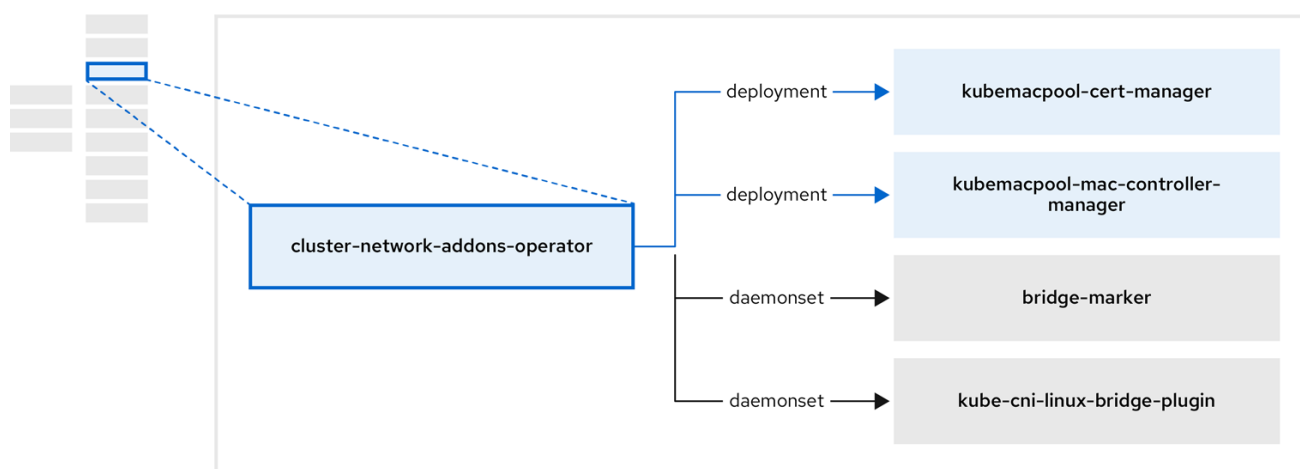
220_OpenShift_0722

Table 2.2. cdi-operator components

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

2.4. ABOUT THE CLUSTER-NETWORK-ADDONS-OPERATOR

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



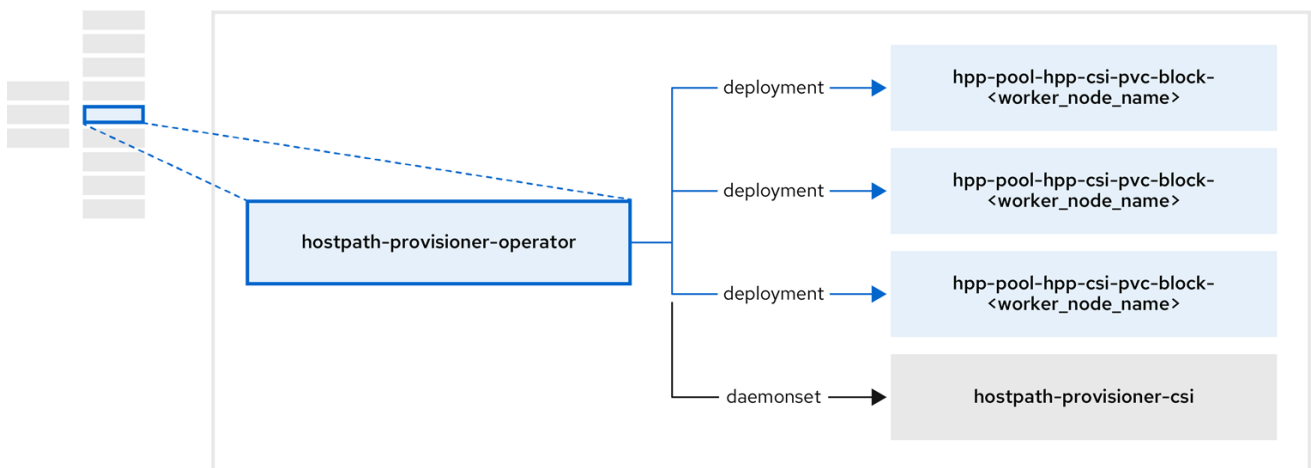
220_OpenShift_0722

Table 2.3. cluster-network-addons-operator components

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

2.5. ABOUT THE HOSTPATH-PROVISIONER-OPERATOR

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



220_OpenShift_0622

Table 2.4. hostpath-provisioner-operator components

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

2.6. ABOUT THE SSP-OPERATOR

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

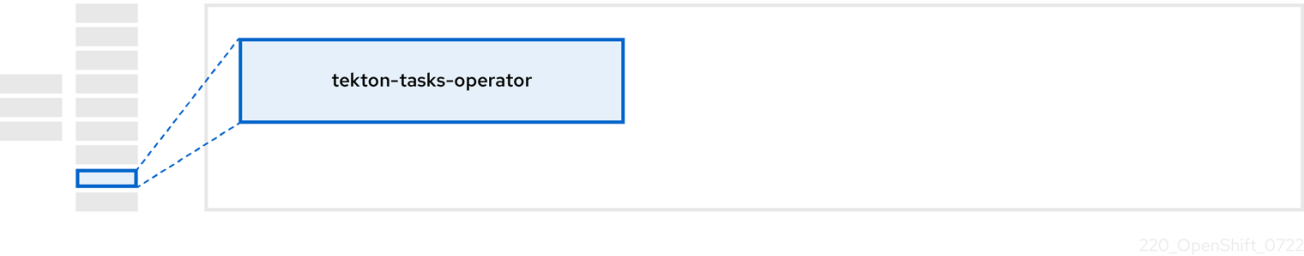


Table 2.5. ssp-operator components

Component	Description
deployment/virt-template-validator	Checks vm.kubevirt.io/validations annotations on virtual machines created from templates, and rejects them if they are invalid.

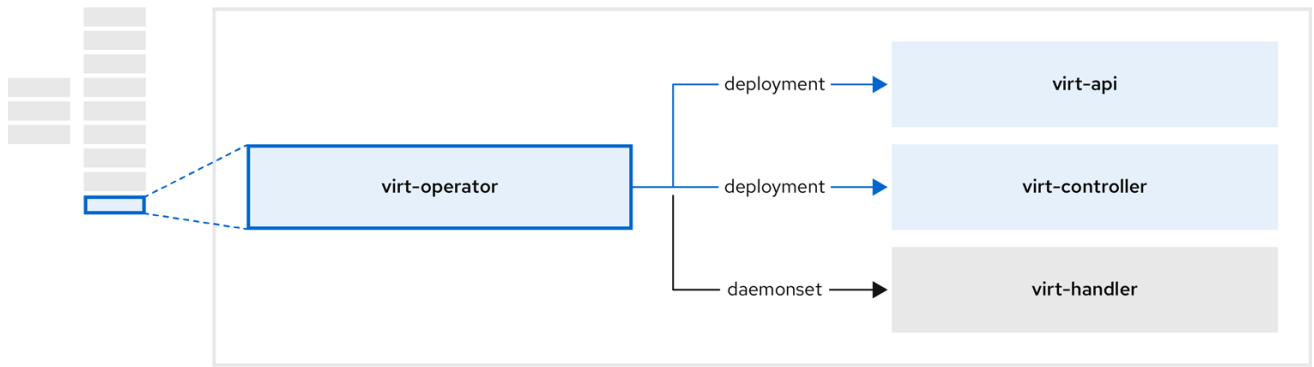
2.7. ABOUT THE TEKTON-TASKS-OPERATOR

The **tekton-tasks-operator** deploys example pipelines showing the usage of OpenShift Pipelines for VMs. It also deploys additional OpenShift Pipeline tasks that allow users to create VMs from templates, copy and modify templates, and create data volumes.



2.8. ABOUT THE VIRT-OPERATOR

The **virt-operator** deploys, upgrades, and manages OpenShift Virtualization without disrupting current virtual machine (VM) workloads.



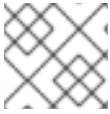
220_OpenShift_0622

Table 2.6. virt-operator components

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

CHAPTER 3. GETTING STARTED WITH OPENSIFT VIRTUALIZATION

You can install and configure a basic OpenShift Virtualization environment in order to explore its features and functionality.



NOTE

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

3.1. BEFORE YOU BEGIN

- [Prepare your cluster](#) for OpenShift Virtualization.
- Review the [storage requirements](#) for cloning, snapshots, and live migration.
- Install the [OpenShift Virtualization Operator](#).
- Install the [virtctl](#) tool.

3.1.1. Additional resources

- [Using a CSI-enabled storage provider](#).
- [Configuring local storage](#) for virtual machines.
- [About the Kubernetes NMState Operator](#).
- [Specifying nodes for virtual machines](#).

3.2. GETTING STARTED

Create a virtual machine:

- [Quick create](#) a virtual machine using the web console.
- Create and customize [Windows boot sources](#).
- Install [VirtIO drivers and the QEMU guest agent](#) on the virtual machine.

Connect to a virtual machine:

Connect to a virtual machine

- Connect to the [serial console](#) or [VNC console](#) of a virtual machine using the web console.
- Connect to a virtual machine [using SSH](#).
- Connect to a Windows virtual machine [using RDP](#).

Manage a virtual machine

- Stop, start, pause, and restart a virtual machine [from the web console](#).

- Manage a virtual machine, expose a port, and connect to the serial console of a virtual machine [from the command line](#) with **virtctl**.

3.3. NEXT STEPS

Configure additional networks

- Linux bridge network:
 - Create a [node network configuration policy](#)
 - Create a [network attachment definition](#).
 - [Attach a virtual machine](#) to the Linux bridge network.
- SR-IOV network:
 - Install the [SR-IOV Operator](#).
 - Configure an [SR-IOV network device](#).
 - [Attach a virtual machine](#) to the SR-IOV network.

Monitor your OpenShift Virtualization environment

- Monitor resources, details, status, and top consumers on the [Virtualization Overview](#) page.
- View high-level information about your virtual machines on the [Virtual Machines](#) dashboard.
- View virtual machine [logs](#).

Automate your deployments

- [Automate virtual machine deployments](#) with Ansible.
- [Automate Windows virtual machine deployments](#) with **sysprep**.

3.3.1. Additional resources

- [Creating virtual machine templates](#)
- [Live migration](#)
- [Virtual machine templates](#)
- [Configuring local storage](#)
- [Backup and restore](#)

CHAPTER 4. OPENSIFT VIRTUALIZATION RELEASE NOTES

4.1. 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](#).

4.2. 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](#).

Learn more about [OpenShift Virtualization architecture and deployments](#).

[Prepare your cluster](#) for OpenShift Virtualization.

4.2.1. OpenShift Virtualization supported cluster version

OpenShift Virtualization 4.11 is supported for use on OpenShift Container Platform 4.11 clusters. To use the latest z-stream release of OpenShift Virtualization, you must first upgrade to the latest version of OpenShift Container Platform.

4.2.2. Supported guest operating systems

To view the supported guest operating systems for OpenShift Virtualization, refer to [Certified Guest Operating Systems in Red Hat OpenStack Platform, Red Hat Virtualization and OpenShift Virtualization](#).

4.3. NEW AND CHANGED FEATURES

- You can now deploy OpenShift Virtualization on a [three-node cluster](#) with zero compute nodes.
- Virtual machines run as unprivileged workloads in *session mode* by default. This feature improves cluster security by mitigating escalation-of-privilege attacks.
- Red Hat Enterprise Linux (RHEL) 9 is now supported as a guest operating system.
- The link for installing the Migration Toolkit for Virtualization (MTV) Operator in the OpenShift Container Platform web console has been moved. It is now located in the **Related operators** section of the **Getting started resources** card on the **Virtualization → Overview** page.
- You can configure the [verbosity level](#) of the **virtLauncher**, **virtHandler**, **virtController**, **virtAPI**, and **virtOperator** pod logs to debug specific components by editing the **HyperConverged** custom resource (CR).

4.3.1. 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.

4.3.2. Storage

- New metrics are available that provide information about [virtual machine snapshots](#).
- You can reduce the number of logs on disconnected environments or reduce resource usage by [disabling the automatic imports and updates for a boot source](#) .

4.3.3. Web console

- You can set the [boot mode](#) of templates and virtual machines to **BIOS**, **UEFI**, or **UEFI (secure)** by using the web console.
- You can now enable and disable the descheduler from the web console on the **Scheduling** tab of the [VirtualMachine details](#) page.
- You can access virtual machines by navigating to **Virtualization** → **VirtualMachines** in the side menu. Each virtual machine now has an [updated Overview tab](#) that provides information about the virtual machine configuration, alerts, snapshots, network interfaces, disks, usage data, and hardware devices.
- The **Create a Virtual Machine** wizard in the web console is now [replaced by the Catalog page](#), which lists available templates that you can use to create a virtual machine. You can use a template with an available boot source to quickly create a virtual machine or you can customize a template to create a virtual machine.
- If your Windows virtual machine has a vGPU attached, you can now [switch between the default display and the vGPU display](#) by using the web console.
- You can access virtual machine templates by navigating to **Virtualization** → **Templates** in the side menu. The updated **VirtualMachine Templates** page now provides useful information about each template, including workload profile, boot source, and CPU and memory configuration.
- The **Create Template** wizard has been removed from the **VirtualMachine Templates** page. You [create a virtual machine template](#) by editing a YAML file example.

4.4. DEPRECATED AND REMOVED FEATURES

4.4.1. 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.

- In a future release, support for the legacy HPP custom resource, and the associated storage class, will be deprecated. Beginning in OpenShift Virtualization 4.11, the HPP Operator uses the Kubernetes Container Storage Interface (CSI) driver to configure local storage. The Operator

continues to support the existing (legacy) format of the HPP custom resource and the associated storage class. If you use the HPP Operator, plan to [create a storage class for the CSI driver](#) as part of your migration strategy.

4.4.2. Removed features

Removed features are not supported in the current release.

- OpenShift Virtualization 4.11 removes support for [nmstate](#), including the following objects:
 - **NodeNetworkState**
 - **NodeNetworkConfigurationPolicy**
 - **NodeNetworkConfigurationEnactment**

To preserve and support your existing nmstate configuration, install the [Kubernetes NMState Operator](#) before updating to OpenShift Virtualization 4.11. You can install it from the **OperatorHub** in the OpenShift Container Platform web console, or by using the OpenShift CLI (**oc**).

- The Node Maintenance Operator (NMO) is no longer shipped with OpenShift Virtualization. You can [install the NMO](#) from the **OperatorHub** in the OpenShift Container Platform web console, or by using the OpenShift CLI (**oc**).

You must perform one of the following tasks before updating to OpenShift Virtualization 4.11 from OpenShift Virtualization 4.10.2 and later releases:

- Move all nodes out of maintenance mode.
- Install the standalone NMO and replace the **nodemaintenances.nodemaintenance.kubevirt.io** custom resource (CR) with a **nodemaintenances.nodemaintenance.medik8s.io** CR.

4.5. 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 use Microsoft Windows 11 as a guest operating system. However, OpenShift Virtualization 4.11 does not support USB disks, which are required for a critical function of BitLocker recovery. To protect recovery keys, use other methods described in the [BitLocker recovery guide](#).
- You can now [deploy OpenShift Virtualization on AWS bare metal nodes](#).
- OpenShift Virtualization has [critical alerts](#) that inform you when a problem occurs that requires immediate attention. Now, each alert has a corresponding description of the problem, a reason for why the alert is occurring, a troubleshooting process to diagnose the source of the problem, and steps for resolving the alert.
- Administrators can now declaratively [create and expose mediated devices](#) such as virtual graphics processing units (vGPUs) by editing the **HyperConverged** CR. Virtual machine owners can then assign these devices to VMs.

- You can [transfer the static IP configuration of the NIC attached to the bridge](#) by applying a single **NodeNetworkConfigurationPolicy** manifest to the cluster.
- You can now install OpenShift Virtualization on IBM Cloud bare-metal servers. Bare-metal servers offered by other cloud providers are not supported.
- You can check your OpenShift Virtualization cluster for compliance issues by installing the [Compliance Operator](#) and running a scan with the [ocp4-moderate](#) and [ocp4-moderate-node](#) profiles.
- OpenShift Virtualization now includes a [diagnostic framework](#) to run predefined checkups that can be used for cluster maintenance and troubleshooting. You can run a predefined checkup to [check network connectivity and latency](#) for virtual machines on a secondary network.
- You can create live migration policies with specific parameters, such as bandwidth usage, maximum number of parallel migrations, and timeout, and apply the policies to groups of virtual machines by using virtual machine and namespace labels.

4.6. BUG FIXES


- Previously, on a large cluster, the OpenShift Virtualization MAC pool manager would take too much time to boot and OpenShift Virtualization might not become ready. With this update, the pool initialization and startup latency is reduced. As a result, VMs can now be successfully defined. ([BZ#2035344](#))
- If a Windows VM crashes or hangs during shutdown, you can now manually issue a force shutdown request to stop the VM. ([BZ#2040766](#))
- The YAML examples in the VM wizard have now been updated to contain the latest upstream changes. ([BZ#2055492](#))
- The **Add Network Interface** button on the VM **Network Interfaces** tab is no longer disabled for non-privileged users. ([BZ#2056420](#))
- A non-privileged user can now successfully add disks to a VM without getting a RBAC rule error. ([BZ#2056421](#))
- The web console now successfully displays virtual machine templates that are deployed to a custom namespace. ([BZ#2054650](#))
- Previously, updating a Single Node OpenShift (SNO) cluster failed if the **spec.evictionStrategy** field was set to **LiveMigrate** for a VMI. For live migration to succeed, the cluster must have more than one compute node. With this update, the **spec.evictionStrategy** field is removed from the virtual machine template in a SNO environment. As a result, cluster update is now successful. ([BZ#2073880](#))

4.7. KNOWN ISSUES

- The OVN-Kubernetes cluster network provider crashes from peak RAM and CPU usage if you create a large number of **NodePort** services. This can happen if you use **NodePort** services to expose SSH access to a large number of virtual machines (VMs). ([OCBUGS-1940](#))
 - As a workaround, use the OpenShift SDN cluster network provider if you want to expose SSH access to a large number of VMs via **NodePort** services.

- Updating to OpenShift Virtualization 4.11 from version 4.10 is blocked until you install the standalone Kubernetes NMState Operator. This occurs even if your cluster configuration does not use any [nmstate](#) resources. ([BZ#2126537](#))
 - As a workaround:
 1. Verify that there are no node network configuration policies defined on the cluster:


```
$ oc get nncp
```
 2. Choose the appropriate method to update OpenShift Virtualization:
 - a. If the list of node network configuration policies is not empty, exit this procedure and install the [Kubernetes NMState Operator](#) to preserve and support your existing nmstate configuration.
 - b. If the list is empty, go to step 3.
 3. Annotate the **HyperConverged** custom resource (CR). The following command overwrites any existing JSON patches:


```
$ oc annotate --overwrite -n openshift-cnv hco kubevirt-hyperconverged
'networkaddonsconfigs.kubevirt.io/jsonpatch=[{"op": "replace", "path":
"/spec/nmstate", "value": null}]'
```
- 

NOTE

The **HyperConverged** object reports a **TaintedConfiguration** condition while this patch is applied. This is benign.
4. Update OpenShift Virtualization.
 5. After the update completes, remove the annotation by running the following command:


```
$ oc annotate -n openshift-cnv hco kubevirt-hyperconverged
networkaddonsconfigs.kubevirt.io/jsonpatch-
```
 6. Optional: Add back any previously configured JSON patches that were overwritten.
- Some persistent volume claim (PVC) annotations created by the Containerized Data Importer (CDI) can cause the virtual machine snapshot restore operation to hang indefinitely. ([BZ#2070366](#))
 - As a workaround, you can remove the annotations manually:
 1. Obtain the [VirtualMachineSnapshotContent](#) custom resource (CR) name from the **status.virtualMachineSnapshotContentName** value in the **VirtualMachineSnapshot** CR.
 2. Edit the **VirtualMachineSnapshotContent** CR and remove all lines that contain **k8s.io/cloneRequest**.
 3. If you did not specify a value for **spec.dataVolumeTemplates** in the **VirtualMachine** object, delete any **DataVolume** and **PersistentVolumeClaim** objects in this namespace where both of the following conditions are true:

- a. The object's name begins with **restore-**.
- b. The object is not referenced by virtual machines.
This step is optional if you specified a value for **spec.dataVolumeTemplates**.

4. Repeat the [restore operation](#) with the updated **VirtualMachineSnapshot** CR.

- Windows 11 virtual machines do not boot on clusters running in [FIPS mode](#). Windows 11 requires a TPM (trusted platform module) device by default. However, the **swtpm** (software TPM emulator) package is incompatible with FIPS. ([BZ#2089301](#))
- In a Single Node OpenShift (SNO) cluster, a **VMCannotBeEvicted** alert occurs on virtual machines that are created from common templates that have the eviction strategy set to **LiveMigrate**. ([BZ#2092412](#))
- The QEMU guest agent on a Fedora 35 virtual machine is blocked by SELinux and does not report data. Other Fedora versions might be affected. ([BZ#2028762](#))
 - As a workaround, disable SELinux on the virtual machine, run the QEMU guest agent commands, and then re-enable SELinux.
- 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 device to a host's default interface 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.
- If you use Red Hat Ceph Storage or Red Hat OpenShift Data Foundation Storage, cloning more than 100 VMs at once might fail. ([BZ#1989527](#))
 - As a workaround, you can perform a host-assisted copy by setting **spec.cloneStrategy: copy** in the storage profile manifest. For example:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <provisioner_class>
# ...
spec:
  claimPropertySets:
    - accessModes:
        - ReadWriteOnce
      volumeMode: Filesystem
    cloneStrategy: copy ❶
  status:
    provisioner: <provisioner>
    storageClass: <provisioner_class>
```

❶ The default cloning method set to **copy**.

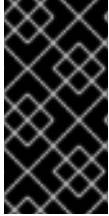
- In some instances, multiple virtual machines can mount the same PVC in read-write mode, which might result in data corruption. ([BZ#1992753](#))
 - As a workaround, avoid using a single PVC in read-write mode with multiple VMs.

- The Pod Disruption Budget (PDB) prevents pod disruptions for migratable virtual machine images. If the PDB detects pod disruption, then **openshift-monitoring** sends a **PodDisruptionBudgetAtLimit** alert every 60 minutes for virtual machine images that use the **LiveMigrate** eviction strategy. ([BZ#2026733](#))
 - As a workaround, [silence alerts](#).
- OpenShift Virtualization links a service account token in use by a pod to that specific pod. OpenShift Virtualization implements a service account volume by creating a disk image that contains a token. If you migrate a VM, then the service account volume becomes invalid. ([BZ#2037611](#))
 - As a workaround, use user accounts rather than service accounts because user account tokens are not bound to a specific pod.
- If you configure the **HyperConverged** custom resource (CR) to enable mediated devices before drivers are installed, the new device configuration does not take effect. This issue can be triggered by updates. For example, if **virt-handler** is updated before **daemonset**, which installs NVIDIA drivers, then nodes cannot provide virtual machine GPUs. ([BZ#2046298](#))
 - As a workaround:
 1. Remove **mediatedDevicesConfiguration** and **permittedHostDevices** from the **HyperConverged** CR.
 2. Update both **mediatedDevicesConfiguration** and **permittedHostDevices** stanzas with the configuration you want to use.
- If you clone more than 100 VMs using the **csi-clone** cloning strategy, then the Ceph CSI might not purge the clones. Manually deleting the clones can also fail. ([BZ#2055595](#))
 - As a workaround, you can restart the **ceph-mgr** to purge the VM clones.

CHAPTER 5. INSTALLING

5.1. 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.

Single-node OpenShift differences

You can install OpenShift Virtualization on a single-node cluster. See [About single-node OpenShift](#) for more information. Single-node OpenShift does not support high availability, which results in the following differences:

- [Pod disruption budgets](#) are not supported.
- [Live migration](#) is not supported.
- Templates or virtual machines that use data volumes or storage profiles must not have the [evictionStrategy](#) set.

FIPS mode

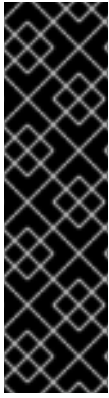
If you install your cluster in [FIPS mode](#), no additional setup is required for OpenShift Virtualization.

5.1.1. 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. See [Deploy OpenShift Virtualization on AWS Bare Metal Nodes](#) for details.
- IBM Cloud Bare Metal Servers. See [Deploy OpenShift Virtualization on IBM Cloud Bare Metal Nodes](#) for details.



IMPORTANT

Installing OpenShift Virtualization on AWS bare metal instances or on IBM Cloud Bare Metal Servers 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

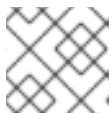
- Supported by Red Hat Enterprise Linux (RHEL) 8
- Support for Intel 64 or AMD64 CPU extensions
- Intel VT or AMD-V hardware virtualization extensions enabled
- NX (no execute) flag enabled

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.

- If your cluster uses worker nodes with different CPUs, live migration failures can occur because different CPUs have different capabilities. To avoid such failures, use CPUs with appropriate capacity for each node and set node affinity on your virtual machines to ensure successful migration. See [Configuring a required node affinity rule](#) for more information.

Additional resources

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

5.1.2. 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.

5.1.2.1. Memory overhead

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

Cluster memory overhead

Memory overhead per infrastructure node ≈ 150 MiB

Memory overhead per worker node ≈ 360 MiB

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

Virtual machine memory overhead

Memory overhead per virtual machine $\approx (1.002 * \text{requested memory}) + 146 \text{ MiB} \setminus$
 $+ 8 \text{ MiB} * (\text{number of vCPUs}) \setminus$ **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.

5.1.2.2. 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 ≈ 4 cores

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

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

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

Virtual machine CPU overhead

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

5.1.2.3. 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.

5.1.2.4. 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.

5.1.3. Object maximums

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

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

5.1.4. 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.

5.1.5. Live migration

Live migration has the following requirements:

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

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

5.1.6. Snapshots and cloning

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

5.1.7. 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.

- Automatic high availability for both IPI and non-IPI is available by using the [Node Health Check Operator](#) on the OpenShift Container Platform cluster to deploy the **NodeHealthCheck** controller. The controller identifies unhealthy nodes and uses the Self Node Remediation Operator to remediate the unhealthy nodes.



IMPORTANT

Node Health Check Operator 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/>.

- 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>**.

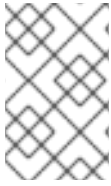


NOTE

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

5.2. 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.

5.2.1. 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.

5.2.1.1. 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.

5.2.1.2. 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-cnv
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.11.0
  channel: "stable"
  config: ❶
```

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

5.2.1.3. 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-cnv
spec:
  infra:
    nodePlacement: ❶
    ...
  workloads:
    nodePlacement:
    ...
```

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

5.2.1.4. 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.

5.2.1.5. 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](#)

5.2.2. Example manifests

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

5.2.2.1. Operator Lifecycle Manager Subscription object

5.2.2.1.1. 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.v4.11.0
channel: "stable"
config:
  nodeSelector:
    example.io/example-infra-key: example-infra-value

```

5.2.2.1.2. 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.v4.11.0
  channel: "stable"
  config:
    tolerations:
      - key: "key"
        operator: "Equal"
        value: "virtualization"
        effect: "NoSchedule"

```

5.2.2.2. HyperConverged object

5.2.2.2.1. 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

```

5.2.2.2.2. 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
```

5.2.2.2.3. 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"
```

5.2.2.3. HostPathProvisioner object

5.2.2.3.1. 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
```

5.3. 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 4.11 [web console](#) to subscribe to and deploy the OpenShift Virtualization Operators.

5.3.1. Installing the OpenShift Virtualization Operator

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

Prerequisites

- Install OpenShift Container Platform 4.11 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.

5.3.2. 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.

5.4. INSTALLING OPENSIFT VIRTUALIZATION USING THE CLI

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](#).

5.4.1. Prerequisites

- Install OpenShift Container Platform 4.11 on your cluster.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

5.4.2. 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-cnv
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.11.0
  channel: "stable" 1
```

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

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

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



NOTE

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

5.4.3. 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-cnv** namespace.

Procedure

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

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

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

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

Verification

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

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

The following output displays if deployment was successful:

Example output

```
NAME                                DISPLAY                VERSION  REPLACES  PHASE
kubevirt-hyperconverged-operator.v4.11.0  OpenShift Virtualization  4.11.0
Succeeded
```

5.4.4. 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.

5.5. ENABLING 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.

5.5.1. Downloading and installing the virtctl client

5.5.1.1. Downloading the virtctl client

Download the **virtctl** client by using the link provided in the **ConsoleCLIDownload** custom resource (CR).

Procedure

1. View the **ConsoleCLIDownload** object by running the following command:

```
$ oc get ConsoleCLIDownload virtctl-clidownloads-kubevirt-hyperconverged -o yaml
```

2. Download the **virtctl** client by using the link listed for your distribution.

5.5.1.2. Installing the virtctl client

Extract and install the **virtctl** client after downloading from the appropriate location for your operating system.

Prerequisites

- You must have downloaded the **virtctl** client.

Procedure

- For Linux:

1. Extract the tarball. The following CLI command extracts it into the same directory as the tarball:

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

2. Navigate the extracted folder hierarchy and run the following command to make the **virtctl** binary executable:

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

3. Move the **virtctl** binary to a directory in your **PATH** environment variable.
4. To check your path, run the following command:

```
$ echo $PATH
```

- For Windows users:

1. Unpack and unzip the archive.
2. Navigate the extracted folder hierarchy and double-click the **virtctl** executable file to install the client.
3. Move the **virtctl** binary to a directory in your **PATH** environment variable.
4. To check your path, run the following command:

```
C:\> path
```

- For macOS users:

1. Unpack and unzip the archive.
2. Move the **virtctl** binary to a directory in your **PATH** environment variable.
3. To check your path, run the following command:

```
echo $PATH
```

5.5.2. Installing the virtctl RPM package

You can install the **virtctl** client as an RPM after enabling the OpenShift Virtualization repository.

5.5.2.1. Enabling OpenShift Virtualization repositories

Enable the OpenShift Virtualization repository for your version of Red Hat Enterprise Linux (RHEL).

Prerequisites

- Your system is registered to a Red Hat account with an active subscription to the "Red Hat Container Native Virtualization" entitlement.

Procedure

- Enable the appropriate OpenShift Virtualization repository for your operating system by using the **subscription-manager** CLI tool.
 - To enable the repository for RHEL 8, run:

```
# subscription-manager repos --enable cnv-4.11-for-rhel-8-x86_64-rpms
```

- To enable the repository for RHEL 7, run:

```
# subscription-manager repos --enable rhel-7-server-cnv-4.11-rpms
```

5.5.2.2. Installing the virtctl client using the yum utility

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

Prerequisites

- You enabled an OpenShift Virtualization repository on your Red Hat Enterprise Linux (RHEL) system.

Procedure

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

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

5.5.3. Additional resources

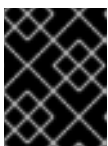
- [Using the CLI tools](#) for OpenShift Virtualization.

5.6. UNINSTALLING OPENSIFT VIRTUALIZATION USING THE WEB CONSOLE

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

5.6.1. Prerequisites

- You must have OpenShift Virtualization 4.11 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.


5.6.2. 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
```

5.6.3. 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.

5.6.4. Deleting a namespace using the web console

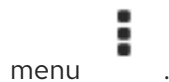
You can delete a namespace by using the OpenShift Container Platform web console.

**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



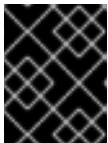
4. When the **Delete Namespace** pane opens, enter the name of the namespace that you want to delete in the field.
5. Click **Delete**.

5.7. UNINSTALLING OPENSIFT VIRTUALIZATION USING THE CLI

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

5.7.1. Prerequisites

- You must have OpenShift Virtualization 4.11 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.

5.7.2. 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.v4.11.0"
deleted
```


CHAPTER 6. UPDATING OPENSIFT VIRTUALIZATION

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

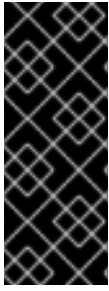


NOTE

- The Node Maintenance Operator (NMO) is no longer shipped with OpenShift Virtualization. You can [install the NMO](#) from the **OperatorHub** in the OpenShift Container Platform web console, or by using the OpenShift CLI (**oc**). You must perform one of the following tasks before updating to OpenShift Virtualization 4.11 from OpenShift Virtualization 4.10.2 and later releases:
 - Move all nodes out of maintenance mode.
 - Install the standalone NMO and replace the **nodemaintenances.nodemaintenance.kubevirt.io** custom resource (CR) with a **nodemaintenances.nodemaintenance.medik8s.io** CR.

6.1. ABOUT UPDATING OPENSIFT VIRTUALIZATION

- Operator Lifecycle Manager (OLM) manages the lifecycle of the OpenShift Virtualization Operator. The Marketplace Operator, which is deployed during OpenShift Container Platform installation, makes external Operators available to your cluster.
- OLM provides z-stream and minor version updates for OpenShift Virtualization. Minor version updates become available when you update OpenShift Container Platform to the next minor version. You cannot update OpenShift Virtualization to the next minor version without first updating OpenShift Container Platform.
- 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 update 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 4.11 on OpenShift Container Platform 4.11.
 - 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.
- Updating OpenShift Virtualization does not interrupt network connections.
- Data volumes and their associated persistent volume claims are preserved during update.



IMPORTANT

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

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

6.2. CONFIGURING AUTOMATIC WORKLOAD UPDATES

6.2.1. About workload updates

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



NOTE

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

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

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

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

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

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

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

Migration attempts and timeouts

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

5 minutes

If the pod is pending because it is **`Unschedulable`**.

15 minutes

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

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



NOTE

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

6.2.2. Configuring workload update methods

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

Prerequisites

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



NOTE

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

Procedure

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

```
$ oc edit hco -n openshift-cn v kubevirt-hyperconverged
```

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

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

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

2

The least disruptive update method. VMIs that support live migration are updated by migrating the virtual machine (VM) guest into a new pod with the updated components

- 3 A disruptive method that shuts down VMI pods during upgrade. **Evict** is the only update method available if live migration is not enabled in the cluster. If a VMI is controlled by a **VirtualMachine** object that has **runStrategy: always** configured, a new VMI is created in a new pod with updated components.
- 4 The number of VMIs that can be forced to be updated at a time by using the **Evict** method. This does not apply to the **LiveMigrate** method.
- 5 The interval to wait before evicting the next batch of workloads. This does not apply to the **LiveMigrate** method.



NOTE

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

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

6.3. APPROVING PENDING OPERATOR UPDATES

6.3.1. Manually approving a pending Operator upgrade

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 upgrade display a status with **Upgrade available**. Click the name of the Operator you want to upgrade.
3. Click the **Subscription** tab. Any upgrades 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 upgrade. When satisfied, click **Approve**.
6. Navigate back to the **Operators → Installed Operators** page to monitor the progress of the upgrade. When complete, the status changes to **Succeeded** and **Up to date**.

6.4. MONITORING UPDATE STATUS

6.4.1. Monitoring OpenShift Virtualization upgrade status

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



NOTE

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

Prerequisites

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

Procedure

1. Run the following command:

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

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

Example output

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

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

```
$ oc get 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

6.4.2. Viewing outdated OpenShift Virtualization workloads

You can view a list of outdated workloads by using the CLI.

**NOTE**

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

Procedure

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

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

**NOTE**

[Configure workload updates](#) to ensure that VMIs update automatically.

6.5. ADDITIONAL RESOURCES

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

CHAPTER 7. SECURITY POLICIES

Virtual machine (VM) workloads run as unprivileged pods. So that VMs can use OpenShift Virtualization features, some pods are granted custom security policies that are not available to other pod owners:

- An extended **container_t** SELinux policy applies to **virt-launcher** pods.
- [Security context constraints](#) (SCCs) are defined for the **kubevirt-controller** service account.

7.1. ABOUT WORKLOAD SECURITY

By default, virtual machine (VM) workloads do not run with root privileges in OpenShift Virtualization.

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

There are no supported OpenShift Virtualization features that require root privileges. If a feature requires root, it might not be supported for use with OpenShift Virtualization.

7.2. 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.

7.3. 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.

7.3.1. 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 plug-in.
- **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**.

7.3.2. 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
```

7.4. ADDITIONAL RESOURCES

- [Managing security context constraints](#)
- [Using RBAC to define and apply permissions](#)
- [Optimizing virtual machine network performance](#) in the Red Hat Enterprise Linux (RHEL) documentation
- [Configuring huge pages](#) in the RHEL documentation

CHAPTER 8. 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

8.1. PREREQUISITES

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

8.2. 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 8.1. **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.

8.3. 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 8.2. virtctl client commands

Command	Description
virtctl start <vm_name>	Start a virtual machine.
virtctl start --paused <vm_name>	Start a virtual machine in a paused state. This option enables you to interrupt the boot process from the VNC console.
virtctl stop <vm_name>	Stop a virtual machine.
virtctl stop <vm_name> --grace-period 0 --force	Force stop a virtual machine. This option might cause data inconsistency or data loss.
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.
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.

Command	Description
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 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.

8.4. CREATING A CONTAINER USING VIRTCTL GUESTFS

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

Procedure

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

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

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

8.5. LIBGUESTFS TOOLS AND VIRTCTL GUESTFS

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

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

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

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

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

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

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



NOTE

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

8.6. ADDITIONAL RESOURCES

- [Libguestfs: tools for accessing and modifying virtual machine disk images](#) .

CHAPTER 9. VIRTUAL MACHINES

9.1. CREATING VIRTUAL MACHINES

Use one of these procedures to create a virtual machine:

- Quick Start guided tour
- Quick create from the **Catalog**
- Pasting a pre-configured YAML file with the virtual machine wizard
- Using the CLI



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](#).



IMPORTANT

Due to differences in storage behavior, some virtual machine templates are incompatible with single-node OpenShift. To ensure compatibility, do not set the **evictionStrategy** field for any templates or virtual machines that use data volumes or storage profiles.

9.1.1. 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

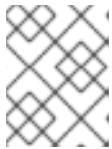
1. In the web console, select **Quick Starts** from the Help menu.
2. Click on a tile in the Quick Starts catalog. For example: **Creating a Red Hat Linux Enterprise Linux virtual machine**.
3. Follow the instructions in the guided tour and complete the tasks for importing an operating system image and creating a virtual machine. The **Virtualization** → **VirtualMachines** page displays the virtual machine.

9.1.2. Quick creating a virtual machine

You can quickly create a virtual machine (VM) by using a template with an available boot source.

Procedure

1. Click **Virtualization** → **Catalog** in the side menu.
2. Click **Boot source available** to filter templates with boot sources.



NOTE

By default, the template list will show only **Default Templates**. Click **All Items** when filtering to see all available templates for your chosen filters.

3. Click a template to view its details.
4. Click **Quick Create VirtualMachine** to create a VM from the template.
The virtual machine **Details** page is displayed with the provisioning status.

Verification

1. Click **Events** to view a stream of events as the VM is provisioned.
2. Click **Console** to verify that the VM booted successfully.

9.1.3. Creating a virtual machine from a customized template

Some templates require additional parameters, for example, a PVC with a boot source. You can customize select parameters of a template to create a virtual machine (VM).

Procedure

1. In the web console, select a template:
 - a. Click **Virtualization** → **Catalog** in the side menu.
 - b. Optional: Filter the templates by project, keyword, operating system, or workload profile.
 - c. Click the template that you want to customize.
2. Click **Customize VirtualMachine**.
3. Specify parameters for your VM, including its **Name** and **Disk source**. You can optionally specify a data source to clone.

Verification

1. Click **Events** to view a stream of events as the VM is provisioned.
2. Click **Console** to verify that the VM booted successfully.

Refer to the virtual machine fields section when creating a VM from the web console.

9.1.3.1. Virtual machine fields

The following table lists the virtual machine fields that you can edit in the OpenShift Container Platform web console:

Table 9.1. Virtual machine fields

Tab	Fields or functionality
Overview	<ul style="list-style-type: none"> ● Description ● CPU/Memory ● Boot mode ● GPU devices ● Host devices
YAML	<ul style="list-style-type: none"> ● View, edit, or download the custom resource.
Scheduling	<ul style="list-style-type: none"> ● Node selector ● Tolerations ● Affinity rules ● Dedicated resources ● Eviction strategy ● Descheduler setting
Environment	<ul style="list-style-type: none"> ● Add, edit, or delete a config map, secret, or service account.
Network Interfaces	<ul style="list-style-type: none"> ● Add, edit, or delete a network interface.
Disks	<ul style="list-style-type: none"> ● Add, edit, or delete a disk.

Tab	Fields or functionality
Scripts	<ul style="list-style-type: none"> • cloud-init settings • Authorized SSH key • Sysprep answer files
Metadata	<ul style="list-style-type: none"> • Labels • Annotations

9.1.3.1.1. 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	<p>List of available binding methods. Select the binding method suitable for the network interface:</p> <ul style="list-style-type: none"> • Default pod network: masquerade • Linux bridge network: bridge • SR-IOV network: SR-IOV
MAC Address	MAC address for the network interface controller. If a MAC address is not specified, one is assigned automatically.

9.1.3.1.2. 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 storage settings


The following advanced storage settings are optional and available for **Blank**, **Import via URL**, and **Clone existing PVC** disks. Before OpenShift Virtualization 4.11, if you do not specify these parameters, the system uses the default values from the **kubevirt-storage-class-defaults** config map. In OpenShift Virtualization 4.11 and later, the system uses the default values from the [storage profile](#).



NOTE

Use storage profiles to ensure consistent advanced storage settings when provisioning storage for OpenShift Virtualization.

To manually specify **Volume Mode** and **Access Mode**, you must clear the **Apply optimized StorageProfile settings** checkbox, which is selected by default.

Name	Mode description	Parameter	Parameter description
Volume Mode	Defines whether the persistent volume uses a formatted file system or raw block state. Default is Filesystem .	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	Access mode of the persistent volume.	ReadWriteOnce (RWO)	Volume can be mounted as read-write by a single node.
		ReadWriteMany (RWX)	Volume can be mounted as read-write by many nodes at one time.  NOTE This is required for some features, such as live migration of virtual machines between nodes.
		ReadOnlyMany (ROX)	Volume can be mounted as read only by many nodes.

9.1.3.1.3. 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](#).

9.1.3.2. 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 **Virtualization** → **VirtualMachines** from the side menu.
2. Click **Create** and select **With YAML**.
3. 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.
4. Optional: Click **Download** to download the YAML configuration file in its present state.
5. Click **Create** to create the virtual machine.

The virtual machine is listed on the **VirtualMachines** page.

9.1.4. 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 9.1. 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>
```

9.1.5. Virtual machine storage volume types

Storage volume type	Description
ephemeral	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.
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	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.

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>  <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> </div>
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>

9.1.6. 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
```

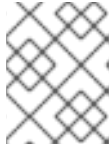
```
  template:
```

```
  ...
```

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

9.1.7. Additional resources

- The **VirtualMachineSpec** definition in the [KubeVirt v0.53.2 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.

- Enable the [CPU Manager](#) to use the high-performance workload profile.
- See [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.
- [Customizing the storage profile](#)

9.2. 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.

9.2.1. 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.

You can edit labels and annotations for any templates, including those provided by Red Hat. Other fields are editable for user-customized templates only.

Procedure

1. Click **Virtualization** → **VirtualMachines** from the side menu.
2. Optional: Use the **Filter** drop-down menu to sort the list of virtual machines by attributes such as status, template, node, or operating system (OS).
3. Select a virtual machine to open the **VirtualMachine details** page.
4. Click any field that has the pencil icon, which indicates that the field is editable. For example, click the current **Boot mode** setting, such as BIOS or UEFI, to open the **Boot mode** window and select an option from the list.
5. 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.

9.2.1.1. Virtual machine fields

The following table lists the virtual machine fields that you can edit in the OpenShift Container Platform web console:

Table 9.2. Virtual machine fields

Tab	Fields or functionality
Details	<ul style="list-style-type: none"> ● Labels ● Annotations ● Description ● CPU/Memory ● Boot mode ● Boot order ● GPU devices ● Host devices ● SSH access
YAML	<ul style="list-style-type: none"> ● View, edit, or download the custom resource.
Scheduling	<ul style="list-style-type: none"> ● Node selector ● Tolerations ● Affinity rules ● Dedicated resources ● Eviction strategy ● Descheduler setting
Network Interfaces	<ul style="list-style-type: none"> ● Add, edit, or delete a network interface.

Tab	Fields or functionality
Disks	<ul style="list-style-type: none"> ● Add, edit, or delete a disk.
Scripts	<ul style="list-style-type: none"> ● cloud-init settings
Snapshots	<ul style="list-style-type: none"> ● Add, restore, or delete a virtual machine snapshot.

9.2.2. 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.



NOTE

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

Procedure

1. Click **Virtualization** → **VirtualMachines** 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.

9.2.3. 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>
```

9.2.4. Adding a virtual disk to a virtual machine

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

Procedure

1. Click **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** screen.
3. Click the **Disks** tab and then click **Add disk**.
4. In the **Add disk** window, specify the **Source**, **Name**, **Size**, **Type**, **Interface**, and **Storage Class**.
 - a. Optional: You can enable preallocation if you use a blank disk source and require maximum write performance when creating data volumes. To do so, select the **Enable preallocation** checkbox.
 - b. Optional: You can clear **Apply optimized StorageProfile settings** to change the **Volume Mode** and **Access Mode** for the virtual disk. If you do not specify these parameters, the system uses the default values from the **kubevirt-storage-class-defaults** config map.
5. Click **Add**.



NOTE

If the 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](#).

9.2.4.1. Editing CD-ROMs for VirtualMachines

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

Procedure

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

9.2.4.2. 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

Name	Selection	Description
Interface		Type of disk device. Supported interfaces are virtIO , SATA , and SCSI .
Storage Class		The storage class that is used to create the disk.

Advanced storage settings

The following advanced storage settings are optional and available for **Blank**, **Import via URL**, and **Clone existing PVC** disks. Before OpenShift Virtualization 4.11, if you do not specify these parameters, the system uses the default values from the **kubevirt-storage-class-defaults** config map. In OpenShift Virtualization 4.11 and later, the system uses the default values from the [storage profile](#).




NOTE

Use storage profiles to ensure consistent advanced storage settings when provisioning storage for OpenShift Virtualization.

To manually specify **Volume Mode** and **Access Mode**, you must clear the **Apply optimized StorageProfile settings** checkbox, which is selected by default.

Name	Mode description	Parameter	Parameter description
Volume Mode	Defines whether the persistent volume uses a formatted file system or raw block state. Default is Filesystem .	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	Access mode of the persistent volume.	ReadWriteOnce (RWO)	Volume can be mounted as read-write by a single node.

Name	Mode description	Parameter	Parameter description
		ReadWriteMany (RWX)	Volume can be mounted as read-write by many nodes at one time.  NOTE This is required for some features, such as live migration of virtual machines between nodes.
		ReadOnlyMany (ROX)	Volume can be mounted as read only by many nodes.

9.2.5. Adding a network interface to a virtual machine

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

Procedure

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

9.2.5.1. 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	<p>List of available binding methods. Select the binding method suitable for the network interface:</p> <ul style="list-style-type: none"> • Default pod network: masquerade • Linux bridge network: bridge • SR-IOV network: SR-IOV
MAC Address	MAC address for the network interface controller. If a MAC address is not specified, one is assigned automatically.

9.2.6. Additional resources

- [Customizing the storage profile](#)

9.3. 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.

9.3.1. 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 **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Details** tab.
4. Click the pencil icon that is located on the right side of **Boot Order**. If a YAML configuration

does not exist, or if this is the first time that you are creating a boot order list, the following message displays: **No resource selected. VM will attempt to boot from disks by order of appearance in YAML file.**

5. Click **Add Source** and select a bootable disk or network interface controller (NIC) for the virtual machine.
6. Add any additional disks or NICs to the boot order list.
7. Click **Save**.



NOTE

If the virtual machine is running, changes to **Boot Order** will not take effect until you restart the virtual machine.

You can view pending changes by clicking **View Pending Changes** on the right side of the **Boot Order** field. The **Pending Changes** banner at the top of the page displays a list of all changes that will be applied when the virtual machine restarts.

9.3.2. Editing a boot order list in the web console

Edit the boot order list in the web console.

Procedure

1. Click **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Details** tab.
4. Click the pencil icon that is located on the right side of **Boot Order**.
5. Choose the appropriate method to move the item in the boot order list:
 - If you do not use a screen reader, hover over the arrow icon next to the item that you want to move, drag the item up or down, and drop it in a location of your choice.
 - If you use a screen reader, press the Up Arrow key or Down Arrow key to move the item in the boot order list. Then, press the **Tab** key to drop the item in a location of your choice.
6. Click **Save**.



NOTE

If the virtual machine is running, changes to the boot order list will not take effect until you restart the virtual machine.

You can view pending changes by clicking **View Pending Changes** on the right side of the **Boot Order** field. The **Pending Changes** banner at the top of the page displays a list of all changes that will be applied when the virtual machine restarts.

9.3.3. 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.

9.3.4. 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 **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Details** tab.
4. Click the pencil icon that is located on the right side of **Boot Order**.
5. Click the **Remove** icon  next to the item. The item is removed from the boot order list and saved in the list of available boot sources. If you remove all items from the boot order list, the following message displays: **No resource selected. VM will attempt to boot from disks by**

order of appearance in YAML file.



NOTE

If the virtual machine is running, changes to **Boot Order** will not take effect until you restart the virtual machine.

You can view pending changes by clicking **View Pending Changes** on the right side of the **Boot Order** field. The **Pending Changes** banner at the top of the page displays a list of all changes that will be applied when the virtual machine restarts.

9.4. DELETING VIRTUAL MACHINES

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

9.4.1. 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 Container Platform console, click **Virtualization** → **VirtualMachines** from the side menu.
2. Click the Options menu  of the virtual machine that you want to delete and select **Delete**.
 - Alternatively, click the virtual machine name to open the **VirtualMachine details** page and click **Actions** → **Delete**.
3. In the confirmation pop-up window, click **Delete** to permanently delete the virtual machine.

9.4.2. 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.

9.5. 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 by using **oc** or **virtctl** commands from the command-line interface (CLI).

The **virtctl** command provides more virtualization options than the **oc** command. For example, you can use **virtctl** to pause a VM or expose a port.

9.5.1. 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.

9.5.2. 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
```

9.5.3. 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 **Virtualization** → **VirtualMachines** from the side menu.
You can identify a standalone VMI by a dark colored badge next to its name.

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

You can edit the annotations and labels of a standalone virtual machine instance (VMI) using the web console. Other fields are not editable.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization** → **VirtualMachines** from the side menu.
2. Select a standalone VMI to open the **VirtualMachineInstance details** page.
3. On the **Details** tab, click the pencil icon beside **Annotations** or **Labels**.
4. Make the relevant changes and click **Save**.

9.5.5. 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>
```

9.5.6. 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 **Virtualization** → **VirtualMachines** from the side menu.
2. Click **Actions** → **Delete VirtualMachineInstance**.
3. In the confirmation pop-up window, click **Delete** to permanently delete the standalone VMI.

9.6. CONTROLLING VIRTUAL MACHINE STATES


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

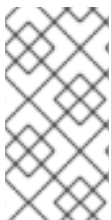
You can use **virtctl** to manage virtual machine states and perform other actions from the CLI. For example, you can use **virtctl** to force stop a VM or expose a port.

9.6.1. Starting a virtual machine

You can start a virtual machine from the web console.

Procedure

1. Click **Virtualization** → **VirtualMachines** from the side menu.
2. Find the row that contains the virtual machine that you want to start.
3. Navigate to the appropriate menu for your use case:
 - To stay on this page, where you can perform actions on multiple 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 **VirtualMachine details** page by clicking the name of the virtual machine.
 - b. Click **Actions**.
4. Select **Restart**.
5. In the confirmation window, click **Start** to start the virtual machine.

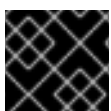


NOTE

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.

9.6.2. 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 **Virtualization** → **VirtualMachines** from the side menu.
2. Find the row that contains the virtual machine that you want to restart.
3. Navigate to the appropriate menu for your use case:
 - To stay on this page, where you can perform actions on multiple 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 **VirtualMachine details** page by clicking the name of the virtual machine.
 - b. Click **Actions** → **Restart**.
4. In the confirmation window, click **Restart** to restart the virtual machine.

9.6.3. Stopping a virtual machine

You can stop a virtual machine from the web console.

Procedure

1. Click **Virtualization** → **VirtualMachines** from the side menu.
2. Find the row that contains the virtual machine that you want to stop.
3. Navigate to the appropriate menu for your use case:
 - To stay on this page, where you can perform actions on multiple 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 **VirtualMachine details** page by clicking the name of the virtual machine.
 - b. Click **Actions** → **Stop**.
4. In the confirmation window, click **Stop** to stop the virtual machine.

9.6.4. 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 **Virtualization** → **VirtualMachines** from the side menu.
2. Find the row that contains the virtual machine that you want to unpause.
3. Navigate to the appropriate menu for your use case:
 - To stay on this page, where you can perform actions on multiple 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 **VirtualMachine details** page by clicking the name of the virtual machine.
 - b. Click the pencil icon that is located on the right side of **Status**.
4. In the confirmation window, click **Unpause** to unpause the virtual machine.

9.7. 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.

9.7.1. 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.

9.7.1.1. Connecting to the serial console

Connect to the serial console of a running virtual machine from the **Console** tab on the **VirtualMachine details** page of the web console.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Console** tab. The VNC console opens by default.
4. Click **Disconnect** to ensure that only one console session is open at a time. Otherwise, the VNC console session remains active in the background.

5. Click the **VNC Console** drop-down list and select **Serial Console**.
6. Click **Disconnect** to end the console session.
7. Optional: Open the serial console in a separate window by clicking **Open Console in New Window**.

9.7.1.2. Connecting to the VNC console

Connect to the VNC console of a running virtual machine from the **Console** tab on the **VirtualMachine details** page of the web console.

Procedure

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

9.7.1.3. 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 on the **VirtualMachine Details** page 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 Container Platform console, click **Virtualization** → **VirtualMachines** from the side menu.
2. Click a Windows virtual machine to open the **VirtualMachine details** page.
3. Click the **Console** tab.
4. In the **Console** list, select **Desktop Viewer**.

5. In the **Network Interface** list, select the layer-2 NIC.
6. Click **Launch Remote Desktop** to download the **console.rdp** file.
7. Open an RDP client and reference the **console.rdp** file. For example, using **remmina**:

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

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

9.7.1.4. Switching between virtual machine displays

If your Windows virtual machine (VM) has a vGPU attached, you can switch between the default display and the vGPU display by using the web console.

Prerequisites

- The mediated device is configured in the **HyperConverged** custom resource and assigned to the VM.
- The VM is running.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization** → **VirtualMachines**
2. Select a Windows virtual machine to open the **Overview** screen.
3. Click the **Console** tab.
4. From the list of consoles, select **VNC console**.
5. Choose the appropriate key combination from the **Send Key** list:
 - a. To access the default VM display, select **Ctl + Alt + 1**.
 - b. To access the vGPU display, select **Ctl + Alt + 2**.

Additional resources

- [Configuring mediated devices](#)

9.7.2. Accessing virtual machine consoles by using CLI commands

9.7.2.1. 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.

3. 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
```

9.7.2.2. 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: |
                  #cloud-config
                  user: fedora
                  password: fedora
                  chpasswd: {expire: False}
# ...

```

- 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	v1.24.0	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
```

9.7.2.3. 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>
```

9.7.2.4. 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
```

9.7.2.5. 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.

9.8. AUTOMATING WINDOWS INSTALLATION WITH SYSPREP

You can use Microsoft DVD images and **sysprep** to automate the installation, setup, and software provisioning of Windows virtual machines.

9.8.1. Using a Windows DVD to create a VM disk image

Microsoft does not provide disk images for download, but you can create a disk image using a Windows DVD. This disk image can then be used to create virtual machines.

Procedure

1. In the OpenShift Virtualization web console, click **Storage → PersistentVolumeClaims → Create PersistentVolumeClaim With Data upload form**
2. Select the intended project.
3. Set the **Persistent Volume Claim Name**
4. Upload the VM disk image from the Windows DVD. The image is now available as a boot source to create a new Windows VM.

9.8.2. Using a disk image to install Windows

You can use a disk image to install Windows on your virtual machine.

Prerequisites

- You must create a disk image using a Windows DVD.

- You must create an **autounattend.xml** answer file. See the [Microsoft documentation](#) for details.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization** → **Catalog** from the side menu.
2. Select a Windows template and click **Customize VirtualMachine**.
3. Select **Upload (Upload a new file to a PVC)** from the **Disk source** list and browse to the DVD image.
4. Click **Review and create VirtualMachine**.
5. Clear **Clone available operating system source to this Virtual Machine**
6. Clear **Start this VirtualMachine after creation**
7. On the **Sysprep** section of the **Scripts** tab, click **Edit**.
8. Browse to the **autounattend.xml** answer file and click **Save**.
9. Click **Create VirtualMachine**.
10. On the **YAML** tab, replace **running:false** with **runStrategy: RerunOnFailure** and click **Save**.


The VM will start with the **sysprep** disk containing the **autounattend.xml** answer file.

9.8.3. Generalizing a Windows VM using sysprep

Generalizing an image allows that image to remove all system-specific configuration data when the image is deployed on a virtual machine (VM).

Before generalizing the VM, you must ensure the **sysprep** tool cannot detect an answer file after the unattended Windows installation.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization** → **VirtualMachines**.
2. Select a Windows VM to open the **VirtualMachine details** page.
3. Click the **Disks** tab.
4. Click the Options menu  for the **sysprep** disk and select **Detach**.
5. Click **Detach**.
6. Rename **C:\Windows\Panther\unattend.xml** to avoid detection by the **sysprep** tool.
7. Start the **sysprep** program by running the following command:

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

8. After the **sysprep** tool completes, the Windows VM shuts down. The disk image of the VM is now available to use as an installation image for Windows VMs.

You can now specialize the VM.

9.8.4. Specializing a Windows virtual machine

Specializing a virtual machine (VM) configures the computer-specific information from a generalized Windows image onto the VM.

Prerequisites

- You must have a generalized Windows disk image.
- You must create an **unattend.xml** answer file. See the [Microsoft documentation](#) for details.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization** → **Catalog**.
2. Select a Windows template and click **Customize VirtualMachine**.
3. Select **PVC (clone PVC)** from the **Disk source** list.
4. Specify the **Persistent Volume Claim project** and **Persistent Volume Claim name** of the generalized Windows image.
5. Click **Review and create VirtualMachine**.
6. Click the **Scripts** tab.
7. In the **Sysprep** section, click **Edit**, browse to the **unattend.xml** answer file, and click **Save**.
8. Click **Create VirtualMachine**.

During the initial boot, Windows uses the **unattend.xml** answer file to specialize the VM. The VM is now ready to use.

9.8.5. Additional resources

- [Creating virtual machines](#)
- [Microsoft, Sysprep \(Generalize\) a Windows installation](#)
- [Microsoft, generalize](#)
- [Microsoft, specialize](#)

9.9. 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.

9.9.1. 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**).

9.9.2. 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.

9.9.3. 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.

9.9.3.1. 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
```

9.10. 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.

9.10.1. 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.

To check if your virtual machine (VM) has the QEMU guest agent installed and running, verify that **AgentConnected** is listed in the VM spec.



NOTE

To create snapshots of an online (Running state) VM with the highest integrity, install the QEMU guest agent.

The QEMU guest agent takes a consistent snapshot by attempting to quiesce the VM's file system as much as possible, depending on the system workload. This ensures that in-flight I/O is written to the disk before the snapshot is taken. If the guest agent is not present, quiescing is not possible and a best-effort snapshot is taken. The conditions under which the snapshot was taken are reflected in the snapshot indications that are displayed in the web console or CLI.

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
```

9.10.2. Installing QEMU guest agent on a Windows virtual machine

For Windows virtual machines, the QEMU guest agent is included in the VirtIO drivers. Install the drivers on an existing or a new Windows installation.

To check if your virtual machine (VM) has the QEMU guest agent installed and running, verify that **AgentConnected** is listed in the VM spec.



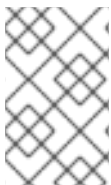
NOTE

To create snapshots of an online (Running state) VM with the highest integrity, install the QEMU guest agent.

The QEMU guest agent takes a consistent snapshot by attempting to quiesce the VM's file system as much as possible, depending on the system workload. This ensures that in-flight I/O is written to the disk before the snapshot is taken. If the guest agent is not present, quiescing is not possible and a best-effort snapshot is taken. The conditions under which the snapshot was taken are reflected in the snapshot indications that are displayed in the web console or CLI.

9.10.2.1. 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.

9.10.2.2. 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.

9.11. 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.

9.11.1. Prerequisites

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

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

When the QEMU guest agent is installed, the **Overview** and **Details** tabs on the **VirtualMachine details** page displays information about the hostname, operating system, time zone, and logged in users.

The **VirtualMachine details** page 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 **Overview** and the **Details** tabs display information about the operating system that was specified when the virtual machine was created.

9.11.3. 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 **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine name to open the **VirtualMachine details** page.
3. Click the **Details** tab to view active users.
4. Click the **Disks** tab to view information about the file systems.

9.12. 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.

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

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

These resources are added to the virtual machine as disks. You then mount the secret, config map, or service account as you would mount any other disk.

If the virtual machine is running, changes 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.

Prerequisites

- The secret, config map, or service account that you want to add must exist in the same namespace as the target virtual machine.

Procedure

1. Click **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. In the **Environment** tab, click **Add Config Map, Secret or Service Account**
4. Click **Select a resource** and select a resource from the list. A six character serial number is automatically generated for the selected resource.
5. Optional: Click **Reload** to revert the environment to its last saved state.
6. Click **Save**.

Verification

1. On the **VirtualMachine details** page, click the **Disks** tab and verify that the secret, config map, or service account is included in the list of disks.
2. Restart the virtual machine by clicking **Actions** → **Restart**.

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


9.12.2. 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 **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Environment** tab.
4. Find the item that you want to delete in the list, and click **Remove**  on the right side of the item.
5. Click **Save**.



NOTE

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

Verification

1. On the **VirtualMachine details** 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.

9.12.3. Additional resources

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

9.13. INSTALLING VIRTIO DRIVER ON AN EXISTING WINDOWS VIRTUAL MACHINE

9.13.1. About 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](#).

9.13.2. Supported VirtIO drivers for Microsoft Windows virtual machines

Table 9.3. 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.

9.13.3. 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

- 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
```

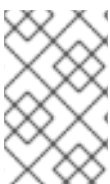
- OpenShift Virtualization boots virtual machine disks in the order defined in the **VirtualMachine** configuration file. You can either define other disks for the virtual machine before the **container-native-virtualization/virtio-win** container disk or use the optional **bootOrder** parameter to ensure 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.

- 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.

9.13.4. 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.

9.13.5. 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.

9.14. INSTALLING VIRTIO DRIVER ON A NEW WINDOWS VIRTUAL MACHINE

9.14.1. 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.

9.14.2. About 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](#).

9.14.3. Supported VirtIO drivers for Microsoft Windows virtual machines

Table 9.4. 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.

9.14.4. 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 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.

9.14.5. 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.

9.14.6. 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.

9.15. USING VIRTUAL TRUSTED PLATFORM MODULE DEVICES

Add a virtual Trusted Platform Module (vTPM) device to a new or existing virtual machine by editing the **VirtualMachine** (VM) or **VirtualMachineInstance** (VMI) manifest.

9.15.1. About vTPM devices

A virtual Trusted Platform Module (vTPM) device functions like a physical Trusted Platform Module (TPM) hardware chip.

You can use a vTPM device with any operating system, but Windows 11 requires the presence of a TPM chip to install or boot. A vTPM device allows VMs created from a Windows 11 image to function without a physical TPM chip.

If you do not enable vTPM, then the VM does not recognize a TPM device, even if the node has one.

vTPM devices also protect virtual machines by temporarily storing secrets without physical hardware. However, using vTPM for persistent secret storage is not currently supported. vTPM discards stored secrets after a VM shuts down.

9.15.2. Adding a vTPM device to a virtual machine

Adding a virtual Trusted Platform Module (vTPM) device to a virtual machine (VM) allows you to run a VM created from a Windows 11 image without a physical TPM device. A vTPM device also temporarily stores secrets for that VM.

Procedure

1. Run the following command to update the VM configuration:

```
$ oc edit vm <vm_name>
```

2. Edit the VM **spec** so that it includes the **tpm: {}** line. For example:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          tpm: {} 1
  ...
```

1 Adds the TPM device to the VM.

3. To apply your changes, save and exit the editor.
4. Optional: If you edited a running virtual machine, you must restart it for the changes to take effect.

9.16. ADVANCED VIRTUAL MACHINE MANAGEMENT

9.16.1. Working with resource quotas for virtual machines

Create and manage resource quotas for virtual machines.

9.16.1.1. 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.

9.16.1.2. Additional resources

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

9.16.2. Specifying nodes for virtual machines

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

9.16.2.1. 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.

9.16.2.2. Node placement examples

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

9.16.2.2.1. 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
  ...

```

9.16.2.2.2. 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: ❶
      - labelSelector:
          matchExpressions:
            - key: example-key-1
              operator: In
              values:
                - example-value-1
          topologyKey: kubernetes.io/hostname
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution: ❷
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
              - key: example-key-2
                operator: In
                values:
                  - example-value-2
          topologyKey: kubernetes.io/hostname
  ...
```

❶ If you use the **requiredDuringSchedulingIgnoredDuringExecution** rule type, the VM is not scheduled if the constraint is not met.

❷ 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.

9.16.2.2.3. 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: ❶
      nodeSelectorTerms:
        - matchExpressions:
            - key: example.io/example-key
              operator: In
              values:
                - example-value-1
                - example-value-2
      preferredDuringSchedulingIgnoredDuringExecution: ❷
        - weight: 1
          preference:
            matchExpressions:
              - key: example-node-label-key
                operator: In
                values:
                  - example-node-label-value
  ...

```

❶ If you use the **requiredDuringSchedulingIgnoredDuringExecution** rule type, the VM is not scheduled if the constraint is not met.

❷ 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.

9.16.2.2.4. 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"
```

```
...
```

9.16.2.3. 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](#)

9.16.3. Configuring certificate rotation

Configure certificate rotation parameters to replace existing certificates.

9.16.3.1. 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 ❶
    server:
      duration: 24h0m0s ❷
      renewBefore: 12h0m0s ❸
```

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

9.16.3.2. 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.

9.16.4. 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.

9.16.4.1. 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.

9.16.4.2. 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
```



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](#).

9.16.4.3. 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](#)

9.16.5. Using UEFI mode for virtual machines

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

9.16.5.1. About UEFI mode for virtual machines

Unified Extensible Firmware Interface (UEFI), like legacy BIOS, initializes hardware components and operating system image files when a computer starts. UEFI supports more modern features and customization options than BIOS, enabling faster boot times.

It stores all the information about initialization and startup in a file with a **.efi** extension, which is stored on a special partition called EFI System Partition (ESP). The ESP also contains the boot loader programs for the operating system that is installed on the computer.

9.16.5.2. Booting virtual machines in UEFI mode

You can configure a virtual machine to boot in UEFI mode by editing the **VirtualMachine** manifest.

Prerequisites

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

Procedure

1. Edit or create a **VirtualMachine** manifest file. Use the **spec.firmware.bootloader** stanza to configure UEFI mode:

Booting in UEFI mode with secure boot active

```
apiversion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    special: vm-secureboot
  name: vm-secureboot
spec:
  template:
    metadata:
      labels:
        special: vm-secureboot
    spec:
      domain:
        devices:
          disks:
            - disk:
                bus: virtio
                name: containerdisk
          features:
            acpi: {}
            smm:
              enabled: true 1
          firmware:
```



```

bootloader:
  efi:
    secureBoot: true 2
...

```

- 1 OpenShift Virtualization requires System Management Mode (**SMM**) to be enabled for Secure Boot in UEFI mode to occur.
- 2 OpenShift Virtualization supports a VM with or without Secure Boot when using UEFI mode. If Secure Boot is enabled, then UEFI mode is required. However, UEFI mode can be enabled without using Secure Boot.

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

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

9.16.6. 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.

9.16.6.1. Prerequisites

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

9.16.6.2. 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 ❶
      },
      {
        "type": "cnv-tuning" ❷
      }
    ]
  }'
```

- ❶ Optional: The VLAN tag.
- ❷ The **cnv-tuning** plug-in provides support for custom MAC addresses.

**NOTE**

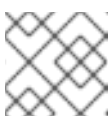
The virtual machine instance will be attached to the bridge **br1** through an access port with the requested VLAN.

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
```

9.16.6.3. Template: Virtual machine configuration file for PXE booting

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  creationTimestamp: null
  labels:
    special: vm-pxe-boot
  name: vm-pxe-boot
spec:
  template:
    metadata:
      labels:
        special: vm-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:00:00:00:00:de
              bootOrder: 1
        machine:
          type: ""
      resources:
        requests:
          memory: 1024M
      networks:
        - name: default
          pod: {}
        - multus:
            networkName: pxe-net-conf
            name: pxe-net
      terminationGracePeriodSeconds: 180
      volumes:
        - name: containerdisk
          containerDisk:
            image: kubevirt/fedora-cloud-container-disk-demo
        - cloudInitNoCloud:
            userData: |
              #cloud-config
```

```

user: fedora
password: fedora
chpasswd: {expire: False}
name: cloudinitdisk
status: {}

```

9.16.6.4. OpenShift Virtualization networking glossary

OpenShift Virtualization provides advanced networking functionality by using custom resources and plug-ins.

The following terms are used throughout OpenShift Virtualization documentation:

Container Network Interface (CNI)

a [Cloud Native Computing Foundation](#) project, focused on container network connectivity. OpenShift Virtualization uses CNI plug-ins to build upon the basic Kubernetes networking functionality.

Multus

a "meta" CNI plug-in that allows multiple CNIs to exist so that a pod or virtual machine can use the interfaces it needs.

Custom resource definition (CRD)

a [Kubernetes](#) API resource that allows you to define custom resources, or an object defined by using the CRD API resource.

Network attachment definition (NAD)

a CRD introduced by the Multus project that allows you to attach pods, virtual machines, and virtual machine instances to one or more networks.

Node network configuration policy (NNCP)

a description of 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.

Preboot eXecution Environment (PXE)

an interface that enables an administrator to boot a client machine from a server over the network. Network booting allows you to remotely load operating systems and other software onto the client.

9.16.7. Using huge pages with virtual machines

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

9.16.7.1. Prerequisites

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

9.16.7.2. 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.

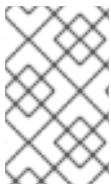
A huge page is a memory page that is larger than 4Ki. On x86_64 architectures, there are two common huge page sizes: 2Mi and 1Gi. Sizes vary on other architectures. To use huge pages, code must be written so that applications are aware of them. Transparent Huge Pages (THP) attempt to automate the management 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.

9.16.7.3. 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
```

9.16.8. Enabling dedicated resources for virtual machines

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

9.16.8.1. About dedicated resources

When you enable dedicated resources for your virtual machine, your virtual machine's workload is scheduled on CPUs that will not be used by other processes. By using dedicated resources, you can improve the performance of the virtual machine and the accuracy of latency predictions.

9.16.8.2. 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.

9.16.8.3. Enabling dedicated resources for a virtual machine

You enable dedicated resources for a virtual machine in the **Details** tab. Virtual machines that were created from a Red Hat template can be configured with dedicated resources.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. On the **Scheduling** tab, click the pencil icon beside **Dedicated Resources**.
4. Select **Schedule this workload with dedicated resources (guaranteed policy)**
5. Click **Save**.

9.16.9. Scheduling virtual machines

You can schedule a virtual machine (VM) on a node by ensuring that the VM's CPU model and policy attribute are matched for compatibility with the CPU models and policy attributes supported by the node.

9.16.9.1. Policy attributes

You can schedule a virtual machine (VM) by specifying a policy attribute and a CPU feature that is matched for compatibility when the VM is scheduled on a node. A policy attribute specified for a VM determines how that VM is scheduled on a node.

Policy attribute	Description
force	The VM is forced to be scheduled on a node. This is true even if the host CPU does not support the VM's CPU.
require	Default policy that applies to a VM if the VM is not configured with a specific CPU model and feature specification. If a node is not configured to support CPU node discovery with this default policy attribute or any one of the other policy attributes, VMs are not scheduled on that node. Either the host CPU must support the VM's CPU or the hypervisor must be able to emulate the supported CPU model.
optional	The VM is added to a node if that VM is supported by the host's physical machine CPU.
disable	The VM cannot be scheduled with CPU node discovery.
forbid	The VM is not scheduled even if the feature is supported by the host CPU and CPU node discovery is enabled.

9.16.9.2. Setting a policy attribute and CPU feature

You can set a policy attribute and CPU feature for each virtual machine (VM) to ensure that it is scheduled on a node according to policy and feature. The CPU feature that you set is verified to ensure that it is supported by the host CPU or emulated by the hypervisor.

Procedure

- Edit the **domain** spec of your VM configuration file. The following example sets the CPU feature and the **require** policy for a virtual machine (VM):

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

1 Name of the CPU feature for the VM.

2 Policy attribute for the VM.

9.16.9.3. Scheduling virtual machines with the supported CPU model

You can configure a CPU model for a virtual machine (VM) to schedule it on a node where its CPU model is supported.

Procedure

- Edit the **domain** spec of your virtual machine configuration file. The following example shows a specific CPU model defined for a VM:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          model: Conroe 1
```

- 1 CPU model for the VM.

9.16.9.4. Scheduling virtual machines with the host model

When the CPU model for a virtual machine (VM) is set to **host-model**, the VM inherits the CPU model of the node where it is scheduled.

Procedure

- Edit the **domain** spec of your VM configuration file. The following example shows **host-model** being specified for the virtual machine:

```
apiVersion: kubevirt/v1alpha3
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          model: host-model 1
```

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

9.16.10. 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).

9.16.10.1. 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.

9.16.10.1.1. 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
```

9.16.10.1.2. 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

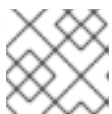
1. Run the **lspci** command to obtain the **vendor-ID** and the **device-ID** for the PCI device.

```
$ lspci -nnv | grep -i nvidia
```

Example output

```
02:01.0 3D controller [0302]: NVIDIA Corporation GV100GL [Tesla V100 PCIe 32GB]
[10de:1eb8] (rev a1)
```

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



NOTE

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

Example

```
variant: openshift
version: 4.11.0
metadata:
  name: 100-worker-vfiopci
  labels:
    machineconfiguration.openshift.io/role: worker 1
storage:
  files:
    - path: /etc/modprobe.d/vfio.conf
      mode: 0644
      overwrite: true
      contents:
```

```

inline: |
  options vfio-pci ids=10de:1eb8 2
- path: /etc/modules-load.d/vfio-pci.conf 3
mode: 0644
overwrite: true
contents:
  inline: vfio-pci

```

- 1 Applies the new kernel argument only to worker nodes.
 - 2 Specify the previously determined **vendor-ID** value (**10de**) and the **device-ID** value (**1eb8**) to bind a single device to the VFIO driver. You can add a list of multiple devices with their vendor and device information.
 - 3 The file that loads the vfio-pci kernel module on the worker nodes.
3. Use Butane to generate a **MachineConfig** object file, **100-worker-vfiopci.yaml**, containing the configuration to be delivered to the worker nodes:

```
$ butane 100-worker-vfiopci.bu -o 100-worker-vfiopci.yaml
```

4. Apply the **MachineConfig** object to the worker nodes:

```
$ oc apply -f 100-worker-vfiopci.yaml
```

5. Verify that the **MachineConfig** object was added.

```
$ oc get MachineConfig
```

Example output

NAME	GENERATEDBYCONTROLLER	IGNITIONVERSION	AGE
00-master	d3da910bfa9f4b599af4ed7f5ac270d55950a3a1	3.2.0	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
```

9.16.10.1.3. 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: ❶
  pciHostDevices: ❷
    - pciDeviceSelector: "10DE:1DB6" ❸
      resourceName: "nvidia.com/GV100GL_Tesla_V100" ❹
    - pciDeviceSelector: "10DE:1EB8"
      resourceName: "nvidia.com/TU104GL_Tesla_T4"
    - pciDeviceSelector: "8086:6F54"
      resourceName: "intel.com/qat"
      externalResourceProvider: true ❺
  ...
```

- ❶ The host devices that are permitted to be used in the cluster.
- ❷ The list of PCI devices available on the node.
- ❸ The **vendor-ID** and the **device-ID** required to identify the PCI device.
- ❹ The name of a PCI host device.
- ❺ Optional: Setting this field to **true** indicates that the resource is provided by an external device plug-in. OpenShift Virtualization allows the usage of this device in the cluster but leaves the allocation and monitoring to an external device plug-in.

**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
```

9.16.10.1.4. 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

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

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

2. Remove the PCI device information from the **spec.permittedHostDevices.pciHostDevices** array by deleting the **pciDeviceSelector**, **resourceName** and **externalResourceProvider** (if applicable) fields for the appropriate device. In this example, the **intel.com/qat** resource has been deleted.

Example configuration file

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  permittedHostDevices:
    pciHostDevices:
      - pciDeviceSelector: "10DE:1DB6"
        resourceName: "nvidia.com/GV100GL_Tesla_V100"
      - pciDeviceSelector: "10DE:1EB8"
        resourceName: "nvidia.com/TU104GL_Tesla_T4"
    ...
```

3. Save your changes and exit the editor.

Verification

- Verify that the PCI host device was removed from the node by running the following command. The example output shows that there are zero devices associated with the **intel.com/qat** resource name.

```
$ oc describe node <node_name>
```

Example output

```
Capacity:
  cpu: 64
  devices.kubevirt.io/kvm: 110
  devices.kubevirt.io/tun: 110
  devices.kubevirt.io/vhost-net: 110
  ephemeral-storage: 915128Mi
  hugepages-1Gi: 0
  hugepages-2Mi: 0
  memory: 131395264Ki
  nvidia.com/GV100GL_Tesla_V100: 1
  nvidia.com/TU104GL_Tesla_T4: 1
  intel.com/qat: 0
  pods: 250
Allocatable:
  cpu: 63500m
  devices.kubevirt.io/kvm: 110
  devices.kubevirt.io/tun: 110
```

```

devices.kubevirt.io/vhost-net: 110
ephemeral-storage:             863623130526
hugepages-1Gi:                 0
hugepages-2Mi:                 0
memory:                        130244288Ki
nvidia.com/GV100GL_Tesla_V100 1
nvidia.com/TU104GL_Tesla_T4   1
intel.com/qat:                  0
pods:                           250

```

9.16.10.2. 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.

9.16.10.2.1. 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)

```

9.16.10.3. Additional resources

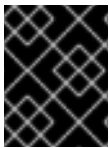
- [Enabling Intel VT-X and AMD-V Virtualization Hardware Extensions in BIOS](#)

- [Managing file permissions](#)
- [Post-installation machine configuration tasks](#)

9.16.11. Configuring vGPU passthrough

Your virtual machines can access a virtual GPU (vGPU) hardware. Assigning a vGPU to your virtual machine allows you do the following:

- Access a fraction of the underlying hardware's GPU to achieve high performance benefits in your virtual machine.
- Streamline resource-intensive I/O operations.



IMPORTANT

vGPU passthrough can only be assigned to devices that are connected to clusters running in a bare metal environment.

9.16.11.1. Assigning vGPU passthrough devices to a virtual machine

Use the OpenShift Container Platform web console to assign vGPU passthrough devices to your virtual machine.

Prerequisites

- The virtual machine must be stopped.

Procedure

1. In the OpenShift Container Platform web console, click **Virtualization** → **VirtualMachines** from the side menu.
2. Select the virtual machine to which you want to assign the device.
3. On the **Details** tab, click **GPU devices**.
If you add a vGPU device as a host device, you cannot access the device with the VNC console.
4. Click **Add GPU device**, enter the **Name** and select the device from the **Device name** list.
5. Click **Save**.
6. Click the **YAML** tab to verify that the new devices have been added to your cluster configuration in the **hostDevices** section.



NOTE

You can add hardware devices to virtual machines created from customized templates or a YAML file. You cannot add devices to pre-supplied boot source templates for specific operating systems, such as Windows 10 or RHEL 7.

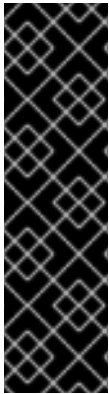
To display resources that are connected to your cluster, click **Compute** → **Hardware Devices** from the side menu.

9.16.11.2. Additional resources

- [Creating virtual machines](#)
- [Creating virtual machine templates](#)

9.16.12. Configuring mediated devices

OpenShift Virtualization automatically creates mediated devices, such as virtual GPUs (vGPUs), if you provide a list of devices in the **HyperConverged** custom resource (CR).



IMPORTANT

Declarative configuration of mediated devices 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/>.

9.16.12.1. Prerequisites

- If your hardware vendor provides drivers, you installed them on the nodes where you want to create mediated devices.
 - If you use NVIDIA cards, you [installed the NVIDIA GRID driver](#).

9.16.12.2. About using virtual GPUs with OpenShift Virtualization

Some graphics processing unit (GPU) cards support the creation of virtual GPUs (vGPUs). OpenShift Virtualization can automatically create vGPUs and other mediated devices if an administrator provides configuration details in the **HyperConverged** custom resource (CR). This automation is especially useful for large clusters.



NOTE

Refer to your hardware vendor's documentation for functionality and support details.

Mediated device

A physical device that is divided into one or more virtual devices. A vGPU is a type of mediated device (mdev); the performance of the physical GPU is divided among the virtual devices. You can assign mediated devices to one or more virtual machines (VMs), but the number of guests must be compatible with your GPU. Some GPUs do not support multiple guests.

9.16.12.2.1. Configuration overview

When configuring mediated devices, an administrator must:

- Create the mediated devices.
- Expose the mediated devices to the cluster.

The **HyperConverged** CR includes APIs that accomplish both tasks:

Creating mediated devices

```
...
spec:
  mediatedDevicesConfiguration:
    mediatedDevicesTypes: <.>
    - <device_type>
    nodeMediatedDeviceTypes: <.>
    - mediatedDevicesTypes: <.>
    - <device_type>
    nodeSelector: <.>
    <node_selector_key>: <node_selector_value>
...
```

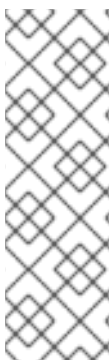
<.> Required: Configures global settings for the cluster. <.> Optional: Overrides the global configuration for a specific node or group of nodes. Must be used with the global **mediatedDevicesTypes** configuration. <.> Required if you use **nodeMediatedDeviceTypes**. Overrides the global **mediatedDevicesTypes** configuration for select nodes. <.> Required if you use **nodeMediatedDeviceTypes**. Must include a **key:value** pair.

Exposing mediated devices to the cluster

```
...
permittedHostDevices:
  mediatedDevices:
    - mdevNameSelector: GRID T4-2Q <.>
      resourceName: nvidia.com/GRID_T4-2Q
...
```

<.> Exposes the mediated devices that map to this value on the host.

+



NOTE

You can see the mediated device types that your device supports by viewing the contents of **/sys/bus/pci/devices/<slot>:<bus>:<domain>.<function>/mdev_supported_types/<type>/name**, substituting the correct values for your system.

For example, the name file for the **nvidia-231** type contains the selector string **GRID T4-2Q**. Using **GRID T4-2Q** as the **mdevNameSelector** value allows nodes to use the **nvidia-231** type.

9.16.12.2.2. How vGPUs are assigned to nodes

For each physical device, OpenShift Virtualization configures:

- A single mdev type.
- The maximum number of instances of the selected mdev type.

The cluster architecture affects how devices are created and assigned to nodes.

Large cluster with multiple cards per node

On nodes with multiple cards that can support similar vGPU types, the relevant device types are created in a round-robin manner. For example:

```
...
mediatedDevicesConfiguration:
  mediatedDevicesTypes:
    - nvidia-222
    - nvidia-228
    - nvidia-105
    - nvidia-108
...
```

In this scenario, each node has two cards, both of which support the following vGPU types:

```
nvidia-105
...
nvidia-108
nvidia-217
nvidia-299
...
```

On each node, OpenShift Virtualization creates:

- 16 vGPUs of type **nvidia-105** on the first card.
- 2 vGPUs of type **nvidia-108** on the second card.

One node has a single card that supports more than one requested vGPU type

OpenShift Virtualization uses the supported type that comes first on the **mediatedDevicesTypes** list.

For example, a node's card supports **nvidia-223** and **nvidia-224**. The following **mediatedDevicesTypes** list is configured:

```
...
mediatedDevicesConfiguration:
  mediatedDevicesTypes:
    - nvidia-22
    - nvidia-223
    - nvidia-224
...
```

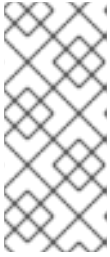
In this example, OpenShift Virtualization uses the **nvidia-223** type.

9.16.12.2.3. About changing and removing mediated devices

OpenShift Virtualization updates the cluster's mediated device configuration if:

- You edit the **HyperConverged** CR and change the contents of the **mediatedDevicesTypes** stanza.
- You change the node labels that match the **nodeMediatedDeviceTypes** node selector.

- You remove the device information from the **spec.mediatedDevicesConfiguration** and **spec.permittedHostDevices** stanzas of the **HyperConverged** CR.



NOTE

If you remove the device information from the **spec.permittedHostDevices** stanza without also removing it from the **spec.mediatedDevicesConfiguration** stanza, you cannot create a new mediated device type on the same node. To properly remove mediated devices, remove the device information from both stanzas.

Depending on the specific changes, these actions cause OpenShift Virtualization to reconfigure mediated devices or remove them from the cluster nodes.

9.16.12.3. Preparing hosts for mediated devices

You must enable the IOMMU (Input-Output Memory Management Unit) driver before you can configure mediated devices.

9.16.12.3.1. 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 ❶
  name: 100-worker-iommu ❷
spec:
  config:
    ignition:
      version: 3.2.0
  kernelArguments:
    - intel_iommu=on ❸
  ...
```

- ❶ 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
```

9.16.12.4. Adding and removing mediated devices

9.16.12.4.1. Creating and exposing mediated devices

You can expose and create mediated devices such as virtual GPUs (vGPUs) by editing the **HyperConverged** custom resource (CR).

Prerequisites

- You enabled the IOMMU (Input-Output Memory Management Unit) driver.

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 mediated device information to the **HyperConverged** CR **spec**, ensuring that you include the **mediatedDevicesConfiguration** and **permittedHostDevices** stanzas. For example:

Example configuration file

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  mediatedDevicesConfiguration: <.>
    mediatedDevicesTypes: <.>
    - nvidia-231
    nodeMediatedDeviceTypes: <.>
    - mediatedDevicesTypes: <.>
    - nvidia-233
    nodeSelector:
      kubernetes.io/hostname: node-11.redhat.com
  permittedHostDevices: <.>
```

```
mediatedDevices:
- mdevNameSelector: GRID T4-2Q
  resourceName: nvidia.com/GRID_T4-2Q
- mdevNameSelector: GRID T4-8Q
  resourceName: nvidia.com/GRID_T4-8Q
...
```

<.> Creates mediated devices. <.> Required: Global **mediatedDevicesTypes** configuration. <.> Optional: Overrides the global configuration for specific nodes. <.> Required if you use **nodeMediatedDeviceTypes**. <.> Exposes mediated devices to the cluster.

3. Save your changes and exit the editor.

Verification

- You can verify that a device was added to a specific node by running the following command:

```
$ oc describe node <node_name>
```

9.16.12.4.2. Removing mediated devices from the cluster using the CLI

To remove a mediated device from the cluster, delete the information for that device from 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. Remove the device information from the **spec.mediatedDevicesConfiguration** and **spec.permittedHostDevices** stanzas of the **HyperConverged** CR. Removing both entries ensures that you can later create a new mediated device type on the same node. For example:

Example configuration file

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  mediatedDevicesConfiguration:
    mediatedDevicesTypes: ❶
    - nvidia-231
  permittedHostDevices:
    mediatedDevices: ❷
    - mdevNameSelector: GRID T4-2Q
      resourceName: nvidia.com/GRID_T4-2Q
```

- ❶ To remove the **nvidia-231** device type, delete it from the **mediatedDevicesTypes** array.
- ❷ To remove the **GRID T4-2Q** device, delete the **mdevNameSelector** field and its corresponding **resourceName** field.

3. Save your changes and exit the editor.

9.16.12.5. Assigning a mediated device to a virtual machine

Assign mediated devices such as virtual GPUs (vGPUs) to virtual machines.

Prerequisites

- The mediated device is configured in the **HyperConverged** custom resource.

Procedure

- Assign the mediated device to a virtual machine (VM) by editing the **spec.domain.devices.gpus** stanza of the **VirtualMachine** manifest:

Example virtual machine manifest

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  domain:
    devices:
      gpus:
        - deviceName: nvidia.com/TU104GL_Tesla_T4 1
          name: gpu1 2
        - deviceName: nvidia.com/GRID_T4-1Q
          name: gpu2
```

- 1** The resource name associated with the mediated device.
- 2** A name to identify the device on the VM.

Verification

- To verify that the device is available from the virtual machine, run the following command, substituting **<device_name>** with the **deviceName** value from the **VirtualMachine** manifest:

```
$ lspci -nnk | grep <device_name>
```

9.16.12.6. Additional resources

- [Enabling Intel VT-X and AMD-V Virtualization Hardware Extensions in BIOS](#)

9.16.13. Configuring a watchdog

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

9.16.13.1. Prerequisites

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

9.16.13.2. Defining a watchdog device

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

Table 9.5. 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
  ...
```

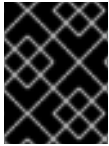
- 1 Specify the **watchdog** action (**poweroff**, **reset**, or **shutdown**).

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
```

9.16.13.3. 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
```

9.16.13.4. Additional resources

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

9.16.14. Automatic importing and updating of pre-defined boot sources

You can use boot sources that are *system-defined* and included with OpenShift Virtualization or *user-defined*, which you create. System-defined boot source imports and updates are controlled by the product feature gate. You can enable, disable, or re-enable updates using the feature gate. User-defined boot sources are not controlled by the product feature gate and must be individually managed to opt in or opt out of automatic imports and updates.



IMPORTANT

As of version 4.10, OpenShift Virtualization automatically imports and updates boot sources, unless you manually opt out or do not set a default storage class.

If you upgrade to version 4.10, you must manually enable automatic imports and updates for boot sources from version 4.9 or earlier.

9.16.14.1. Enabling automatic boot source updates

If you have boot sources from OpenShift Virtualization 4.9 or earlier, you must manually turn on automatic updates for these boot sources. All boot sources in OpenShift Virtualization 4.10 and later are automatically updated by default.

To enable automatic boot source imports and updates, set the **cdi.kubevirt.io/dataImportCron** field to **true** for each boot source you want to update automatically.

Procedure

- To turn on automatic updates for a boot source, use the following command to apply the **dataImportCron** label to the data source:

```
$ oc label --overwrite DataSource rhel8 -n openshift-virtualization-os-images
cdi.kubevirt.io/dataImportCron=true 1
```

- 1 Specifying **true** turns on automatic updates for the **rhel8** boot source.

9.16.14.2. Disabling automatic boot source updates

Disabling automatic boot source imports and updates can be helpful to reduce the number of logs in disconnected environments or to reduce resource usage.

To disable automatic boot source imports and updates, set the **spec.featureGates.enableCommonBootImageImport** field in the **HyperConverged** custom resource (CR) to **false**.



NOTE

User-defined boot sources are not affected by this setting.

Procedure

- Use the following command to disable automatic boot source updates:

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

9.16.14.3. Re-enabling automatic boot source updates

If you have previously disabled automatic boot source updates, you must manually re-enable the feature. Set the **spec.featureGates.enableCommonBootImageImport** field in the **HyperConverged** custom resource (CR) to **true**.

Procedure

- Use the following command to re-enable automatic updates:

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

9.16.14.4. Configuring a storage class for user-defined boot source updates

You can configure a storage class that allows automatic importing and updating for user-defined boot sources.

Procedure

1. Define a new **storageClassName** by editing the **HyperConverged** custom resource (CR).

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      name: rhel8-image-cron
    spec:
      template:
        spec:
          storageClassName: <appropriate_class_name>
  ...
```

2. Set the new default storage class by running the following commands:

```
$ oc patch storageclass <current_default_storage_class> -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

```
$ oc patch storageclass <appropriate_storage_class> -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

9.16.14.5. Enabling automatic updates for user-defined boot sources

OpenShift Virtualization automatically updates system-defined boot sources by default, but does not automatically update user-defined boot sources. You must manually enable automatic imports and updates on a user-defined boot sources by editing the **HyperConverged** custom resource (CR).

Procedure

1. Use the following command to open the **HyperConverged** CR for editing:

```
$ oc edit -n openshift-cnv HyperConverged
```

2. Edit the **HyperConverged** CR, adding the appropriate template and boot source in the **dataImportCronTemplates** section. For example:

Example in CentOS 7

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      name: centos7-image-cron
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true" ❶
      spec:
        schedule: "0 */12 * * *" ❷
        template:
          spec:
            source:
              registry: ❸
              url: docker://quay.io/containerdisks/centos:7-2009
            storage:
              resources:
                requests:
                  storage: 10Gi
            managedDataSource: centos7 ❹
          retentionPolicy: "None" ❺
```

- ❶ This annotation is required for storage classes with **volumeBindingMode** set to **WaitForFirstConsumer**.
- ❷ Schedule for the job specified in cron format.
- ❸ Use to create a data volume from a registry source. Use the default **pod pullMethod** and not **node pullMethod**, which is based on the **node** docker cache. The **node** docker cache is useful when a registry image is available via **Container.Image**, but the CDI importer is not authorized to access it.
- ❹ For the custom image to be detected as an available boot source, the name of the image's **managedDataSource** must match the name of the template's **DataSource**, which is found under **spec.dataVolumeTemplates.spec.sourceRef.name** in the VM template YAML file.
- ❺ Use **All** to retain data volumes and data sources when the cron job is deleted. Use **None** to delete data volumes and data sources when the cron job is deleted.

9.16.14.6. Disabling an automatic update for a system-defined or user-defined boot source

You can disable automatic imports and updates for a user-defined boot source and for a system-defined boot source.

Because system-defined boot sources are not listed by default in the **spec.dataImportCronTemplates** of the **HyperConverged** custom resource (CR), you must add the boot source and disable auto imports and updates.

Procedure

- To disable automatic imports and updates for a user-defined boot source, remove the boot source from the **spec.dataImportCronTemplates** field in the custom resource list.
- To disable automatic imports and updates for a system-defined boot source:
 - Edit the **HyperConverged** CR and add the boot source to **spec.dataImportCronTemplates**.
 - Disable automatic imports and updates by setting the **dataimportcrontemplate.kubevirt.io/enable** annotation to **false**. For example:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
    - metadata:
        annotations:
          dataimportcrontemplate.kubevirt.io/enable: false
        name: rhel8-image-cron
  ...
```

9.16.14.7. Verifying the status of a boot source

You can verify whether a boot source is system-defined or user-defined.

The **status** section of each boot source listed in the **status.dataImportChronTemplates** field of the **HyperConverged** CR indicates the type of boot source. For example, **commonTemplate: true** indicates a system-defined (**commonTemplate**) boot source and **status: {}** indicates a user-defined boot source.

Procedure

1. Use the **oc get** command to list the **dataImportChronTemplates** in the **HyperConverged** CR.
2. Verify the status of the boot source.

Example output

```
...
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
...
spec:
  ...
  status: 1
  ...
  dataImportCronTemplates: 2
    - metadata:
        annotations:
          cdi.kubevirt.io/storage.bind.immediate.requested: "true"
        name: centos-7-image-cron
```

```

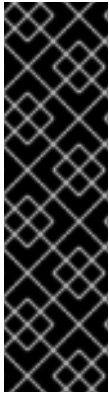
spec:
  garbageCollect: Outdated
  managedDataSource: centos7
  schedule: 55 8/12 * * *
  template:
    metadata: {}
    spec:
      source:
        registry:
          url: docker://quay.io/containerdisks/centos:7-2009
        storage:
          resources:
            requests:
              storage: 30Gi
      status: {}
    status:
      commonTemplate: true ❸
  ...
- metadata:
  annotations:
    cdi.kubevirt.io/storage.bind.immediate.requested: "true"
  name: user-defined-dic
  spec:
    garbageCollect: Outdated
    managedDataSource: user-defined-centos-stream8
    schedule: 55 8/12 * * *
    template:
      metadata: {}
      spec:
        source:
          registry:
            pullMethod: node
            url: docker://quay.io/containerdisks/centos-stream:8
          storage:
            resources:
              requests:
                storage: 30Gi
        status: {}
      status: {} ❹
  ...

```

- ❶ The **status** field for the **HyperConverged** CR.
- ❷ The **dataImportCronTemplates** field, which lists all defined boot sources.
- ❸ Indicates a system-defined boot source.
- ❹ Indicates a user-defined boot source.

9.16.15. Enabling descheduler evictions on virtual machines

You can use the descheduler to evict pods so that the pods can be rescheduled onto more appropriate nodes. If the pod is a virtual machine, the pod eviction causes the virtual machine to be live migrated to another node.



IMPORTANT

Descheduler eviction for virtual machines 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/>.

9.16.15.1. Descheduler profiles

Use the Technology Preview **DevPreviewLongLifecycle** profile to enable the descheduler on a virtual machine. This is the only descheduler profile currently available for OpenShift Virtualization. To ensure proper scheduling, create VMs with CPU and memory requests for the expected load.

DevPreviewLongLifecycle

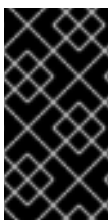
This profile balances resource usage between nodes and enables the following strategies:

- **RemovePodsHavingTooManyRestarts:** removes pods whose containers have been restarted too many times and pods where the sum of restarts over all containers (including Init Containers) is more than 100. Restarting the VM guest operating system does not increase this count.
- **LowNodeUtilization:** evicts pods from overutilized nodes when there are any underutilized nodes. The destination node for the evicted pod will be determined by the scheduler.
 - A node is considered underutilized if its usage is below 20% for all thresholds (CPU, memory, and number of pods).
 - A node is considered overutilized if its usage is above 50% for any of the thresholds (CPU, memory, and number of pods).

9.16.15.2. Installing the descheduler

The descheduler is not available by default. To enable the descheduler, you must install the Kube Descheduler Operator from OperatorHub and enable one or more descheduler profiles.

By default, the descheduler runs in predictive mode, which means that it only simulates pod evictions. You must change the mode to automatic for the descheduler to perform the pod evictions.



IMPORTANT

If you have enabled hosted control planes in your cluster, set a custom priority threshold to lower the chance that pods in the hosted control plane namespaces are evicted. Set the priority threshold class name to **hypershift-control-plane**, because it has the lowest priority value (**100000000**) of the hosted control plane priority classes.

Prerequisites

- Cluster administrator privileges.
- Access to the OpenShift Container Platform web console.

Procedure

1. Log in to the OpenShift Container Platform web console.
2. Create the required namespace for the Kube Descheduler Operator.
 - a. Navigate to **Administration** → **Namespaces** and click **Create Namespace**.
 - b. Enter **openshift-kube-descheduler-operator** in the **Name** field, enter **openshift.io/cluster-monitoring=true** in the **Labels** field to enable descheduler metrics, and click **Create**.
3. Install the Kube Descheduler Operator.
 - a. Navigate to **Operators** → **OperatorHub**.
 - b. Type **Kube Descheduler Operator** into the filter box.
 - c. Select the **Kube Descheduler Operator** and click **Install**.
 - d. On the **Install Operator** page, select **A specific namespace on the cluster** Select **openshift-kube-descheduler-operator** from the drop-down menu.
 - e. Adjust the values for the **Update Channel** and **Approval Strategy** to the desired values.
 - f. Click **Install**.
4. Create a descheduler instance.
 - a. From the **Operators** → **Installed Operators** page, click the **Kube Descheduler Operator**.
 - b. Select the **Kube Descheduler** tab and click **Create KubeDescheduler**.
 - c. Edit the settings as necessary.
 - i. To evict pods instead of simulating the evictions, change the **Mode** field to **Automatic**.
 - ii. Expand the **Profiles** section and select **DevPreviewLongLifecycle**. The **AffinityAndTaints** profile is enabled by default.



IMPORTANT

The only profile currently available for OpenShift Virtualization is **DevPreviewLongLifecycle**.

You can also configure the profiles and settings for the descheduler later using the OpenShift CLI (**oc**).

9.16.15.3. Enabling descheduler evictions on a virtual machine (VM)

After the descheduler is installed, you can enable descheduler evictions on your VM by adding an annotation to the **VirtualMachine** custom resource (CR).

Prerequisites

- Install the descheduler in the OpenShift Container Platform web console or OpenShift CLI (**oc**).

- Ensure that the VM is not running.

Procedure

1. Before starting the VM, add the **descheduler.alpha.kubernetes.io/evict** annotation to the **VirtualMachine** CR:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    metadata:
      annotations:
        descheduler.alpha.kubernetes.io/evict: "true"
```

2. If you did not already set the **DevPreviewLongLifecycle** profile in the web console during installation, specify the **DevPreviewLongLifecycle** in the **spec.profile** section of the **KubeDescheduler** object:

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
  name: cluster
  namespace: openshift-kube-descheduler-operator
spec:
  deschedulingIntervalSeconds: 3600
  profiles:
    - DevPreviewLongLifecycle
```

The descheduler is now enabled on the VM.

9.16.15.4. Additional resources

- [Evicting pods using the descheduler](#)

9.17. IMPORTING VIRTUAL MACHINES

9.17.1. TLS certificates for data volume imports

9.17.1.1. 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>
```

9.17.1.2. Example: Config map created from a TLS certificate

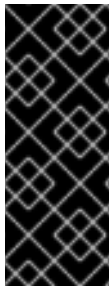
The following example is of a config map created from **ca.pem** TLS certificate.

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

9.17.2. 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.

9.17.2.1. 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.

9.17.2.2. 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)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ 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](#).

9.17.2.3. 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.

9.17.2.4. 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 ❶
  labels:
    app: containerized-data-importer
type: Opaque
data:
  accessKeyId: "" ❷
  secretKey: "" ❸
```

- ❶ Specify the name of the **Secret**.
- ❷ Specify the Base64-encoded key ID or user name.
- ❸ 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 ❶
spec:
  dataVolumeTemplates:
    - metadata:
        creationTimestamp: null
        name: fedora-dv ❷
      spec:
        storage:
          resources:
            requests:
              storage: 10Gi
```

```

    storageClassName: local
  source:
    http: ❸
    url: "https://mirror.arizona.edu/fedora/linux/releases/35/Cloud/x86_64/images/Fedora-
Cloud-Base-35-1.2.x86_64.qcow2" ❹
    secretRef: endpoint-secret ❺
    certConfigMap: "" ❻
  status: {}
  running: true
  template:
    metadata:
      creationTimestamp: null
      labels:
        kubevirt.io/vm: vm-fedora-datavolume
    spec:
      domain:
        devices:
          disks:
            - disk:
                bus: virtio
                name: datavolumedisk1
          machine:
            type: ""
      resources:
        requests:
          memory: 1.5Gi
      terminationGracePeriodSeconds: 180
      volumes:
        - dataVolume:
            name: fedora-dv
            name: datavolumedisk1
  status: {}

```

- ❶ Specify the name of the virtual machine.
- ❷ Specify the name of the data volume.
- ❸ Specify **http** for an HTTP or HTTPS endpoint. Specify **registry** for a container disk image imported from a registry.
- ❹ Specify the URL or registry endpoint of the virtual machine image you want to import. This example references a virtual machine image at an HTTPS endpoint. An example of a container registry endpoint is **url: "docker://kubevirt/fedora-cloud-container-disk-demo:latest"**.
- ❺ Specify the **Secret** name if you created a **Secret** for the data source.
- ❻ 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:

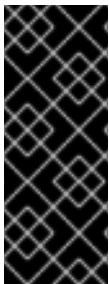
```
$ virtctl console vm-fedora-datavolume
```

9.17.2.5. Additional resources

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

9.17.3. 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.

9.17.3.1. 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.

9.17.3.2. 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.

9.17.3.3. 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.

9.17.3.4. 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> 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.

9.17.3.5. 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 Specify the **Secret** name 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.

9.17.3.6. 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)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ 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](#).

9.17.3.7. Additional resources

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

9.18. CLONING VIRTUAL MACHINES

9.18.1. 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.

9.18.1.1. Prerequisites

- Only a user with the **cluster-admin** role can create cluster roles.

9.18.1.2. 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.

9.18.1.3. 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.

9.18.2. 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](#).

9.18.2.1. Prerequisites

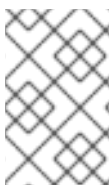
- Users need [additional permissions](#) to clone the PVC of a virtual machine disk into another namespace.

9.18.2.2. 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.

9.18.2.3. 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.

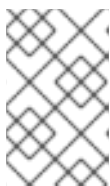
For example:

```
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.

9.18.2.4. 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"

```

9.18.2.5. 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)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

☐ Unsupported operation

* Requires scratch space

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

9.18.3. 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](#).

9.18.3.1. Prerequisites

- Users need [additional permissions](#) to clone the PVC of a virtual machine disk into another namespace.

9.18.3.2. 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.

9.18.3.3. 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 ❶
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"
```

❶ The virtual machine to create.

3. Create the virtual machine with the PVC-cloned data volume:

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

9.18.3.4. 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: 180
        volumes:
        - dataVolume:
            name: example-dv
            name: example-dv-disk
```

1 The **HTTP** source of the image you want to import, if applicable.

9.18.3.5. 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* ☐ GZ ☐ XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* ☐ GZ ☐ XZ	✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

☐ Unsupported operation

* Requires scratch space

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

9.18.4. 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](#).

9.18.4.1. Prerequisites

- Users need [additional permissions](#) to clone the PVC of a virtual machine disk into another namespace.

9.18.4.2. 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.

9.18.4.3. 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.

9.18.4.4. 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.

9.18.4.5. 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.

9.18.4.6. 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)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

☐ Unsupported operation

* Requires scratch space

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

9.19. VIRTUAL MACHINE NETWORKING

9.19.1. 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

9.19.1.1. 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

- Edit the **interfaces** spec of your virtual machine configuration file:

```
kind: VirtualMachine
spec:
  domain:
    devices:
      interfaces:
        - name: default
```

```

masquerade: {} ❶
ports: ❷
  - port: 80
networks:
  - name: default
    pod: {}

```

- ❶ Connect using masquerade mode.
- ❷ 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
```

9.19.1.2. Configuring masquerade mode with dual-stack (IPv4 and IPv6)

You can configure a new virtual machine (VM) to use both IPv6 and IPv4 on the default pod network by using cloud-init.

The **Network.pod.vmlIPv6NetworkCIDR** field in the virtual machine instance configuration determines the static IPv6 address of the VM and the gateway IP address. These are used by the virt-launcher pod to route IPv6 traffic to the virtual machine and are not used externally. The **Network.pod.vmlIPv6NetworkCIDR** field specifies an IPv6 address block in Classless Inter-Domain Routing (CIDR) notation. The default value is **fd10:0:2::2/120**. You can edit this value based on your network requirements.

When the virtual machine is running, incoming and outgoing traffic for the virtual machine is routed to both the IPv4 address and the unique IPv6 address of the virt-launcher pod. The virt-launcher pod then routes the IPv4 traffic to the DHCP address of the virtual machine, and the IPv6 traffic to the statically set IPv6 address of the virtual machine.

Prerequisites

- The OpenShift Container Platform cluster must use the OVN-Kubernetes Container Network Interface (CNI) network 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: {} ❶
    ports:
      - port: 80 ❷
  networks:
    - name: default
      pod: {}
  volumes:
    - cloudInitNoCloud:
        networkData: |
          version: 2
          ethernets:
            eth0:
              dhcp4: true
              addresses: [ fd10:0:2::2/120 ] ❸
              gateway6: fd10:0:2::1 ❹

```

- ❶ Connect using masquerade mode.
- ❷ Allows incoming traffic on port 80 to the virtual machine.
- ❸ The static IPv6 address as determined by the **Network.pod.vmlPv6NetworkCIDR** field in the virtual machine instance configuration. The default value is **fd10:0:2::2/120**.
- ❹ The gateway IP address as determined by the **Network.pod.vmlPv6NetworkCIDR** field in the virtual machine instance configuration. The default value is **fd10:0:2::1**.

2. Create the virtual machine in the namespace:

```
$ oc create -f example-vm-ipv6.yaml
```

Verification

- To verify that IPv6 has been configured, start the virtual machine and view the interface status of the virtual machine instance to ensure it has an IPv6 address:

```
$ oc get vmi <vmi-name> -o jsonpath="{.status.interfaces[*].ipAddresses}"
```

9.19.2. 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.

9.19.2.1. 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.

9.19.2.1.1. 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]**

9.19.2.2. 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 ❶
# ...

```

- ❶ 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 ❶
  namespace: example-namespace ❷
spec:
  externalTrafficPolicy: Cluster ❸
  ports:
    - nodePort: 30000 ❹
      port: 27017
      protocol: TCP
      targetPort: 22 ❺
  selector:
    special: key ❻
  type: NodePort ❼

```

- ❶ The name of the **Service** object.
- ❷ The namespace where the **Service** object resides. This must match the **metadata.namespace** field of the **VirtualMachine** manifest.
- ❸ 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.
- ❹ 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.
- ❺ 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**.
- ❻ 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.

9.19.2.3. Additional resources

- [Configuring ingress cluster traffic using a NodePort](#)

- [Configuring ingress cluster traffic using a load balancer](#)

9.19.3. 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.

9.19.3.1. Connecting to the network through the network attachment definition

9.19.3.1.1. 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.

9.19.3.2. Creating a Linux bridge network attachment definition



WARNING

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

9.19.3.2.1. Creating a Linux bridge network attachment definition in the web console

Network administrators can create network attachment definitions to provide 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.

9.19.3.2.2. 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.

Prerequisites

- The node must support nftables and the **nft** binary must be deployed to enable MAC spoof check.

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> ❶
  annotations:
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/<bridge-interface> ❷
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "<bridge-network>", ❸
    "type": "cnv-bridge", ❹
    "bridge": "<bridge-interface>", ❺
    "macspoofchk": true, ❻
    "vlan": 1 ❼
  }'
```

- ❶ The name for the **NetworkAttachmentDefinition** object.
- ❷ Optional: Annotation key-value pair for node selection, where **bridge-interface** must match 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.
- ❸ The name for the configuration. It is recommended to match the configuration name to the **name** value of the network attachment definition.
- ❹ The actual name of the Container Network Interface (CNI) plug-in that provides the network for this network attachment definition. Do not change this field unless you want to use a different CNI.
- ❺ 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>
```

9.19.3.3. Configuring the virtual machine for a Linux bridge network**9.19.3.3.1. Creating a NIC for a virtual machine in the web console**

Create and attach additional NICs to a virtual machine from the web console.

Prerequisites

- A network attachment definition must be available.

Procedure

1. In the correct project in the OpenShift Container Platform console, click **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Network Interfaces** tab to view the NICs already attached to the virtual machine.
4. Click **Add Network Interface** to create a new slot in the list.
5. Select a network attachment definition from the **Network** list for the additional network.
6. Fill in the **Name**, **Model**, **Type**, and **MAC Address** for the new NIC.
7. Click **Save** to save and attach the NIC to the virtual machine.

9.19.3.3.2. 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. Select the binding method suitable for the network interface: <ul style="list-style-type: none"> • Default pod network: masquerade • Linux bridge network: bridge • SR-IOV network: SR-IOV
MAC Address	MAC address for the network interface controller. If a MAC address is not specified, one is assigned automatically.

9.19.3.3.3. 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) plug-in 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.

9.19.4. 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.

9.19.4.1. 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.

9.19.5. 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.

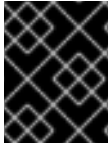
9.19.5.1. Prerequisites

- You must have [installed the SR-IOV Operator](#).
- You must have [configured the SR-IOV Operator](#).
- You must have [enabled global SR-IOV and VT-d settings in the BIOS for the host](#).

9.19.5.2. 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.

9.19.5.2.1. 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 ❶
  namespace: openshift-sriov-network-operator
  ownerReferences:
    - apiVersion: sriovnetwork.openshift.io/v1
      blockOwnerDeletion: true
      controller: true
      kind: SriovNetworkNodePolicy
      name: default
spec:
  dpConfigVersion: "39824"
status:
  interfaces: ❷
    - 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.

9.19.5.3. 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> ❶
  namespace: openshift-sriov-network-operator ❷
spec:
  resourceName: <sriov_resource_name> ❸
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" ❹
  priority: <priority> ❺
  mtu: <mtu> ❻
  numVfs: <num> ❼
  nicSelector: ❽
    vendor: "<vendor_code>" ❾
    deviceID: "<device_id>" ❿
    pfNames: ["<pf_name>", ...] 11
    rootDevices: ["<pci_bus_id>", "..."] 12
  deviceType: vfio-pci 13
  isRdma: false 14

```

- ❶ Specify a name for the CR object.
- ❷ Specify the namespace where the SR-IOV Operator is installed.
- ❸ Specify the resource name of the SR-IOV device plug-in. You can create multiple **SriovNetworkNodePolicy** objects for a resource name.
- ❹ 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) plug-in and device plug-in are deployed only on selected nodes.
- ❺ 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**.
- ❻ Optional: Specify a value for the maximum transmission unit (MTU) of the virtual function. The maximum MTU value can vary for different NIC models.
- ❼ 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**.
- ❽ 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.
- ❾ Optional: Specify the vendor hex code of the SR-IOV network device. The only allowed values are either **8086** or **15b3**.
- ❿ 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}'
```

9.19.5.4. Next steps

- [Configuring an SR-IOV network attachment for virtual machines](#)

9.19.6. Connecting virtual machines to a service mesh

OpenShift Virtualization is now integrated with OpenShift Service Mesh. You can monitor, visualize, and control traffic between pods that run virtual machine workloads on the default pod network with IPv4.

9.19.6.1. Prerequisites

- You must have [installed the Service Mesh Operator](#) and [deployed the service mesh control plane](#).
- You must have added the namespace where the virtual machine is created to the [service mesh member roll](#).

- You must use the **masquerade** binding method for the default pod network.

9.19.6.2. Configuring a virtual machine for the service mesh

To add a virtual machine (VM) workload to a service mesh, enable automatic sidecar injection in the VM configuration file by setting the **sidecar.istio.io/inject** annotation to **true**. Then expose your VM as a service to view your application in the mesh.

Prerequisites

- To avoid port conflicts, do not use ports used by the Istio sidecar proxy. These include ports 15000, 15001, 15006, 15008, 15020, 15021, and 15090.

Procedure

1. Edit the VM configuration file to add the **sidecar.istio.io/inject: "true"** annotation.

Example configuration file

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm-istio
  name: vm-istio
spec:
  runStrategy: Always
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm-istio
        app: vm-istio 1
      annotations:
        sidecar.istio.io/inject: "true" 2
    spec:
      domain:
        devices:
          interfaces:
            - name: default
              masquerade: {} 3
          disks:
            - disk:
                bus: virtio
                name: containerdisk
            - disk:
                bus: virtio
                name: cloudinitdisk
          resources:
            requests:
              memory: 1024M
          networks:
            - name: default
              pod: {}
      terminationGracePeriodSeconds: 180
      volumes:
```



```
- containerDisk:
  image: registry:5000/kubevirt/fedora-cloud-container-disk-demo:devel
  name: containerdisk
```

- 1 The key/value pair (label) that must be matched to the service selector attribute.
- 2 The annotation to enable automatic sidecar injection.
- 3 The binding method (masquerade mode) for use with the default pod network.

2. Apply the VM configuration:

```
$ oc apply -f <vm_name>.yaml 1
```

- 1 The name of the virtual machine YAML file.

3. Create a **Service** object to expose your VM to the service mesh.

```
apiVersion: v1
kind: Service
metadata:
  name: vm-istio
spec:
  selector:
    app: vm-istio 1
  ports:
    - port: 8080
      name: http
      protocol: TCP
```

- 1 The service selector that determines the set of pods targeted by a service. This attribute corresponds to the **spec.metadata.labels** field in the VM configuration file. In the above example, the **Service** object named **vm-istio** targets TCP port 8080 on any pod with the label **app=vm-istio**.

4. Create the service:

```
$ oc create -f <service_name>.yaml 1
```

- 1 The name of the service YAML file.

9.19.7. 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.

9.19.7.1. Prerequisites

- You must have [configured an SR-IOV device for virtual machines](#).

9.19.7.2. 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 **<spoof_check>** with the spoof check mode of the VF. The allowed values are the strings **"on"** and **"off"**.



IMPORTANT

You must enclose the value you specify in quotes or the CR is rejected by the SR-IOV Network Operator.

- 7 Optional: Replace **<link_state>** with the link state of virtual function (VF). Allowed value are **enable**, **disable** and **auto**.
- 8 Optional: Replace **<max_tx_rate>** with a maximum transmission rate, in Mbps, for the VF.
- 9 Optional: Replace **<min_tx_rate>** with a minimum transmission rate, in Mbps, for the VF. This value should always be less than or equal to Maximum transmission rate.



NOTE

Intel NICs do not support the **minTxRate** parameter. For more information, see [BZ#1772847](#).

- 10 Optional: Replace **<vlan_qos>** with an IEEE 802.1p priority level for the VF. The default value is **0**.
- 11 Optional: Replace **<trust_vf>** with the trust mode of the VF. The allowed values are the strings **"on"** and **"off"**.



IMPORTANT

You must enclose the value you specify in quotes or the CR is rejected by the SR-IOV Network Operator.

- 12 Optional: Replace **<capabilities>** with the capabilities to configure for this network.

2. To create the object, enter the following command. Replace **<name>** with a name for this additional network.

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

3. Optional: To confirm that the **NetworkAttachmentDefinition** object associated with the **SriovNetwork** object that you created in the previous step exists, enter the following command. Replace **<namespace>** with the namespace you specified in the **SriovNetwork** object.

```
$ oc get net-attach-def -n <namespace>
```

9.19.7.3. Next steps

- [Attaching a virtual machine to an SR-IOV network.](#)

9.19.8. 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.

9.19.8.1. Prerequisites

- You must have [configured an SR-IOV device for virtual machines](#).
- You must have [defined an SR-IOV network](#).

9.19.8.2. 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.

9.19.9. 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.

9.19.9.1. Prerequisites

- Install the QEMU guest agent on the virtual machine.

9.19.9.2. 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
```

9.19.9.3. Viewing the IP address of a virtual machine interface in the web console

The IP information is displayed on the **VirtualMachine details** page for the virtual machine.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine name to open the **VirtualMachine details** page.

The information for each attached NIC is displayed under **IP Address** on the **Details** tab.

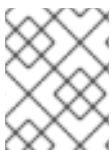
9.19.10. Using a MAC address pool for virtual machines

The *KubeMacPool* component provides a MAC address pool service for virtual machine NICs in a namespace.

9.19.10.1. 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.

9.19.10.2. 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
```

9.19.10.3. 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-
```

9.20. VIRTUAL MACHINE DISKS

9.20.1. Storage features

Use the following table to determine feature availability for local and shared persistent storage in OpenShift Virtualization.

9.20.1.1. OpenShift Virtualization storage feature matrix

Table 9.6. 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 Data Foundation: 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

Do not set the **evictionStrategy** field to **LiveMigrate** for these virtual machines.

9.20.2. Configuring local storage for virtual machines

You can configure local storage for virtual machines by using the hostpath provisioner (HPP).

9.20.2.1. About the hostpath provisioner

When you install the OpenShift Virtualization Operator, the Hostpath Provisioner (HPP) Operator is automatically installed. The HPP is a local storage provisioner designed for OpenShift Virtualization that is created by the Hostpath Provisioner Operator. To use the HPP, you must create an HPP custom resource (CR).

**IMPORTANT**

In OpenShift Virtualization 4.10, the HPP Operator configures the Kubernetes CSI driver. The Operator also recognizes the existing (legacy) format of the HPP CR.

The legacy HPP and the Container Storage Interface (CSI) driver are supported in parallel for a number of releases. However, at some point, the legacy HPP will no longer be supported. If you use the HPP, plan to create a storage class for the CSI driver as part of your migration strategy.

If you upgrade to OpenShift Virtualization version 4.10 on an existing cluster, the HPP Operator is upgraded and the system performs the following actions:

- The CSI driver is installed.
- The CSI driver is configured with the contents of your legacy HPP CR.

If you install OpenShift Virtualization version 4.10 on a new cluster, you must perform the following actions:

- Create an HPP CR with a basic storage pool.
- Create a storage class for the CSI driver.

Optional: You can create a storage pool with a PVC template for multiple HPP volumes.

9.20.2.2. Creating a hostpath provisioner with a basic storage pool

You configure a hostpath provisioner (HPP) with a basic storage pool by creating an HPP custom resource (CR) with a **storagePools** stanza. The storage pool specifies the name and path used by the CSI driver.

Prerequisites

- The directories specified in **spec.storagePools.path** must have read/write access.
- The storage pools must not be in the same partition as the operating system. Otherwise, the operating system partition might become filled to capacity, which will impact performance or cause the node to become unstable or unusable.

Procedure

1. Create an **hpp_cr.yaml** file with a **storagePools** stanza as in the following example:

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
  storagePools: ❶
  - name: any_name
    path: "/var/myvolumes" ❷
workload:
  nodeSelector:
    kubernetes.io/os: linux
```

- ❶ The **storagePools** stanza is an array to which you can add multiple entries.
- ❷ Specify the storage pool directories under this node path.

2. Save the file and exit.
3. Create the HPP by running the following command:

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

9.20.2.3. About creating storage classes

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

In order to use the hostpath provisioner (HPP) you must create an associated storage class for the CSI driver with the **storagePools** stanza.



NOTE

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

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

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

You create a storage class custom resource (CR) for the hostpath provisioner (HPP) CSI driver.

Prerequisites

- You must have OpenShift Virtualization 4.10 or later.

Procedure

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

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: hostpath-csi 1
provisioner: kubevirt.io.hostpath-provisioner
reclaimPolicy: Delete 2
volumeBindingMode: WaitForFirstConsumer 3
parameters:
  storagePool: my-storage-pool 4
```

- 1** Assign any meaningful name to the storage class. In this example, **`csi`** is used to specify that the class is using the CSI provisioner instead of the legacy provisioner. Choosing descriptive names for storage classes, based on legacy or CSI driver provisioning, eases implementation of your migration strategy.
- 2** The two possible **`reclaimPolicy`** values are **`Delete`** and **`Retain`**. If you do not specify a value, the default value is **`Delete`**.
- 3** The **`volumeBindingMode`** parameter determines when dynamic provisioning and volume binding occur. Specify **`WaitForFirstConsumer`** to delay the binding and provisioning of a persistent volume (PV) until after a pod that uses the persistent volume claim (PVC) is created. This ensures that the PV meets the pod's scheduling requirements.
- 4** Specify the name of the storage pool defined in the HPP CR.

- Save the file and exit.
- Create the **`StorageClass`** object by running the following command:

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

9.20.2.3.2. Creating a storage class for the legacy hostpath provisioner

You create a storage class for the legacy hostpath provisioner (HPP) by creating a **`StorageClass`** object without the **`storagePool`** parameter.

Procedure

- Create a **`storageclass.yaml`** file to define the storage class:

```
apiVersion: storage.k8s.io/v1
```

```
kind: StorageClass
metadata:
  name: hostpath-provisioner
provisioner: kubevirt.io/hostpath-provisioner
reclaimPolicy: Delete 1
volumeBindingMode: WaitForFirstConsumer 2
```

- 1** The two possible **reclaimPolicy** values are **Delete** and **Retain**. If you do not specify a value, the storage class defaults to **Delete**.
- 2** The **volumeBindingMode** value determines when dynamic provisioning and volume binding occur. Specify the **WaitForFirstConsumer** value to delay the binding and provisioning of a persistent volume until after a pod that uses the persistent volume claim (PVC) is created. This ensures that the PV meets the pod's scheduling requirements.

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

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

Additional resources

- [Storage classes](#)

9.20.2.4. About storage pools created with PVC templates

If you have a single, large persistent volume (PV), you can create a storage pool by defining a PVC template in the hostpath provisioner (HPP) custom resource (CR).

A storage pool created with a PVC template can contain multiple HPP volumes. Splitting a PV into smaller volumes provides greater flexibility for data allocation.

The PVC template is based on the **spec** stanza of the **PersistentVolumeClaim** object:

Example PersistentVolumeClaim object

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: iso-pvc
spec:
  volumeMode: Block 1
  storageClassName: my-storage-class
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

- 1** This value is only required for block volume mode PVs.

You define a storage pool using a **pvcTemplate** specification in the HPP CR. The Operator creates a

PVC from the **pvcTemplate** specification for each node containing the HPP CSI driver. The PVC created from the PVC template consumes the single large PV, allowing the HPP to create smaller dynamic volumes.

You can combine basic storage pools with storage pools created from PVC templates.

9.20.2.4.1. Creating a storage pool with a PVC template

You can create a storage pool for multiple hostpath provisioner (HPP) volumes by specifying a PVC template in the HPP custom resource (CR).

Prerequisites

- The directories specified in **spec.storagePools.path** must have read/write access.
- The storage pools must not be in the same partition as the operating system. Otherwise, the operating system partition might become filled to capacity, which will impact performance or cause the node to become unstable or unusable.

Procedure

1. Create an **hpp_pvc_template_pool.yaml** file for the HPP CR that specifies a persistent volume (PVC) template in the **storagePools** stanza according to the following example:

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
  storagePools: ❶
  - name: my-storage-pool
    path: "/var/myvolumes" ❷
    pvcTemplate:
      volumeMode: Block ❸
      storageClassName: my-storage-class ❹
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 5Gi ❺
  workload:
    nodeSelector:
      kubernetes.io/os: linux
```

- ❶ The **storagePools** stanza is an array that can contain both basic and PVC template storage pools.
- ❷ Specify the storage pool directories under this node path.
- ❸ Optional: The **volumeMode** parameter can be either **Block** or **Filesystem** as long as it matches the provisioned volume format. If no value is specified, the default is **Filesystem**. If the **volumeMode** is **Block**, the mounting pod creates an XFS file system on the block volume before mounting it.

- 4 If the **storageClassName** parameter is omitted, the default storage class is used to create PVCs. If you omit **storageClassName**, ensure that the HPP storage class is not the default
- 5 You can specify statically or dynamically provisioned storage. In either case, ensure the requested storage size is appropriate for the volume you want to virtually divide or the PVC cannot be bound to the large PV. If the storage class you are using uses dynamically provisioned storage, pick an allocation size that matches the size of a typical request.

2. Save the file and exit.
3. Create the HPP with a storage pool by running the following command:

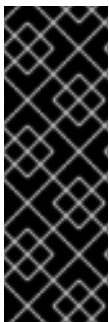
```
$ oc create -f hpp_pvc_template_pool.yaml
```

Additional resources

- [Customizing the storage profile](#)

9.20.3. Creating data volumes

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.

9.20.3.1. 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> ❶
spec:
  source:
    pvc: ❷
    namespace: "<source_namespace>" ❸
    name: "<my_vm_disk>" ❹
  storage: ❺
  resources:
    requests:
      storage: 2Gi ❻
  storageClassName: <storage_class> ❼
```

- ❶ The name of the new data volume.
- ❷ Indicate that the source of the import is an existing persistent volume claim (PVC).
- ❸ The namespace where the source PVC exists.
- ❹ The name of the source PVC.
- ❺ Indicates allocation using the storage API.
- ❻ Specifies the amount of available space that you request for the PVC.
- ❼ Optional: The name of the storage class. If the storage class is not specified, the system default storage class is used.

9.20.3.2. 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:

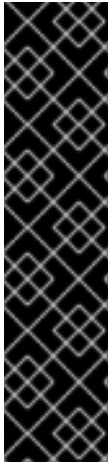
- **accessModes** (**ReadWriteOnce**, **ReadWriteMany**, or **ReadOnlyMany**)
- **volumeMode** (**Filesystem** or **Block**)
- **capacity** of **storage** (**5Gi**, for example)

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> ❶
spec:
  source:
    pvc: ❷
    namespace: "<source_namespace>" ❸
    name: "<my_vm_disk>" ❹
  pvc: ❺
  accessModes: ❻
    - ReadWriteMany
  resources:
    requests:
      storage: 2Gi ❼
  volumeMode: Block ❸
  storageClassName: <storage_class> ❹
```

- ❶ The name of the new data volume.
- ❷ In the **source** section, **pvc** indicates that the source of the import is an existing persistent volume claim (PVC).
- ❸ The namespace where the source PVC exists.
- ❹ The name of the source PVC.
- ❺ Indicates allocation using the PVC API.
- ❻ **accessModes** is required when using the PVC API.
- ❼ Specifies the amount of space you are requesting for your data volume.
- ❸ Specifies that the destination is a block PVC.
- ❹ 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.

9.20.3.3. 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: <unknown_provisioner_class>
# ...
spec: {}
status:
  provisioner: <unknown_provisioner>
  storageClass: <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: <unknown_provisioner_class>
# ...
spec:
  claimPropertySets:
    - accessModes:
      - ReadWriteOnce 1
    volumeMode:
      Filesystem 2
status:
  provisioner: <unknown_provisioner>
  storageClass: <unknown_provisioner_class>

```

- 1** The **accessModes** that you select.
- 2** The **volumeMode** that you select.

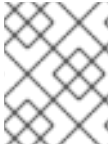
After you save your changes, the selected values appear in the storage profile **status** element.

9.20.3.3.1. Setting a default cloning strategy using a storage profile

You can use storage profiles to set a default cloning method for a storage class, creating a *cloning strategy*. Setting cloning strategies can be helpful, for example, if your storage vendor only supports certain cloning methods. It also allows you to select a method that limits resource usage or maximizes performance.

Cloning strategies can be specified by setting the **cloneStrategy** attribute in a storage profile to one of these values:

- **snapshot** - This method is used by default when snapshots are configured. This cloning strategy uses a temporary volume snapshot to clone the volume. The storage provisioner must support CSI snapshots.
- **copy** - This method uses a source pod and a target pod to copy data from the source volume to the target volume. Host-assisted cloning is the least efficient method of cloning.
- **csi-clone** - This method uses the CSI clone API to efficiently clone an existing volume without using an interim volume snapshot. Unlike **snapshot** or **copy**, which are used by default if no storage profile is defined, CSI volume cloning is only used when you specify it in the **StorageProfile** object for the provisioner's storage class.

**NOTE**

You can also set clone strategies using the CLI without modifying the default **claimPropertySets** in your YAML **spec** section.

Example storage profile

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <provisioner_class>
# ...
spec:
  claimPropertySets:
  - accessModes:
    - ReadWriteOnce ❶
    volumeMode:
      Filesystem ❷
  cloneStrategy:
    csi-clone ❸
status:
  provisioner: <provisioner>
  storageClass: <provisioner_class>
```

- ❶ The **accessModes** that you select.
- ❷ The **volumeMode** that you select.
- ❸ The default cloning method of your choice. In this example, CSI volume cloning is specified.

9.20.3.4. Additional resources

- [About creating storage classes](#)
- [Overriding the default file system overhead value](#)
- [Cloning a data volume using smart cloning](#)

9.20.4. 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.

9.20.4.1. 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.

9.20.4.2. 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
```

9.20.5. 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.

9.20.5.1. 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.

9.20.5.2. 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-cnv 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.

9.20.5.3. Additional resources

- [Resource quotas per project](#)

9.20.6. 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.

9.20.6.1. 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

9.20.7. 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.

9.20.7.1. 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.

9.20.7.2. 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.

9.20.8. Uploading local disk images by using the web console

You can upload a locally stored disk image file by using the web console.

9.20.8.1. 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.

9.20.8.2. 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)	✓ 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)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

☐ Unsupported operation

* Requires scratch space

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

9.20.8.3. 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

- From the side menu of the web console, click **Storage → Persistent Volume Claims**
- Click the **Create Persistent Volume Claim** drop-down list to expand it.
- Click **With Data Upload Form** to open the **Upload Data to Persistent Volume Claim** page.
- 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**.

9.20.8.4. Additional resources

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

9.20.9. 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.

9.20.9.1. Prerequisites

- [Enable](#) 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.

9.20.9.2. 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.

9.20.9.3. 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
```

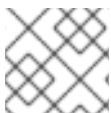
- 1** The name of the data volume.
- 2** The size of the data volume. Ensure that this value is greater than or equal to the size of the disk that you upload.

2. Create the data volume by running the following command:

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

9.20.9.4. 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.

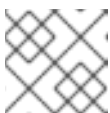


NOTE

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
```

9.20.9.5. 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)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

☐ Unsupported operation

* Requires scratch space

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

9.20.9.6. Additional resources

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

9.20.10. 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.

9.20.10.1. Prerequisites

- [Enable](#) 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.

9.20.10.2. 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.

9.20.10.3. 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.

9.20.10.4. 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.

9.20.10.5. 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
```

9.20.10.6. 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> \ ❶
--size=<datavolume_size> \ ❷
--image-path=</path/to/image> \ ❸
```

- ❶ The name of the data volume.
- ❷ The size of the data volume. For example: **--size=500Mi**, **--size=1G**
- ❸ 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
```

9.20.10.7. 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)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

☐ Unsupported operation

* Requires scratch space

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

9.20.10.8. Additional resources

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

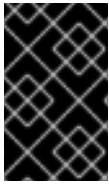
9.20.11. Managing virtual machine snapshots

You can create and delete virtual machine (VM) snapshots for VMs, whether the VMs are powered off (offline) or on (online). You can only restore to a powered off (offline) VM. OpenShift Virtualization supports VM snapshots on the following:

- Red Hat OpenShift Data Foundation

- Any other cloud storage provider with the Container Storage Interface (CSI) driver that supports the Kubernetes Volume Snapshot API

Online snapshots have a default time deadline of five minutes (**5m**) that can be changed, if needed.



IMPORTANT

Online snapshots are supported for virtual machines that have hot-plugged virtual disks. However, hot-plugged disks that are not in the virtual machine specification are not included in the snapshot.



NOTE

To create snapshots of an online (Running state) VM with the highest integrity, install the QEMU guest agent.

The QEMU guest agent takes a consistent snapshot by attempting to quiesce the VM's file system as much as possible, depending on the system workload. This ensures that in-flight I/O is written to the disk before the snapshot is taken. If the guest agent is not present, quiescing is not possible and a best-effort snapshot is taken. The conditions under which the snapshot was taken are reflected in the snapshot indications that are displayed in the web console or CLI.

9.20.11.1. 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.

A VM snapshot is created from a VM that is powered off (Stopped state) or powered on (Running state).

When taking a snapshot of a running VM, the controller checks that the QEMU guest agent is installed and running. If so, it freezes the VM file system before taking the snapshot, and thaws the file system after the snapshot is taken.

The snapshot stores a copy of each Container Storage Interface (CSI) volume attached to the VM and a copy of the VM specification and metadata. Snapshots cannot be changed after creation.

With the 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

9.20.11.1.1. 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.

9.20.11.2. 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.

To check if your virtual machine (VM) has the QEMU guest agent installed and running, verify that **AgentConnected** is listed in the VM spec.



NOTE

To create snapshots of an online (Running state) VM with the highest integrity, install the QEMU guest agent.

The QEMU guest agent takes a consistent snapshot by attempting to quiesce the VM's file system as much as possible, depending on the system workload. This ensures that in-flight I/O is written to the disk before the snapshot is taken. If the guest agent is not present, quiescing is not possible and a best-effort snapshot is taken. The conditions under which the snapshot was taken are reflected in the snapshot indications that are displayed in the web console or CLI.

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
```

9.20.11.3. Installing QEMU guest agent on a Windows virtual machine

For Windows virtual machines, the QEMU guest agent is included in the VirtIO drivers. Install the drivers on an existing or a new Windows installation.

To check if your virtual machine (VM) has the QEMU guest agent installed and running, verify that **AgentConnected** is listed in the VM spec.

**NOTE**

To create snapshots of an online (Running state) VM with the highest integrity, install the QEMU guest agent.

The QEMU guest agent takes a consistent snapshot by attempting to quiesce the VM's file system as much as possible, depending on the system workload. This ensures that in-flight I/O is written to the disk before the snapshot is taken. If the guest agent is not present, quiescing is not possible and a best-effort snapshot is taken. The conditions under which the snapshot was taken are reflected in the snapshot indications that are displayed in the web console or CLI.

9.20.11.3.1. 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.

9.20.11.3.2. 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.

9.20.11.4. Creating a virtual machine snapshot in the web console

You can create a virtual machine (VM) snapshot by using the web console.

**NOTE**

To create snapshots of an online (Running state) VM with the highest integrity, install the QEMU guest agent.

The QEMU guest agent takes a consistent snapshot by attempting to quiesce the VM's file system as much as possible, depending on the system workload. This ensures that in-flight I/O is written to the disk before the snapshot is taken. If the guest agent is not present, quiescing is not possible and a best-effort snapshot is taken. The conditions under which the snapshot was taken are reflected in the snapshot indications that are displayed in the web console or CLI.

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

Procedure

1. Click **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. If the virtual machine is running, click **Actions** → **Stop** to power it down.
4. Click the **Snapshots** tab and then click **Take Snapshot**.

5. Fill in the **Snapshot Name** and optional **Description** fields.
6. Expand **Disks included in this Snapshot** to see the storage volumes to be included in the snapshot.
7. 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.
8. Click **Save**.

9.20.11.5. Creating an virtual machine snapshot in the CLI

You can create a virtual machine (VM) snapshot for an offline or online VM by creating a **VirtualMachineSnapshot** object. Kubevirt will coordinate with the QEMU guest agent to create a snapshot of the online VM.



NOTE

To create snapshots of an online (Running state) VM with the highest integrity, install the QEMU guest agent.

The QEMU guest agent takes a consistent snapshot by attempting to quiesce the VM's file system as much as possible, depending on the system workload. This ensures that in-flight I/O is written to the disk before the snapshot is taken. If the guest agent is not present, quiescing is not possible and a best-effort snapshot is taken. The conditions under which the snapshot was taken are reflected in the snapshot indications that are displayed in the web console or CLI.

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**).
- Optional: Power down the VM for which you want to create a snapshot.

Procedure

1. Create a YAML file to define a **VirtualMachineSnapshot** object that specifies the name of the new **VirtualMachineSnapshot** and the name of the source VM.
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
```

3. Optional: If you are taking an online snapshot, you can use the **wait** command and monitor the status of the snapshot:

- a. Enter the following command:

```
$ kubectl wait my-vm my-vmsnapshot --for condition=Ready
```

- b. Verify the status of the snapshot:

- **InProgress** - The online snapshot operation is still in progress.
- **Succeeded** - The online snapshot operation completed successfully.
- **Failed** - The online snapshot operation failed.

NOTE

Online snapshots have a default time deadline of five minutes (**5m**). If the snapshot does not complete successfully in five minutes, the status is set to **failed**. Afterwards, the file system will be thawed and the VM unfrozen but the status remains **failed** until you delete the failed snapshot image.

To change the default time deadline, add the **FailureDeadline** attribute to the VM snapshot spec with the time designated in minutes (**m**) or in seconds (**s**) that you want to specify before the snapshot operation times out.

To set no deadline, you can specify **0**, though this is generally not recommended, as it can result in an unresponsive VM.

If you do not specify a unit of time such as **m** or **s**, the default is seconds (**s**).

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.

9.20.11.6. Verifying online snapshot creation with snapshot indications

Snapshot indications are contextual information about online virtual machine (VM) snapshot operations. Indications are not available for offline virtual machine (VM) snapshot operations. Indications are helpful in describing details about the online snapshot creation.

Prerequisites

Prerequisites

- To view indications, you must have attempted to create an online VM snapshot using the CLI or the web console.

Procedure

1. Display the output from the snapshot indications by doing one of the following:
 - For snapshots created with the CLI, view indicator output in the **VirtualMachineSnapshot** object YAML, in the **status** field.
 - For snapshots created using the web console, click **VirtualMachineSnapshot > Status** in the **Snapshot details** screen.
2. Verify the status of your online VM snapshot:
 - **Online** indicates that the VM was running during online snapshot creation.
 - **NoGuestAgent** indicates that the QEMU guest agent was not running during online snapshot creation. The QEMU guest agent could not be used to freeze and thaw the file system, either because the QEMU guest agent was not installed or running or due to another error.

9.20.11.7. 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 **Virtualization → VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. If the virtual machine is running, click **Actions → Stop** to power it down.
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 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 VirtualMachineSnapshot**.
6. In the confirmation pop-up window, click **Restore** to restore the VM to its previous configuration represented by the snapshot.

9.20.11.8. 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. You can only restore from an offline 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 ❶
spec:
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: my-vm ❷
  virtualMachineSnapshotName: my-vm snapshot ❸
```

- ❶ The name of the new **VirtualMachineRestore** object.
- ❷ The name of the target VM you want to restore.
- ❸ 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 ❶
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2020-09-30T14:46:28Z"
    reason: Operation complete
    status: "False" ❷
    type: Progressing
  - lastProbeTime: null
    lastTransitionTime: "2020-09-30T14:46:28Z"
    reason: Operation complete
    status: "True" ❸
    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

```

- ❶ Specifies if the process of restoring the VM to the state represented by the snapshot is complete.
- ❷ The **status** field of the **Progressing** condition specifies if the VM is still being restored.
- ❸ The **status** field of the **Ready** condition specifies if the VM restoration process is complete.

9.20.11.9. 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 **Virtualization** → **VirtualMachines** from the side menu.

2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Snapshots** tab. The page displays a list of snapshots associated with the virtual machine.



4. Click the Options menu of the virtual machine snapshot that you want to delete and select **Delete VirtualMachineSnapshot**.
5. In the confirmation pop-up window, click **Delete** to delete the snapshot.

9.20.11.10. 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
```

9.20.11.11. Additional resources

- [CSI Volume Snapshots](#)

9.20.12. 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.

9.20.12.1. 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 1
              operator: In
              values:
                - node01 2
...

```

- 1 The **kubernetes.io/hostname** key uses the node hostname to select a node.
- 2 The hostname of the node.

- Add a unique label to the PV:

```
$ oc label pv <destination-pv> node=node01
```

- 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.
- ❹ The label that you applied to the PV in the previous step.
- ❺ 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.

9.20.13. 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.

9.20.13.1. 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.

9.20.13.2. 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
```

9.20.13.3. 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
```

9.20.13.4. Additional resources

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

9.20.14. Cloning a data volume using smart-cloning

Smart-cloning is a built-in feature of Red Hat OpenShift Data Foundation. Smart-cloning is 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.

9.20.14.1. About 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.

9.20.14.2. 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 source and target PVCs share the same **volumeMode**.
- 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.

9.20.14.3. 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.
- [Customizing the storage profile](#)

9.20.15. 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.

9.20.15.1. 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.

9.20.15.2. Importing a RHEL image as a boot source

You can import a Red Hat Enterprise Linux (RHEL) image as a boot source by specifying a URL for the image.

Prerequisites

- You must have access to a web page with the operating system image. For example: Download Red Hat Enterprise Linux web page with images.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization → Templates** from the side menu.
2. Identify the RHEL template for which you want to configure a boot source and click **Add source**.
3. In the **Add boot source to template** window, select **URL (creates PVC)** from the **Boot source type** list.
4. 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.
5. 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.
6. In the **Add boot source to template** window, paste the URL into the **Import URL** field, and click **Save and import**.

Verification

1. Verify that the template displays a green checkmark in the **Boot source** column on the **Templates** page.

You can now use this template to create RHEL virtual machines.

9.20.15.3. 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 **Source available** on the **Templates** page. 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)**
- **URL (creates PVC)**
- **Clone (creates PVC)**
- **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 Container Platform console, click **Virtualization** → **Templates** from the side menu.
2. Click the options menu beside a template and select **Edit boot source**.
3. Click **Add disk**.
4. In the **Add disk** window, select **Use this disk as a boot source**
5. Enter the disk name and select a **Source**, for example, **Blank (creates PVC)** or **Use an existing PVC**.
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.
7. Select a **Type**, for example, **Disk** or **CD-ROM**.
8. Optional: 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.

9. Optional: Clear **Apply optimized StorageProfile settings** to edit the access mode or volume mode.
10. 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 on the **Catalog** page. You can use this template to create a virtual machine.

9.20.15.4. 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 virtual machine from the template.

Procedure

1. In the OpenShift Container Platform web console, click **Virtualization** → **Catalog** in the side menu.
2. Select the updated template and click **Quick create VirtualMachine**.

The **VirtualMachine details** is displayed with the status **Starting**.

9.20.15.5. Additional resources

- [Creating virtual machine templates](#)
- [Automatic importing and updating of pre-defined boot sources](#)

9.20.16. Hot-plugging virtual disks

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.

After you hot-plug a virtual disk, it remains hot-plugged until you detach (unplug) it, even if you restart the virtual machine.

9.20.16.1. 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 instances.
- 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.

9.20.16.2. 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>
```

9.20.16.3. Hot-plugging a virtual disk using the web console

Hot-plug virtual disks that you want to attach to a virtual machine instance (VMI) while a virtual machine is running. When you hot-plug a virtual disk, it remains attached to the VMI until you hot-unplug it.

Prerequisites

- You must have a running virtual machine to hot-plug a virtual disk.

Procedure

1. Click **Virtualization** → **VirtualMachines** from the side menu.
2. Select the running virtual machine to which you want to hot-plug a virtual disk.
3. On the **VirtualMachine details** page, click the **Disks** tab.
4. Click **Add disk**.
5. In the **Add disk (hot plugged)** window, fill in the information for the virtual disk that you want to hot-plug.
6. Click **Save**.


9.20.16.4. Hot-unplugging a virtual disk using the web console

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 with a hot-plugged disk attached.

Procedure

1. Click **Virtualization** → **VirtualMachines** from the side menu.
2. Select the running virtual machine with the disk you want to hot-unplug to open the **VirtualMachine details** page.
3. On the **Disks** tab, click the Options menu  of the virtual disk that you want to hot-unplug.
4. Click **Detach**.

9.20.17. 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](#)

9.20.17.1. 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.

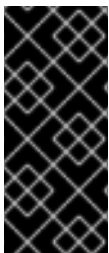
9.20.17.1.1. Importing a container disk into a PVC by using a data volume

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.

9.20.17.1.2. 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.

9.20.17.2. 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.

9.20.17.3. 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.

9.20.17.4. 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.

9.20.18. Preparing CDI scratch space

9.20.18.1. 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.

9.20.18.2. About scratch space

The Containerized Data Importer (CDI) requires scratch space (temporary storage) to complete some operations, such as importing and uploading virtual machine images. During this process, CDI provisions a scratch space PVC equal to the size of the PVC backing the destination data volume (DV). The scratch space PVC is deleted after the operation completes or aborts.

You can define the storage class that is used to bind the scratch space PVC in the **spec.scratchSpaceStorageClass** field of the **HyperConverged** custom resource.

If the defined storage class does not match a storage class in the cluster, then the default storage class defined for the cluster is used. If there is no default storage class defined in the cluster, the storage class used to provision the original DV or PVC is used.



NOTE

CDI requires requesting scratch space with a **file** volume mode, regardless of the PVC backing the origin data volume. If the origin PVC is backed by **block** volume mode, you must define a storage class capable of provisioning **file** volume mode PVCs.

Manual provisioning

If there are no storage classes, CDI uses any PVCs in the project that match the size requirements for the image. If there are no PVCs that match these requirements, the CDI import pod remains in a **Pending** state until an appropriate PVC is made available or until a timeout function kills the pod.

9.20.18.3. 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.
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.

9.20.18.4. 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.

9.20.18.5. 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)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ RAW* ✓ GZ* ✓ XZ*

✓ Supported operation

☐ Unsupported operation

* Requires scratch space

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

9.20.18.6. Additional resources

- [Dynamic provisioning](#)

9.20.19. Re-using persistent volumes

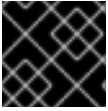
To re-use a statically provisioned persistent volume (PV), you must first reclaim the volume. This involves deleting the PV so that the storage configuration can be re-used.

9.20.19.1. 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.

9.20.19.2. 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>
```

- 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](#).

9.20.20. Expanding a virtual machine disk

You can enlarge the size of a virtual machine's (VM) disk to provide a greater storage capacity by resizing the disk's persistent volume claim (PVC).

However, you cannot reduce the size of a VM disk.

9.20.20.1. Enlarging a virtual machine disk

VM disk enlargement makes extra space available to the virtual machine. However, it is the responsibility of the VM owner to decide how to consume the storage.

If the disk is a **Filesystem** PVC, the matching file expands to the remaining size while reserving some space for file system overhead.

Procedure

- Edit the **PersistentVolumeClaim** manifest of the VM disk that you want to expand:

```
$ oc edit pvc <pvc_name>
```

- Change the value of **spec.resource.requests.storage** attribute to a larger size.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: vm-disk-expand
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 3Gi 1
...
```

- 1 The VM disk size that can be increased

9.20.20.2. Additional resources

- [Extending a basic volume in Windows](#) .
- [Extending an existing file system partition without destroying data in Red Hat Enterprise Linux](#) .
- [Extending a logical volume and its file system online in Red Hat Enterprise Linux](#) .

9.20.21. 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.

9.20.21.1. 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.

9.20.21.2. 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
```

9.20.21.3. 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 10. VIRTUAL MACHINE TEMPLATES

10.1. CREATING VIRTUAL MACHINE TEMPLATES

10.1.1. About virtual machine templates

Preconfigured Red Hat virtual machine templates are listed in the **Virtualization → Templates** 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** page 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.

You can use the filters in the template **Catalog** to sort the templates by attributes such as boot source availability, operating system, and workload.

You cannot edit or delete a **Red Hat Supported** or **Red Hat Provided** template. You can clone the template, save it as a custom virtual machine template, and then edit it.

You can also create a custom virtual machine template by editing a YAML file example.



IMPORTANT

Due to differences in storage behavior, some virtual machine templates are incompatible with single-node OpenShift. To ensure compatibility, do not set the **evictionStrategy** field for any templates or virtual machines that use data volumes or storage profiles.

10.1.2. 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.

10.1.3. Creating a virtual machine template in the web console

You create a virtual machine template by editing a YAML file example in the OpenShift Container Platform web console.

Procedure

1. In the web console, click **Virtualization** → **Templates** in the side menu.
2. Click **Create Template**.
3. Specify the template parameters by editing the YAML file.
4. Click **Create**.
The template is displayed on the **Templates** page.
5. Optional: Click **Download** to download and save the YAML file.

10.1.4. 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 **Source available** on the **Templates** page. 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)**
- **URL (creates PVC)**
- **Clone (creates PVC)**
- **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 Container Platform console, click **Virtualization → Templates** from the side menu.
2. Click the options menu beside a template and select **Edit boot source**.
3. Click **Add disk**.
4. In the **Add disk** window, select **Use this disk as a boot source**
5. Enter the disk name and select a **Source**, for example, **Blank (creates PVC)** or **Use an existing PVC**.
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.
7. Select a **Type**, for example, **Disk** or **CD-ROM**.
8. Optional: 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.

9. Optional: Clear **Apply optimized StorageProfile settings** to edit the access mode or volume mode.
10. 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 on the **Catalog** page. You can use this template to create a virtual machine.

10.1.4.1. 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 on the **Virtualization → Templates** page.

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.
	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 URL 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.
	PVC (creates PVC)	Use a PVC that is already available in the cluster and clone it.
	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 settings	StorageClass	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> <div>  <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.

10.1.5. Marking virtual machine templates as favorites

You can mark frequently used templates as favorites.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization** → **Templates** from the side menu.
2. Click the star icon next to a template to mark it as a favorite.
Favorite templates are displayed at the top of the template list.

10.1.6. Additional resources

- [Configuring the SR-IOV Network Operator](#)
- [Creating and using boot sources](#)
- [Customizing the storage profile](#)

10.2. EDITING VIRTUAL MACHINE TEMPLATES

You can edit a virtual machine template in the web console.



NOTE

You cannot edit a template provided by the Red Hat Virtualization Operator. If you clone the template, you can edit it.

10.2.1. 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.

You can edit labels and annotations for any templates, including those provided by Red Hat. Other fields are editable for user-customized templates only.

Procedure

1. Click **Virtualization** → **Templates** from the side menu.

2. Optional: Use the **Filter** drop-down menu to sort the list of virtual machine templates by attributes such as status, template, node, or operating system (OS).
3. Select a virtual machine template to open the **Template details** page.
4. Click any field that has the pencil icon, which indicates that the field is editable. For example, click the current **Boot mode** setting, such as BIOS or UEFI, to open the **Boot mode** window and select an option from the list.
5. Make the relevant changes and click **Save**.

Editing a virtual machine template will not affect virtual machines already created from that template.

10.2.1.1. Virtual machine template fields

The following table lists the virtual machine template fields that you can edit in the OpenShift Container Platform web console:

Table 10.1. Virtual machine template fields

Tab	Fields or functionality
Details	<ul style="list-style-type: none"> ● Labels ● Annotations ● Display name ● Description ● Workload profile ● CPU/Memory ● Boot mode ● GPU devices ● Host devices
YAML	<ul style="list-style-type: none"> ● View, edit, or download the custom resource.
Scheduling	<ul style="list-style-type: none"> ● Node selector ● Tolerations ● Affinity rules ● Dedicated resources ● Eviction strategy ● Descheduler setting

Tab	Fields or functionality
Network Interfaces	<ul style="list-style-type: none"> ● Add, edit, or delete a network interface.
Disks	<ul style="list-style-type: none"> ● Add, edit, or delete a disk.
Scripts	<ul style="list-style-type: none"> ● cloud-init settings
Parameters (optional)	<ul style="list-style-type: none"> ● Virtual machine name ● cloud-user password

10.2.1.2. 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 **Virtualization → Templates** from the side menu.
2. Select a virtual machine template to open the **Template details** screen.
3. Click the **Network Interfaces** tab.
4. Click **Add Network Interface**.
5. In the **Add Network Interface** window, specify the **Name**, **Model**, **Network**, **Type**, and **MAC Address** of the network interface.
6. Click **Add**.

10.2.1.3. 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 **Virtualization → Templates** from the side menu.
2. Select a virtual machine template to open the **Template details** screen.
3. Click the **Disks** tab and then click **Add disk**.
4. In the **Add disk** window, specify the **Source**, **Name**, **Size**, **Type**, **Interface**, and **Storage Class**.
 - a. Optional: You can enable preallocation if you use a blank disk source and require maximum write performance when creating data volumes. To do so, select the **Enable preallocation** checkbox.


- b. Optional: You can clear **Apply optimized StorageProfile settings** to change the **Volume Mode** and **Access Mode** for the virtual disk. If you do not specify these parameters, the system uses the default values from the **kubevirt-storage-class-defaults** config map.

5. Click **Add**.

10.2.1.4. Editing CD-ROMs for Templates

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

Procedure

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

10.3. ENABLING DEDICATED RESOURCES FOR VIRTUAL MACHINE TEMPLATES

Virtual machines can have resources of a node, such as CPU, dedicated to them to improve performance.

10.3.1. 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.

10.3.2. Prerequisites

- The [CPU Manager](#) must be configured on the node. Verify that the node has the **cpumanager = true** label before scheduling virtual machine workloads.

10.3.3. Enabling dedicated resources for a virtual machine template

You enable dedicated resources for a virtual machine template in the **Details** tab. Virtual machines that were created from a Red Hat template can be configured with dedicated resources.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization** → **Templates** from the side menu.

2. Select a virtual machine template to open the **Template details** page.
3. On the **Scheduling** tab, click the pencil icon beside **Dedicated Resources**.
4. Select **Schedule this workload with dedicated resources (guaranteed policy)**
5. Click **Save**.

10.4. DEPLOYING A VIRTUAL MACHINE TEMPLATE TO A CUSTOM NAMESPACE

Red Hat provides preconfigured virtual machine templates that are installed in the **openshift** namespace. The **ssp-operator** deploys virtual machine templates to the **openshift** namespace by default. Templates in the **openshift** namespace are publicly available to all users. These templates are listed on the **Virtualization → Templates** page for different operating systems.

10.4.1. Creating a custom namespace for templates

You can create a custom namespace that is used to deploy virtual machine templates for use by anyone who has permissions to access those templates. To add templates to a custom namespace, edit the **HyperConverged** custom resource (CR), add **commonTemplatesNamespace** to the spec, and specify the custom namespace for the virtual machine templates. After the **HyperConverged** CR is modified, the **ssp-operator** populates the templates in the custom namespace.

Prerequisites

- Install the OpenShift Container Platform CLI **oc**.
- Log in as a user with cluster-admin privileges.

Procedure

- Use the following command to create your custom namespace:

```
$ oc create namespace <mycustomnamespace>
```

10.4.2. Adding templates to a custom namespace

The **ssp-operator** deploys virtual machine templates to the **openshift** namespace by default. Templates in the **openshift** namespace are publicly available to all users. When a custom namespace is created and templates are added to that namespace, you can modify or delete virtual machine templates in the **openshift** namespace. To add templates to a custom namespace, edit the **HyperConverged** custom resource (CR) which contains the **ssp-operator**.

Procedure

1. View the list of virtual machine templates that are available in the **openshift** namespace.

```
$ oc get templates -n openshift
```

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

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

3. View the list of virtual machine templates that are available in the custom namespace.

```
$ oc get templates -n customnamespace
```

4. Add the **commonTemplatesNamespace** attribute and specify the custom namespace.
Example:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  commonTemplatesNamespace: customnamespace 1
```

- 1 The custom namespace for deploying templates.

5. Save your changes and exit the editor. The **ssp-operator** adds virtual machine templates that exist in the default **openshift** namespace to the custom namespace.

10.4.2.1. Deleting templates from a custom namespace

To delete virtual machine templates from a custom namespace, remove the **commonTemplateNamespace** attribute from the **HyperConverged** custom resource (CR) and delete each template from that custom namespace.

Procedure

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

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

2. Remove the **commonTemplateNamespace** attribute.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  commonTemplatesNamespace: customnamespace 1
```

- 1 The **commonTemplatesNamespace** attribute to be deleted.

3. Delete a specific template from the custom namespace that was removed.

```
$ oc delete templates -n customnamespace <template_name>
```

Verification

- Verify that the template was deleted from the custom namespace.

```
$ oc get templates -n customnamespace
```


10.4.2.2. Additional resources

- [Creating virtual machine templates](#)

10.5. DELETING VIRTUAL MACHINE TEMPLATES

You can delete customized virtual machine templates based on Red Hat templates by using the web console.

You cannot delete Red Hat templates.

10.5.1. Deleting a virtual machine template in the web console


Deleting a virtual machine template permanently removes it from the cluster.



NOTE

You can delete customized virtual machine templates. You cannot delete Red Hat-supplied templates.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization → Templates** from the side menu.
2. Click the Options menu  of a template and select **Delete template**.
3. Click **Delete**.

CHAPTER 11. LIVE MIGRATION

11.1. VIRTUAL MACHINE LIVE MIGRATION

11.1.1. About 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.

You can use live migration if the following conditions are met:

- Shared storage with **ReadWriteMany** (RWX) access mode.
- Sufficient RAM and network bandwidth.
- If the virtual machine uses a host model CPU, the nodes must support the virtual machine's host model CPU.

By default, live migration traffic is encrypted using Transport Layer Security (TLS).

11.1.2. 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](#)

11.2. 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).

11.2.1. 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**.

11.2.2. Cluster-wide live migration limits and timeouts

Table 11.1. 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.	64Mi
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

11.3. 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.



NOTE

If a virtual machine uses a host model CPU, you can perform live migration of that virtual machine only between nodes that support its host CPU model.

11.3.1. 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** action is visible to all users but only admin users can initiate a virtual machine migration.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization** → **VirtualMachines** from the side menu.
2. You can initiate the migration from this page, which makes it easier to perform actions on multiple virtual machines on the same page, or from the **VirtualMachine details** page where you can view comprehensive details of the selected virtual machine:
 - Click the Options menu  next to the virtual machine and select **Migrate**.
 - Click the virtual machine name to open the **VirtualMachine details** page and click **Actions** → **Migrate**.
3. Click **Migrate** to migrate the virtual machine to another node.

11.3.2. 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](#)

11.4. MIGRATING A VIRTUAL MACHINE OVER A DEDICATED ADDITIONAL NETWORK

You can configure a dedicated [Multus network](#) for live migration. A dedicated network minimizes the effects of network saturation on tenant workloads during live migration.

11.4.1. Configuring a dedicated secondary network for virtual machine live migration

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

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You logged in to the cluster as a user with the **cluster-admin** role.
- The Multus Container Network Interface (CNI) plug-in is installed on the cluster.
- Every node on the cluster has at least two Network Interface Cards (NICs), and the NICs to be used for live migration are connected to the same VLAN.
- The virtual machine (VM) is running with the **LiveMigrate** eviction strategy.

Procedure

1. Create a **NetworkAttachmentDefinition** manifest.

Example configuration file

```
apiVersion: "k8s.cni.cncf.io/v1"
```

```

kind: NetworkAttachmentDefinition
metadata:
  name: my-secondary-network ❶
  namespace: openshift-cnv
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "migration-bridge",
    "type": "macvlan",
    "master": "eth1", ❷
    "mode": "bridge",
    "ipam": {
      "type": "whereabouts", ❸
      "range": "10.200.5.0/24" ❹
    }
  }'
```

- ❶ The name of the **NetworkAttachmentDefinition** object.
- ❷ The name of the NIC to be used for live migration.
- ❸ The name of the CNI plug-in that provides the network for this network attachment definition.
- ❹ The IP address range for the secondary network. This range must not have any overlap with the IP addresses of the main network.

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

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

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

Example configuration file

```

apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  liveMigrationConfig:
    completionTimeoutPerGiB: 800
    network: my-secondary-network ❶
    parallelMigrationsPerCluster: 5
    parallelOutboundMigrationsPerNode: 2
    progressTimeout: 150
  ...
```

- ❶ The name of the Multus **NetworkAttachmentDefinition** object to be used for live migrations.

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

Verification

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

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

11.4.2. Additional resources

- [Live migration limits and timeouts](#)

11.5. 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.

11.5.1. 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 on the **VirtualMachines** page or on the **VirtualMachine details** page of the migrating virtual machine.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.

11.5.2. 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

```

11.6. 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.

11.6.1. Cancelling live migration of a virtual machine instance in the web console

You can cancel the live migration of a virtual machine instance in the web console.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization** → **VirtualMachines** from the side menu.



2. Click the Options menu  beside a virtual machine and select **Cancel Migration**.

11.6.2. 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
```

11.7. 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.

11.7.1. 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>
```

11.8. CONFIGURING LIVE MIGRATION POLICIES

You can define different migration configurations for specified groups of virtual machine instances (VMIs) by using a live migration policy.



IMPORTANT

Live migration policy 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/>.

11.8.1. Configuring a live migration policy

Use the **MigrationPolicy** custom resource definition (CRD) to define migration policies for one or more groups of selected virtual machine instances (VMIs).

You can specify groups of VMIs by using any combination of the following:

- Virtual machine instance labels such as **size**, **os**, **gpu**, and other VMI labels.
- Namespace labels such as **priority**, **bandwidth**, **hpc-workload**, and other namespace labels.

For the policy to apply to a specific group of VMIs, all labels on the group of VMIs must match the labels in the policy.



NOTE

If multiple live migration policies apply to a VMI, the policy with the highest number of matching labels takes precedence. If multiple policies meet this criteria, the policies are sorted by lexicographic order of the matching labels keys, and the first one in that order takes precedence.

Procedure

1. Create a **MigrationPolicy** CRD for your specified group of VMIs. The following example YAML configures a group with the labels **hpc-workloads:true**, **xyz-workloads-type: ""**, **workload-type: db**, and **operating-system: ""**:

```
apiVersion: migrations.kubevirt.io/v1alpha1
kind: MigrationPolicy
metadata:
  name: my-awesome-policy
spec:
  # Migration Configuration
  allowAutoConverge: true
  bandwidthPerMigration: 217Ki
  completionTimeoutPerGiB: 23
  allowPostCopy: false

  # Matching to VMIs
  selectors:
    namespaceSelector: 1
    matchLabels:
      hpc-workloads: "True"
      xyz-workloads-type: ""
    virtualMachineInstanceSelector: 2
    matchLabels:
      workload-type: "db"
      operating-system: ""
```

- 1 Use **namespaceSelector** to define a group of VMIs by using namespace labels.
- 2 Use **virtualMachineInstanceSelector** to define a group of VMIs by using VMI labels.

CHAPTER 12. NODE MAINTENANCE

12.1. ABOUT NODE MAINTENANCE

12.1.1. About node maintenance mode

Nodes can be placed into maintenance mode using the **oc adm** utility, or using **NodeMaintenance** custom resources (CRs).



NOTE

The **node-maintenance-operator** (NMO) is no longer shipped with OpenShift Virtualization. It is now available to deploy as a standalone Operator from the **OperatorHub** in the OpenShift Container Platform web console, or by using the OpenShift CLI (**oc**).

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.

12.1.2. 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.

12.1.3. Additional resources

- [Installing the Node Maintenance Operator by using the CLI](#)
- [Setting a node to maintenance mode](#)
- [Resuming a node from maintenance mode](#)
- [About RunStrategies for virtual machines](#)
- [Virtual machine live migration](#)
- [Configuring virtual machine eviction strategy](#)

12.2. 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.

12.2.1. 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

12.3. 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.

12.3.1. 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 12.1. 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.

12.3.2. 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 12.2. 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 12.3. 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 12.4. 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
```

12.3.3. 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-cnv
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.

12.4. PREVENTING NODE RECONCILIATION

Use **skip-node** annotation to prevent the **node-labeller** from reconciling a node.

12.4.1. 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.

12.4.2. Additional resources

- [Managing node labeling for obsolete CPU models](#)

CHAPTER 13. LOGGING, EVENTS, AND MONITORING

13.1. REVIEWING VIRTUALIZATION OVERVIEW

The **Virtualization Overview** page provides a comprehensive view of virtualization resources, details, status, and top consumers. By gaining an insight into the overall health of OpenShift Virtualization, you can determine if intervention is required to resolve specific issues identified by examining the data.

Use the **Getting Started** resources to access quick starts, read the latest blogs on virtualization, and learn how to use operators. Obtain complete information about alerts, events, inventory, and status of virtual machines. Customize the **Top Consumer** cards to obtain data on high utilization of a specific resource by projects, virtual machines, or nodes. Click **View virtualization dashboard** for quick access to the [Dashboards](#) page.

13.1.1. Prerequisites

To use the **vCPU wait** metric in the **Top Consumers** card, 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.

13.1.2. Resources monitored actively in the Virtualization Overview page

The following table shows actively monitored resources, metrics, and fields in the **Virtualization Overview** page. This information is useful when you need to obtain relevant data and intervene to resolve a problem.

Monitored resources, fields, and metrics	Description
Details	A brief overview of service and version information for OpenShift Virtualization .
Status	Alerts for virtualization and networking.
Activity	Ongoing events for virtual machines. Messages are related to recent activity in the cluster, such as pod creation or virtual machine migration to another host.
Running VMs by Template	The donut chart displays a unique color for each virtual machine template and shows the number of running virtual machines that use each template.
Inventory	Total number of active virtual machines, templates, nodes, and networks.
Status of VMs	Current status of virtual machines: running , provisioning , starting , migrating , paused , stopping , terminating , and unknown .

Permissions	Tasks for which capabilities are enabled through permissions: Access to public templates , Access to public boot sources , Clone a VM , Attach VM to multiple networks , Upload a base image from local disk , and Share templates .
-------------	--

13.1.3. Resources monitored for top consumption

The **Top Consumers** cards in **Virtualization Overview** page display projects, virtual machines or nodes with maximum consumption of a resource. You can select a project, a virtual machine, or a node and view the top five or top ten consumers of a specific resource.



NOTE

Viewing the maximum resource consumption is limited to the top five or top ten consumers within each **Top Consumers** card.

The following table shows resources monitored for top consumers.

Resources monitored for top consumption	Description
CPU	Projects, virtual machines, or nodes consuming the most CPU.
Memory	Projects, virtual machines, or nodes consuming the most memory (in bytes). The unit of display (for example, MiB or GiB) is determined by the size of the resource consumption.
Used filesystem	Projects, virtual machines, or nodes with the highest consumption of filesystems (in bytes). The unit of display (for example, MiB or GiB) is determined by the size of the resource consumption.
Memory swap	Projects, virtual machines, or nodes consuming the most memory pressure when memory is swapped .
vCPU wait	Projects, virtual machines, or nodes experiencing the maximum wait time (in seconds) for the vCPUs.
Storage throughput	Projects, virtual machines, or nodes with the highest data transfer rate to and from the storage media (in mbps).
Storage IOPS	Projects, virtual machines, or nodes with the highest amount of storage IOPS (input/output operations per second) over a time period.

13.1.4. Reviewing top consumers for projects, virtual machines, and nodes

You can view the top consumers of resources for a selected project, virtual machine, or node in the **Virtualization Overview** page.

Prerequisites

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

Procedure

1. In the **Administrator** perspective in the OpenShift Virtualization web console, navigate to **Virtualization → Overview**.
2. Navigate to the **Top Consumers** cards.
3. From the drop-down menu, select **Show top 5** or **Show top 10**.
4. For a **Top Consumer** card, select the type of resource from the drop-down menu: **CPU**, **Memory**, **Used Filesystem**, **Memory Swap**, **vCPU Wait**, or **Storage Throughput**.
5. Select **By Project**, **By VM**, or **By Node**. A list of the top five or top ten consumers of the selected resource is displayed.

13.1.5. Additional resources

- [Monitoring overview](#)
- [Reviewing monitoring dashboards](#)
- [Dashboards](#)

13.2. VIEWING OPENSIFT VIRTUALIZATION LOGS

You can view logs for OpenShift Virtualization components and virtual machines by using the web console or the **oc** CLI. You can retrieve virtual machine logs from the **virt-launcher** pod. To control log verbosity, edit the **HyperConverged** custom resource.

13.2.1. Viewing OpenShift Virtualization logs with the CLI

Configure log verbosity for OpenShift Virtualization components by editing the **HyperConverged** custom resource (CR). Then, view logs for the component pods by using the **oc** CLI tool.

Procedure

1. To set log verbosity for specific components, open the **HyperConverged** CR in your default text editor by running the following command:

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

2. Set the log level for one or more components by editing the **spec.logVerbosityConfig** stanza. For example:

```
apiVersion: hco.kubevirt.io/v1beta1
```

```

kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  logVerbosityConfig:
    kubevirt:
      virtAPI: 5 1
      virtController: 4
      virtHandler: 3
      virtLauncher: 2
      virtOperator: 6

```

- 1 The log verbosity value must be an integer in the range **1–9**, where a higher number indicates a more detailed log. In this example, the **virtAPI** component logs are exposed if their priority level is **5** or higher.

3. Apply your changes by saving and exiting the editor.
4. View a list of pods in the OpenShift Virtualization namespace by running the following command:

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

Example 13.1. Example output

NAME	READY	STATUS	RESTARTS	AGE
disks-images-provider-7gqbc	1/1	Running	0	32m
disks-images-provider-vg4kx	1/1	Running	0	32m
virt-api-57fcc4497b-7qfmc	1/1	Running	0	31m
virt-api-57fcc4497b-tx9nc	1/1	Running	0	31m
virt-controller-76c784655f-7fp6m	1/1	Running	0	30m
virt-controller-76c784655f-f4pbd	1/1	Running	0	30m
virt-handler-2m86x	1/1	Running	0	30m
virt-handler-9qs6z	1/1	Running	0	30m
virt-operator-7ccfdbf65f-q5snk	1/1	Running	0	32m
virt-operator-7ccfdbf65f-vllz8	1/1	Running	0	32m

5. To view logs for a component pod, run the following command:

```
$ oc logs -n openshift-cnv <pod_name>
```

For example:

```
$ oc logs -n openshift-cnv virt-handler-2m86x
```



NOTE

If a pod fails to start, you can use the **--previous** option to view logs from the last attempt.

To monitor log output in real time, use the **-f** option.

Example 13.2. Example output

```
{
  "component": "virt-handler",
  "level": "info",
  "msg": "set verbosity to 2",
  "pos": "virt-handler.go:453",
  "timestamp": "2022-04-17T08:58:37.373695Z"
}
{"component": "virt-handler", "level": "info", "msg": "set verbosity to 2", "pos": "virt-handler.go:453", "timestamp": "2022-04-17T08:58:37.373726Z"}
{"component": "virt-handler", "level": "info", "msg": "setting rate limiter to 5 QPS and 10 Burst", "pos": "virt-handler.go:462", "timestamp": "2022-04-17T08:58:37.373782Z"}
{"component": "virt-handler", "level": "info", "msg": "CPU features of a minimum baseline CPU model: map[apic:true clflush:true cmov:true cx16:true cx8:true de:true fpu:true fxsr:true lahf_lm:true lm:true mca:true mce:true mmx:true msr:true mtrr:true nx:true pae:true pat:true pge:true pni:true pse:true pse36:true sep:true sse:true sse2:true sse4.1:true ssse3:true syscall:true tsc:true]", "pos": "cpu_plugin.go:96", "timestamp": "2022-04-17T08:58:37.390221Z"}
{"component": "virt-handler", "level": "warning", "msg": "host model mode is expected to contain only one model", "pos": "cpu_plugin.go:103", "timestamp": "2022-04-17T08:58:37.390263Z"}
{"component": "virt-handler", "level": "info", "msg": "node-labeller is running", "pos": "node_labeller.go:94", "timestamp": "2022-04-17T08:58:37.391011Z"}
```

13.2.2. Viewing virtual machine logs in the web console

Get virtual machine logs from the associated virtual machine launcher pod.

Procedure

1. In the OpenShift Container Platform console, click **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Details** tab.
4. Click the **virt-launcher-`<name>`** pod in the **Pod** section to open the **Pod details** page.
5. Click the **Logs** tab to view the pod logs.

13.2.3. Common error messages

The following error messages might appear in OpenShift Virtualization logs:

ErrImagePull or ImagePullBackOff

Indicates an incorrect deployment configuration or problems with the images that are referenced.

13.3. VIEWING EVENTS**13.3.1. About 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](#) .

13.3.2. Viewing the events for a virtual machine in the web console

You can view streaming events for a running virtual machine on the **VirtualMachine details** page of the web console.

Procedure

1. Click **Virtualization** → **VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Events** tab to view streaming events for the virtual machine.
 - The **||** button pauses the events stream.
 - The **▶** button resumes a paused events stream.

13.3.3. 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
```

13.3.4. 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>
```

13.4. DIAGNOSING DATA VOLUMES USING EVENTS AND CONDITIONS

Use the **oc describe** command to analyze and help resolve issues with data volumes.

13.4.1. 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.

13.4.2. 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

```

Status:
Conditions:
  Last Heart Beat Time: 2020-07-15T04:31:39Z

```


Last Transition Time: 2020-07-15T04:31:39Z

Status: True

Type: Ready

13.5. 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.

13.5.1. The Virtual Machines dashboard


Access virtual machines (VMs) from the OpenShift Container Platform web console by navigating to the **Virtualization** → **VirtualMachines** page and clicking a virtual machine (VM) to view the **VirtualMachine details** page.

The **Overview** tab displays the following cards:

- **Details** provides identifying information about the virtual machine, including:

- Name
- Status
- Date of creation
- Operating system
- CPU and memory
- Hostname
- Template

If the VM is running, there is an active VNC preview window and a link to open the VNC web

console. The **Options** menu  on the **Details** card provides options to stop or pause the VM, and to copy the **ssh over nodeport** command for SSH tunneling.

- **Alerts** lists VM alerts with three severity levels:
 - Critical
 - Warning
 - Info
- **Snapshots** provides information about VM snapshots and the ability to take a snapshot. For each snapshot listed, the **Snapshots** card includes:
 - A visual indicator of the status of the snapshot, if it is successfully created, is still in progress, or has failed.

- An **Options** menu  with options to restore or delete the snapshot

- **Network interfaces** provides information about the network interfaces of the VM, including:
 - Name (Network and Type)
 - IP address, with the ability to copy the IP address to the clipboard
- **Disks** lists VM disks details, including:
 - Name
 - Drive
 - Size
- **Utilization** includes charts that display usage data for:
 - CPU
 - Memory
 - Storage
 - Network transfer

**NOTE**

Use the drop-down list to choose a duration for the utilization data. The available options are **5 minutes**, **1 hour**, **6 hours**, and **24 hours**.

- **Hardware Devices** provides information about GPU and host devices, including:
 - Resource name
 - Hardware device name

13.6. 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.

13.6.1. 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.

13.6.2. 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: ❶
      port: 1500 ❷
      path: /healthz ❸
      httpHeaders:
        - name: Custom-Header
          value: Awesome
      initialDelaySeconds: 120 ❹
      periodSeconds: 20 ❺
      timeoutSeconds: 10 ❻
      failureThreshold: 3 ❼
      successThreshold: 3 ❽
# ...
```

- ❶ The HTTP GET request to perform to connect to the VMI.
- ❷ The port of the VMI that the probe queries. In the above example, the probe queries port 1500.
- ❸ 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.
- ❹ The time, in seconds, after the VMI starts before the readiness probe is initiated.
- ❺ The delay, in seconds, between performing probes. The default delay is 10 seconds. This value must be greater than **timeoutSeconds**.
- ❻ 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
```

13.6.3. 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
```

13.6.4. 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 ❶
    periodSeconds: 20 ❷
    httpGet: ❸
      port: 1500 ❹
      path: /healthz ❺
      httpHeaders:
        - name: Custom-Header
          value: Awesome
    timeoutSeconds: 10 ❻
# ...
```

- ❶ The time, in seconds, after the VMI starts before the liveness probe is initiated.
- ❷ The delay, in seconds, between performing probes. The default delay is 10 seconds. This value must be greater than **timeoutSeconds**.
- ❸ The HTTP GET request to perform to connect to the VMI.
- ❹ 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.
- ❺ 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.
- ❻ 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
```

13.6.5. Template: Virtual machine configuration file for defining health checks

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    special: vm-fedora
```

```

name: vm-fedora
spec:
  template:
    metadata:
      labels:
        special: vm-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: 180
      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

```

13.6.6. Additional resources

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

13.7. 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*.

13.7.1. 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).

13.8. REVIEWING RESOURCE USAGE BY VIRTUAL MACHINES

Dashboards in the OpenShift Container Platform web console provide visual representations of cluster metrics to help you to quickly understand the state of your cluster. Dashboards belong to the [Monitoring overview](#) that provides monitoring for core platform components.

The OpenShift Virtualization dashboard provides data on resource consumption for virtual machines and associated pods. The visualization metrics displayed in the OpenShift Virtualization dashboard are based on [Prometheus Query Language \(PromQL\) queries](#).

A [monitoring role](#) is required to monitor user-defined namespaces in the OpenShift Virtualization dashboard.

13.8.1. About reviewing top consumers

In the OpenShift Virtualization dashboard, you can select a specific time period and view the top consumers of resources within that time period. Top consumers are virtual machines or **virt-launcher** pods that are consuming the highest amount of resources.

The following table shows resources monitored in the dashboard and describes the metrics associated with each resource for top consumers.

Monitored resources	Description
Memory swap traffic	Virtual machines consuming the most memory pressure when swapping memory.
vCPU wait	Virtual machines experiencing the maximum wait time (in seconds) for their vCPUs.
CPU usage by pod	The virt-launcher pods that are using the most CPU.
Network traffic	Virtual machines that are saturating the network by receiving the most amount of network traffic (in bytes).
Storage traffic	Virtual machines with the highest amount (in bytes) of storage-related traffic.
Storage IOPS	Virtual machines with the highest amount of I/O operations per second over a time period.
Memory usage	The virt-launcher pods that are using the most memory (in bytes).



NOTE

Viewing the maximum resource consumption is limited to the top five consumers.

13.8.2. Reviewing top consumers

In the **Administrator** perspective, you can view the OpenShift Virtualization dashboard where top consumers of resources are displayed.

Prerequisites

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

Procedure

1. In the **Administrator** perspective in the OpenShift Virtualization web console, navigate to **Observe → Dashboards**.
2. Select the **KubeVirt/Infrastructure Resources/Top Consumers** dashboard from the **Dashboard** list.
3. Select a predefined time period from the drop-down menu for **Period**. You can review the data for top consumers in the tables.
4. Optional: Click **Inspect** to view or edit the Prometheus Query Language (PromQL) query associated with the top consumers for a table.

13.8.3. Additional resources

- [Monitoring overview](#)
- [Reviewing monitoring dashboards](#)

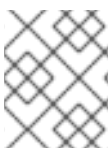
13.9. OPENSIFT CONTAINER PLATFORM CLUSTER MONITORING, LOGGING, AND TELEMETRY

OpenShift Container Platform provides various resources for monitoring at the cluster level.

13.9.1. 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 4.11, 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.

13.9.2. About logging subsystem components

The logging subsystem 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 the logging subsystem 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.

13.9.3. 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.

13.9.3.1. 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

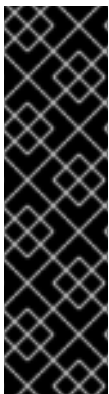
Telemetry does not collect identifying information such as user names or passwords. Red Hat does not intend to collect personal information. If Red Hat discovers that personal information has been inadvertently received, Red Hat will delete such information. To the extent that any telemetry data constitutes personal data, please refer to the [Red Hat Privacy Statement](#) for more information about Red Hat's privacy practices.

13.9.4. CLI troubleshooting and debugging commands

For a list of the **oc** client troubleshooting and debugging commands, see the [OpenShift Container Platform CLI tools](#) documentation.

13.10. RUNNING CLUSTER CHECKUPS

OpenShift Virtualization 4.11 includes a diagnostic framework to run predefined checkups that can be used for cluster maintenance and troubleshooting.



IMPORTANT

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

For more information about the support scope of Red Hat Technology Preview features, see <https://access.redhat.com/support/offerings/techpreview/>.

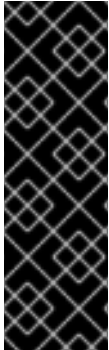
13.10.1. About the OpenShift Container Platform cluster checkpoint framework

A *checkpoint* is an automated test workload that allows you to verify if a specific cluster functionality works as expected. The cluster checkpoint framework uses native Kubernetes resources to configure and execute the checkpoint.

By using predefined checkpoints, cluster administrators can improve cluster maintainability, troubleshoot unexpected behavior, minimize errors, and save time. They can also review the results of the checkpoint and share them with experts for further analysis. Vendors can write and publish checkpoints for features or

services that they provide and verify that their customer environments are configured correctly.

Running a predefined checkup in the cluster involves setting up the namespace and service account for the framework, creating the **ClusterRole** and **ClusterRoleBinding** objects for the service account, enabling permissions for the checkup, and creating the input config map and the checkup job. You can run a checkup multiple times.



IMPORTANT

You must always:

- Verify that the checkup image is from a trustworthy source before applying it.
- Review the checkup permissions before creating the **ClusterRole** objects.
- Verify the names of the **ClusterRole** objects in the config map. This is because the framework automatically binds these permissions to the checkup instance.

13.10.2. Checking network connectivity and latency for virtual machines on a secondary network

As a cluster administrator, you use a predefined checkup to verify network connectivity and measure latency between virtual machines (VMs) that are attached to a secondary network interface.

To run a checkup for the first time, follow the steps in the procedure.

If you have previously run a checkup, skip to step 5 of the procedure because the steps to install the framework and enable permissions for the checkup are not required.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You logged in to the cluster as a user with the **cluster-admin** role.
- The cluster has at least two worker nodes.
- The Multus Container Network Interface (CNI) plug-in is installed on the cluster.
- You configured a network attachment definition for a namespace.

Procedure

1. Create a configuration file that contains the resources to set up the framework. This includes a namespace and service account for the framework, and the **ClusterRole** and **ClusterRoleBinding** objects to define permissions for the service account.

Example 13.3. Example framework manifest file

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: kiagnose
---
apiVersion: v1
```

```

kind: ServiceAccount
metadata:
  name: kiagnose
  namespace: kiagnose
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kiagnose
rules:
  - apiGroups: [ "" ]
    resources: [ "configmaps" ]
    verbs:
      - get
      - list
      - create
      - update
      - patch
  - apiGroups: [ "" ]
    resources: [ "namespaces" ]
    verbs:
      - get
      - list
      - create
      - delete
      - watch
  - apiGroups: [ "" ]
    resources: [ "serviceaccounts" ]
    verbs:
      - get
      - list
      - create
  - apiGroups: [ "rbac.authorization.k8s.io" ]
    resources:
      - roles
      - rolebindings
      - clusterrolebindings
    verbs:
      - get
      - list
      - create
      - delete
  - apiGroups: [ "rbac.authorization.k8s.io" ]
    resources:
      - clusterroles
    verbs:
      - get
      - list
      - create
      - bind
  - apiGroups: [ "batch" ]
    resources: [ "jobs" ]
    verbs:
      - get
      - list
      - create

```

```

- delete
- watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kiagnose
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: kiagnose
subjects:
- kind: ServiceAccount
  name: kiagnose
  namespace: kiagnose
...

```

2. Apply the framework manifest:

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

3. Create a configuration file that contains the **ClusterRole** and **Role** objects with permissions that the checkup requires for cluster access:

Example cluster role manifest file

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: kubevirt-vm-latency-checker
rules:
- apiGroups: ["kubevirt.io"]
  resources: ["virtualmachineinstances"]
  verbs: ["get", "create", "delete"]
- apiGroups: ["subresources.kubevirt.io"]
  resources: ["virtualmachineinstances/console"]
  verbs: ["get"]
- apiGroups: ["k8s.cni.cncf.io"]
  resources: ["network-attachment-definitions"]
  verbs: ["get"]

```

4. Apply the checkup roles manifest:

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

5. Create a **ConfigMap** manifest that contains the input parameters for the checkup. The config map provides the input for the framework to run the checkup and also stores the results of the checkup.

Example input config map

```

apiVersion: v1
kind: ConfigMap

```

```

metadata:
  name: kubevirt-vm-latency-checkup
  namespace: kiagnose
data:
  spec.image: registry.redhat.io/container-native-virtualization/vm-network-latency-
  checkup:v4.11.0
  spec.timeout: 10m
  spec.clusterRoles: |
    kubevirt-vmis-manager
  spec.param.network_attachment_definition_namespace: "default" ❶
  spec.param.network_attachment_definition_name: "bridge-network" ❷
  spec.param.max_desired_latency_milliseconds: "10" ❸
  spec.param.sample_duration_seconds: "5" ❹

```

- ❶ The namespace where the **NetworkAttachmentDefinition** object resides.
- ❷ The name of the **NetworkAttachmentDefinition** object.
- ❸ Optional: The maximum desired latency, in milliseconds, between the virtual machines. If the measured latency exceeds this value, the check fails.
- ❹ Optional: The duration of the latency check, in seconds.

6. Create the config map in the framework's namespace:

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

7. Create a **Job** object to run the checkup:

Example job manifest

```

apiVersion: batch/v1
kind: Job
metadata:
  name: kubevirt-vm-latency-checkup
  namespace: kiagnose
spec:
  backoffLimit: 0
  template:
    spec:
      serviceAccount: kiagnose
      restartPolicy: Never
      containers:
        - name: framework
          image: registry.redhat.io/container-native-virtualization/checkup-framework:v4.11.0
          env:
            - name: CONFIGMAP_NAMESPACE
              value: kiagnose
            - name: CONFIGMAP_NAME
              value: kubevirt-vm-latency-checkup

```

8. Apply the **Job** manifest. The checkup uses the ping utility to verify connectivity and measure latency.

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

9. Wait for the job to complete:

```
$ oc wait --for=condition=complete --timeout=10m job.batch/kubevirt-vm-latency-checkup -n kiagnose
```

10. Review the results of the latency checkup by retrieving the status of the **ConfigMap** object. If the measured latency is greater than the value of the **spec.param.max_desired_latency_milliseconds** attribute, the checkup fails and returns an error.

```
$ oc get configmap kubevirt-vm-latency-checkup -n kiagnose -o yaml
```

Example output config map (success)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kubevirt-vm-latency-checkup
  namespace: kiagnose
...
status.succeeded: "true"
status.failureReason: ""
status.result.minLatencyNanoSec: 2000
status.result.maxLatencyNanoSec: 3000
status.result.avgLatencyNanoSec: 2500
status.results.measurementDurationSec: 300
...
```

11. Delete the framework and checkup resources that you previously created. This includes the job, config map, cluster role, and framework manifest files.



NOTE

Do not delete the framework and cluster role manifest files if you plan to run another checkup.

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

13.10.3. Additional resources

- [Attaching a virtual machine to multiple networks](#)

13.11. 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.

13.11.1. 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.

13.11.2. About 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.

13.11.2.1. 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.

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. Select the **Administrator** perspective in the OpenShift Container Platform web console.
2. Select **Observe → Metrics**.
3. Select **Insert Metric at Cursor** to view a list of predefined queries.
4. To create a custom query, add your Prometheus Query Language (PromQL) query to the **Expression** field.

**NOTE**

As you type a PromQL expression, autocomplete suggestions appear in a drop-down list. These suggestions include functions, metrics, labels, and time tokens. You can use the keyboard arrows to select one of these suggested items and then press Enter to add the item to your expression. You can also move your mouse pointer over a suggested item to view a brief description of that item.

5. To add multiple queries, select **Add Query**.

6. To duplicate an existing query, select  next to the query, then choose **Duplicate query**.

7. To delete a query, select  next to the query, then choose **Delete query**.

8. To disable a query from being run, select  next to the query and choose **Disable query**.

9. To run queries that you created, select **Run Queries**. 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.

10. Optional: The page URL now contains the queries you ran. To use this set of queries again in the future, save this URL.

13.11.2.2. 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 in the **Observe** → **Metrics** page in the web console 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

1. Select the **Developer** perspective in the OpenShift Container Platform web console.
2. Select **Observe → Metrics**.
3. Select the project that you want to view metrics for in the **Project:** list.
4. Select a query from the **Select query** list, or create a custom PromQL query based on the selected query by selecting **Show PromQL**.
5. Optional: Select **Custom query** from the **Select query** list to enter a new query. As you type, autocomplete suggestions appear in a drop-down list. These suggestions include functions and metrics. Click a suggested item to select it.



NOTE

In the **Developer** perspective, you can only run one query at a time.

13.11.3. 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.

13.11.3.1. 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.

13.11.3.2. 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.

13.11.3.3. Storage metrics

13.11.3.3.1. 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.

13.11.3.3.2. Storage snapshot data

kubevirt_vmsnapshot_disks_restored_from_source_total

Returns the total number of virtual machine disks restored from the source virtual machine.

kubevirt_vmsnapshot_disks_restored_from_source_bytes

Returns the amount of space in bytes restored from the source virtual machine.

Examples of storage snapshot data queries

```
kubevirt_vmsnapshot_disks_restored_from_source_total{vm_name="simple-vm",  
vm_namespace="default"} 1
```

- 1** This query returns the total number of virtual machine disks restored from the source virtual machine.

```
kubevirt_vmsnapshot_disks_restored_from_source_bytes{vm_name="simple-vm",  
vm_namespace="default"} 1
```

- 1** This query returns the amount of space in bytes restored from the source virtual machine.

13.11.3.3. I/O performance

The following queries can determine the I/O performance of storage devices:

kubevirt_vmi_storage_iops_read_total

Returns the amount of write I/O operations the virtual machine is performing per second.

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.

13.11.3.4. 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.

13.11.4. Additional resources

- [Monitoring overview](#)

13.12. EXPOSING CUSTOM METRICS FOR VIRTUAL MACHINES

OpenShift Container Platform includes a pre-configured, pre-installed, and self-updating monitoring stack that provides monitoring for core platform components. This monitoring stack is based on the Prometheus monitoring system. Prometheus is a time-series database and a rule evaluation engine for metrics.

In addition to using the OpenShift Container Platform monitoring stack, you can enable monitoring for user-defined projects by using the CLI and query custom metrics that are exposed for virtual machines through the **node-exporter** service.

13.12.1. Configuring the node exporter service

The node-exporter agent is deployed on every virtual machine in the cluster from which you want to collect metrics. Configure the node-exporter agent as a service to expose internal metrics and processes that are associated with virtual machines.

Prerequisites

- Install the OpenShift Container Platform CLI **oc**.
- Log in to the cluster as a user with **cluster-admin** privileges.
- Create the **cluster-monitoring-config ConfigMap** object in the **openshift-monitoring** project.
- Configure the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project by setting **enableUserWorkload** to **true**.

Procedure

1. Create the **Service** YAML file. In the following example, the file is called **node-exporter-service.yaml**.

```
kind: Service
apiVersion: v1
metadata:
  name: node-exporter-service 1
  namespace: dynamation 2
  labels:
    servicetype: metrics 3
spec:
  ports:
    - name: exmet 4
      protocol: TCP
      port: 9100 5
```

```
targetPort: 9100 6
type: ClusterIP
selector:
  monitor: metrics 7
```

- 1 The node-exporter service that exposes the metrics from the virtual machines.
- 2 The namespace where the service is created.
- 3 The label for the service. The **ServiceMonitor** uses this label to match this service.
- 4 The name given to the port that exposes metrics on port 9100 for the **ClusterIP** service.
- 5 The target port used by **node-exporter-service** to listen for requests.
- 6 The TCP port number of the virtual machine that is configured with the **monitor** label.
- 7 The label used to match the virtual machine's pods. In this example, any virtual machine's pod with the label **monitor** and a value of **metrics** will be matched.

2. Create the node-exporter service:

```
$ oc create -f node-exporter-service.yaml
```

13.12.2. Configuring a virtual machine with the node exporter service

Download the **node-exporter** file on to the virtual machine. Then, create a **systemd** service that runs the node-exporter service when the virtual machine boots.

Prerequisites

- The pods for the component are running in the **openshift-user-workload-monitoring** project.
- Grant the **monitoring-edit** role to users who need to monitor this user-defined project.

Procedure

1. Log on to the virtual machine.
2. Download the **node-exporter** file on to the virtual machine by using the directory path that applies to the version of **node-exporter** file.

```
$ wget
https://github.com/prometheus/node_exporter/releases/download/v1.3.1/node_exporter-
1.3.1.linux-amd64.tar.gz
```

3. Extract the executable and place it in the **/usr/bin** directory.

```
$ sudo tar xvf node_exporter-1.3.1.linux-amd64.tar.gz \
--directory /usr/bin --strip 1 "*/node_exporter"
```

4. Create a **node_exporter.service** file in this directory path: **/etc/systemd/system**. This **systemd** service file runs the node-exporter service when the virtual machine reboots.

—

```
[Unit]
Description=Prometheus Metrics Exporter
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=always
RestartSec=1
User=root
ExecStart=/usr/bin/node_exporter

[Install]
WantedBy=multi-user.target
```

5. Enable and start the **systemd** service.

```
$ sudo systemctl enable node_exporter.service
$ sudo systemctl start node_exporter.service
```

Verification

- Verify that the node-exporter agent is reporting metrics from the virtual machine.

```
$ curl http://localhost:9100/metrics
```

Example output

```
go_gc_duration_seconds{quantile="0"} 1.5244e-05
go_gc_duration_seconds{quantile="0.25"} 3.0449e-05
go_gc_duration_seconds{quantile="0.5"} 3.7913e-05
```

13.12.3. Creating a custom monitoring label for virtual machines

To enable queries to multiple virtual machines from a single service, add a custom label in the virtual machine's YAML file.

Prerequisites

- Install the OpenShift Container Platform CLI **oc**.
- Log in as a user with **cluster-admin** privileges.
- Access to the web console for stop and restart a virtual machine.

Procedure

1. Edit the **template** spec of your virtual machine configuration file. In this example, the label **monitor** has the value **metrics**.

```
spec:
  template:
    metadata:
```



```
labels:
  monitor: metrics
```

2. Stop and restart the virtual machine to create a new pod with the label name given to the **monitor** label.

13.12.3.1. Querying the node-exporter service for metrics

Metrics are exposed for virtual machines through an HTTP service endpoint under the **/metrics** canonical name. When you query for metrics, Prometheus directly scrapes the metrics from the metrics endpoint exposed by the virtual machines and presents these metrics for viewing.

Prerequisites

- You have access to the cluster as a user with **cluster-admin** privileges or the **monitoring-edit** role.
- You have enabled monitoring for the user-defined project by configuring the node-exporter service.

Procedure

1. Obtain the HTTP service endpoint by specifying the namespace for the service:

```
$ oc get service -n <namespace> <node-exporter-service>
```

2. To list all available metrics for the node-exporter service, query the **metrics** resource.

```
$ curl http://<172.30.226.162:9100>/metrics | grep -vE "^#|^$"
```

Example output

```
node_arp_entries{device="eth0"} 1
node_boot_time_seconds 1.643153218e+09
node_context_switches_total 4.4938158e+07
node_cooling_device_cur_state{name="0",type="Processor"} 0
node_cooling_device_max_state{name="0",type="Processor"} 0
node_cpu_guest_seconds_total{cpu="0",mode="nice"} 0
node_cpu_guest_seconds_total{cpu="0",mode="user"} 0
node_cpu_seconds_total{cpu="0",mode="idle"} 1.10586485e+06
node_cpu_seconds_total{cpu="0",mode="iowait"} 37.61
node_cpu_seconds_total{cpu="0",mode="irq"} 233.91
node_cpu_seconds_total{cpu="0",mode="nice"} 551.47
node_cpu_seconds_total{cpu="0",mode="softirq"} 87.3
node_cpu_seconds_total{cpu="0",mode="steal"} 86.12
node_cpu_seconds_total{cpu="0",mode="system"} 464.15
node_cpu_seconds_total{cpu="0",mode="user"} 1075.2
node_disk_discard_time_seconds_total{device="vda"} 0
node_disk_discard_time_seconds_total{device="vdb"} 0
node_disk_discarded_sectors_total{device="vda"} 0
node_disk_discarded_sectors_total{device="vdb"} 0
node_disk_discards_completed_total{device="vda"} 0
node_disk_discards_completed_total{device="vdb"} 0
node_disk_discards_merged_total{device="vda"} 0
```

```

node_disk_discards_merged_total{device="vdb"} 0
node_disk_info{device="vda",major="252",minor="0"} 1
node_disk_info{device="vdb",major="252",minor="16"} 1
node_disk_io_now{device="vda"} 0
node_disk_io_now{device="vdb"} 0
node_disk_io_time_seconds_total{device="vda"} 174
node_disk_io_time_seconds_total{device="vdb"} 0.054
node_disk_io_time_weighted_seconds_total{device="vda"} 259.79200000000003
node_disk_io_time_weighted_seconds_total{device="vdb"} 0.039
node_disk_read_bytes_total{device="vda"} 3.71867136e+08
node_disk_read_bytes_total{device="vdb"} 366592
node_disk_read_time_seconds_total{device="vda"} 19.128
node_disk_read_time_seconds_total{device="vdb"} 0.039
node_disk_reads_completed_total{device="vda"} 5619
node_disk_reads_completed_total{device="vdb"} 96
node_disk_reads_merged_total{device="vda"} 5
node_disk_reads_merged_total{device="vdb"} 0
node_disk_write_time_seconds_total{device="vda"} 240.66400000000002
node_disk_write_time_seconds_total{device="vdb"} 0
node_disk_writes_completed_total{device="vda"} 71584
node_disk_writes_completed_total{device="vdb"} 0
node_disk_writes_merged_total{device="vda"} 19761
node_disk_writes_merged_total{device="vdb"} 0
node_disk_written_bytes_total{device="vda"} 2.007924224e+09
node_disk_written_bytes_total{device="vdb"} 0

```

13.12.4. Creating a ServiceMonitor resource for the node exporter service

You can use a Prometheus client library and scrape metrics from the **/metrics** endpoint to access and view the metrics exposed by the node-exporter service. Use a **ServiceMonitor** custom resource definition (CRD) to monitor the node exporter service.

Prerequisites

- You have access to the cluster as a user with **cluster-admin** privileges or the **monitoring-edit** role.
- You have enabled monitoring for the user-defined project by configuring the node-exporter service.

Procedure

1. Create a YAML file for the **ServiceMonitor** resource configuration. In this example, the service monitor matches any service with the label **metrics** and queries the **exmet** port every 30 seconds.

```

apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    k8s-app: node-exporter-metrics-monitor
    name: node-exporter-metrics-monitor 1
    namespace: dynamation 2
spec:
  endpoints:

```

```
- interval: 30s
  port: exmet
  scheme: http
  selector:
    matchLabels:
      servicetype: metrics
```

- 1 The name of the **ServiceMonitor**.
- 2 The namespace where the **ServiceMonitor** is created.
- 3 The interval at which the port will be queried.
- 4 The name of the port that is queried every 30 seconds

2. Create the **ServiceMonitor** configuration for the node-exporter service.

```
$ oc create -f node-exporter-metrics-monitor.yaml
```

13.12.4.1. Accessing the node exporter service outside the cluster

You can access the node-exporter service outside the cluster and view the exposed metrics.

Prerequisites

- You have access to the cluster as a user with **cluster-admin** privileges or the **monitoring-edit** role.
- You have enabled monitoring for the user-defined project by configuring the node-exporter service.

Procedure

1. Expose the node-exporter service.

```
$ oc expose service -n <namespace> <node_exporter_service_name>
```

2. Obtain the FQDN (Fully Qualified Domain Name) for the route.

```
$ oc get route -o=custom-columns=NAME:.metadata.name,DNS:.spec.host
```

Example output

```
NAME          DNS
node-exporter-service  node-exporter-service-dynamation.apps.cluster.example.org
```

3. Use the **curl** command to display metrics for the node-exporter service.

```
$ curl -s http://node-exporter-service-dynamation.apps.cluster.example.org/metrics
```

Example output

```
go_gc_duration_seconds{quantile="0"} 1.5382e-05
go_gc_duration_seconds{quantile="0.25"} 3.1163e-05
go_gc_duration_seconds{quantile="0.5"} 3.8546e-05
go_gc_duration_seconds{quantile="0.75"} 4.9139e-05
go_gc_duration_seconds{quantile="1"} 0.000189423
```

13.12.5. Additional resources

- [Configuring the monitoring stack](#)
- [Enabling monitoring for user-defined projects](#)
- [Managing metrics](#)
- [Reviewing monitoring dashboards](#)
- [Monitoring application health by using health checks](#)
- [Creating and using config maps](#)
- [Controlling virtual machine states](#)

13.13. OPENSIFT VIRTUALIZATION CRITICAL ALERTS

OpenShift Virtualization has alerts that inform you when a problem occurs. Critical alerts require immediate attention.

Each alert has a corresponding description of the problem, a reason for why the alert is occurring, a troubleshooting process to diagnose the source of the problem, and steps for resolving the alert.

13.13.1. Network alerts

Network alerts provide information about problems for the OpenShift Virtualization Network Operator.

13.13.1.1. KubeMacPoolDown alert

Description

The KubeMacPool component allocates MAC addresses and prevents MAC address conflicts.

Reason

If the KubeMacPool-manager pod is down, then the creation of **VirtualMachine** objects fails.

Troubleshoot

1. Determine the Kubemacpool-manager pod namespace and name.

```
$ export KMP_NAMESPACE="$(oc get pod -A --no-headers -l control-plane=mac-controller-
manager | awk '{print $1}')
```

```
$ export KMP_NAME="$(oc get pod -A --no-headers -l control-plane=mac-controller-
manager | awk '{print $2}')
```

2. Check the Kubemacpool-manager pod description and logs to determine the source of the problem.

```
$ oc describe pod -n $KMP_NAMESPACE $KMP_NAME
```

```
$ oc logs -n $KMP_NAMESPACE $KMP_NAME
```

Resolution

Open a support issue and provide the information gathered in the troubleshooting process.

13.13.2. SSP alerts

SSP alerts provide information about problems for the OpenShift Virtualization SSP Operator.

13.13.2.1. SSPFailingToReconcile alert

Description

The SSP Operator's pod is up, but the pod's reconcile cycle consistently fails. This failure includes failure to update the resources for which it is responsible, failure to deploy the template validator, or failure to deploy or update the common templates.

Reason

If the SSP Operator fails to reconcile, then the deployment of dependent components fails, reconciliation of component changes fails, or both. Additionally, the updates to the common templates and template validator reset and fail.

Troubleshoot

1. Check the ssp-operator pod's logs for errors:

```
$ export NAMESPACE="$(oc get deployment -A | grep ssp-operator | awk '{print $1}')
```

```
$ oc -n $NAMESPACE describe pods -l control-plane=ssp-operator
```

```
$ oc -n $NAMESPACE logs --tail=-1 -l control-plane=ssp-operator
```

2. Verify that the template validator is up. If the template validator is not up, then check the pod's logs for errors.

```
$ export NAMESPACE="$(oc get deployment -A | grep ssp-operator | awk '{print $1}')
```

```
$ oc -n $NAMESPACE get pods -l name=virt-template-validator
```

```
$ oc -n $NAMESPACE describe pods -l name=virt-template-validator
```

```
$ oc -n $NAMESPACE logs --tail=-1 -l name=virt-template-validator
```

Resolution

Open a support issue and provide the information gathered in the troubleshooting process.

13.13.2.2. SSPOperatorDown alert

Description

The SSP Operator deploys and reconciles the common templates and the template validator.

Reason

If the SSP Operator is down, then the deployment of dependent components fails, reconciliation of component changes fails, or both. Additionally, the updates to the common template and template validator reset and fail.

Troubleshoot

1. Check ssp-operator's pod namespace:

```
$ export NAMESPACE="$(oc get deployment -A | grep ssp-operator | awk '{print $1}')
```

2. Verify that the ssp-operator's pod is currently down.

```
$ oc -n $NAMESPACE get pods -l control-plane=ssp-operator
```

3. Check the ssp-operator's pod description and logs.

```
$ oc -n $NAMESPACE describe pods -l control-plane=ssp-operator
```

```
$ oc -n $NAMESPACE logs --tail=-1 -l control-plane=ssp-operator
```

Resolution

Open a support issue and provide the information gathered in the troubleshooting process.

13.13.2.3. SSPTemplateValidatorDown alert

Description

The template validator validates that virtual machines (VMs) do not violate their assigned templates.

Reason

If every template validator pod is down, then the template validator fails to validate VMs against their assigned templates.

Troubleshoot

1. Check the namespaces of the ssp-operator pods and the virt-template-validator pods.

```
$ export NAMESPACE_SSP="$(oc get deployment -A | grep ssp-operator | awk '{print $1}')
```

```
$ export NAMESPACE="$(oc get deployment -A | grep virt-template-validator | awk '{print $1}')
```

2. Verify that the virt-template-validator's pod is currently down.

```
$ oc -n $NAMESPACE get pods -l name=virt-template-validator
```

3. Check the pod description and logs of the ssp-operator and the virt-template-validator.

```
$ oc -n $NAMESPACE_SSP describe pods -l name=ssp-operator
```

```
$ oc -n $NAMESPACE_SSP logs --tail=-1 -l name=ssp-operator
```

```
$ oc -n $NAMESPACE describe pods -l name=virt-template-validator
```

```
$ oc -n $NAMESPACE logs --tail=-1 -l name=virt-template-validator
```

Resolution

Open a support issue and provide the information gathered in the troubleshooting process.

13.13.3. Virt alerts

Virt alerts provide information about problems for the OpenShift Virtualization Virt Operator.

13.13.3.1. NoLeadingVirtOperator alert

Description

In the past 10 minutes, no virt-operator pod holds the leader lease, despite one or more virt-operator pods being in **Ready** state. The alert suggests no operating virt-operator pod exists.

Reason

The virt-operator is the first Kubernetes Operator active in a OpenShift Container Platform cluster. Its primary responsibilities are:

- Installation
- Live-update
- Live-upgrade of a cluster
- Monitoring the lifecycle of top-level controllers such as virt-controller, virt-handler, and virt-launcher
- Managing the reconciliation of top-level controllers

In addition, the virt-operator is responsible for cluster-wide tasks such as certificate rotation and some infrastructure management.

The virt-operator deployment has a default replica of two pods with one leader pod holding a leader lease, indicating an operating virt-operator pod.

This alert indicates a failure at the cluster level. Critical cluster-wide management functionalities such as certification rotation, upgrade, and reconciliation of controllers may be temporarily unavailable.

Troubleshoot

Determine a virt-operator pod's leader status from the pod logs. The log messages containing **Started leading** and **acquire leader** indicate the leader status of a given virt-operator pod.

Additionally, always check if there are any running virt-operator pods and the pods' statuses with these commands:

```
$ export NAMESPACE="$(oc get kubevirt -A -o custom-columns='':.metadata.namespace)"
```

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-operator
```

```
$ oc -n $NAMESPACE logs <pod-name>
```

```
$ oc -n $NAMESPACE describe pod <pod-name>
```

Leader pod example:

```
$ oc -n $NAMESPACE logs <pod-name> |grep lead
```

Example output

```
{"component":"virt-operator","level":"info","msg":"Attempting to acquire leader
status","pos":"application.go:400","timestamp":"2021-11-30T12:15:18.635387Z"}
I1130 12:15:18.635452    1 leaderelection.go:243] attempting to acquire leader lease
<namespace>/virt-operator...
I1130 12:15:19.216582    1 leaderelection.go:253] successfully acquired lease <namespace>/virt-
operator
```

```
{"component":"virt-operator","level":"info","msg":"Started
leading","pos":"application.go:385","timestamp":"2021-11-30T12:15:19.216836Z"}
```

Non-leader pod example:

```
$ oc -n $NAMESPACE logs <pod-name> |grep lead
```

Example output

```
{"component":"virt-operator","level":"info","msg":"Attempting to acquire leader
status","pos":"application.go:400","timestamp":"2021-11-30T12:15:20.533696Z"}
I1130 12:15:20.533792    1 leaderelection.go:243] attempting to acquire leader lease
<namespace>/virt-operator...
```

Resolution

There are several reasons for no virt-operator pod holding the leader lease, despite one or more virt-operator pods being in **Ready** state. Identify the root cause and take appropriate action.

Otherwise, open a support issue and provide the information gathered in the troubleshooting process.

13.13.3.2. NoReadyVirtController alert

Description

The virt-controller monitors virtual machine instances (VMIs). The virt-controller also manages the associated pods by creating and managing the lifecycle of the pods associated with the VMI objects.

A VMI object always associates with a pod during its lifetime. However, the pod instance can change over time because of VMI migration.

This alert occurs when detection of no ready virt-controllers occurs for five minutes.

Reason

If the virt-controller fails, then VM lifecycle management completely fails. Lifecycle management tasks include launching a new VMI or shutting down an existing VMI.

Troubleshoot

1. Check the vdeployment status of the virt-controller for available replicas and conditions.

```
$ oc -n $NAMESPACE get deployment virt-controller -o yaml
```

2. Check if the virt-controller pods exist and check their statuses.

```
get pods -n $NAMESPACE |grep virt-controller
```

3. Check the virt-controller pods' events.

```
$ oc -n $NAMESPACE describe pods <virt-controller pod>
```

4. Check the virt-controller pods' logs.

```
$ oc -n $NAMESPACE logs <virt-controller pod>
```

5. Check if there are issues with the nodes, such as if the nodes are in a **NotReady** state.

```
$ oc get nodes
```

Resolution

There are several reasons for no virt-controller pods being in a **Ready** state. Identify the root cause and take appropriate action.

Otherwise, open a support issue and provide the information gathered in the troubleshooting process.

13.13.3.3. NoReadyVirtOperator alert

Description

No detection of a virt-operator pod in the **Ready** state occurs in the past 10 minutes. The virt-operator deployment has a default replica of two pods.

Reason

The virt-operator is the first Kubernetes Operator active in an OpenShift Container Platform cluster. Its primary responsibilities are:

- Installation

- Live-update
- Live-upgrade of a cluster
- Monitoring the lifecycle of top-level controllers such as virt-controller, virt-handler, and virt-launcher
- Managing the reconciliation of top-level controllers

In addition, the virt-operator is responsible for cluster-wide tasks such as certificate rotation and some infrastructure management.



NOTE

Virt-operator is not directly responsible for virtual machines in the cluster. Virt-operator's unavailability does not affect the custom workloads.

This alert indicates a failure at the cluster level. Critical cluster-wide management functionalities such as certification rotation, upgrade, and reconciliation of controllers are temporarily unavailable.

Troubleshoot

1. Check the deployment status of the virt-operator for available replicas and conditions.

```
$ oc -n $NAMESPACE get deployment virt-operator -o yaml
```

2. Check the virt-controller pods' events.

```
$ oc -n $NAMESPACE describe pods <virt-operator pod>
```

3. Check the virt-operator pods' logs.

```
$ oc -n $NAMESPACE logs <virt-operator pod>
```

4. Check if there are issues with the nodes for the control plane and masters, such as if they are in a **NotReady** state.

```
$ oc get nodes
```

Resolution

There are several reasons for no virt-operator pods being in a **Ready** state. Identify the root cause and take appropriate action.

Otherwise, open a support issue and provide the information gathered in the troubleshooting process.

13.13.3.4. VirtAPIDown alert

Description

All OpenShift Container Platform API servers are down.

Reason

If all OpenShift Container Platform API servers are down, then no API calls for OpenShift Container Platform entities occur.

Troubleshoot

1. Modify the environment variable **NAMESPACE**.

```
$ export NAMESPACE="$(oc get kubevirt -A -o custom-columns='':.metadata.namespace)"
```

2. Verify if there are any running virt-api pods.

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-api
```

3. View the pods' logs using **oc logs** and the pods' statuses using **oc describe**.
4. Check the status of the virt-api deployment. Use these commands to learn about related events and show if there are any issues with pulling an image, a crashing pod, or other similar problems.

```
$ oc -n $NAMESPACE get deployment virt-api -o yaml
```

```
$ oc -n $NAMESPACE describe deployment virt-api
```

5. Check if there are issues with the nodes, such as if the nodes are in a **NotReady** state.

```
$ oc get nodes
```

Resolution

Virt-api pods can be down for several reasons. Identify the root cause and take appropriate action.

Otherwise, open a support issue and provide the information gathered in the troubleshooting process.

13.13.3.5. VirtApiRESTErrorsBurst alert

Description

More than 80% of the REST calls fail in virt-api in the last five minutes.

Reason

A very high rate of failed REST calls to virt-api causes slow response, slow execution of API calls, or even complete dismissal of API calls.

Troubleshoot

1. Modify the environment variable **NAMESPACE**.

```
$ export NAMESPACE="$(oc get kubevirt -A -o custom-columns='':.metadata.namespace)"
```

2. Check to see how many running virt-api pods exist.

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-api
```

3. View the pods' logs using **oc logs** and the pods' statuses using **oc describe**.

4. Check the status of the virt-api deployment to find out more information. These commands provide the associated events and show if there are any issues with pulling an image or a crashing pod.

```
$ oc -n $NAMESPACE get deployment virt-api -o yaml
```

```
$ oc -n $NAMESPACE describe deployment virt-api
```

5. Check if there are issues with the nodes, such as if the nodes are overloaded or not in a **NotReady** state.

```
$ oc get nodes
```

Resolution

There are several reasons for a high rate of failed REST calls. Identify the root cause and take appropriate action.

- Node resource exhaustion
- Not enough memory on the cluster
- Nodes are down
- The API server overloads, such as when the scheduler is not 100% available)
- Networking issues

Otherwise, open a support issue and provide the information gathered in the troubleshooting process.

13.13.3.6. VirtControllerDown alert

Description

If no detection of virt-controllers occurs in the past five minutes, then virt-controller deployment has a default replica of two pods.

Reason

If the virt-controller fails, then VM lifecycle management tasks, such as launching a new VMI or shutting down an existing VMI, completely fail.

Troubleshoot

1. Modify the environment variable **NAMESPACE**.

```
$ export NAMESPACE="$(oc get kubevirt -A -o custom-columns='":.metadata.namespace)"
```

2. Check the status of the virt-controller deployment.

```
$ oc get deployment -n $NAMESPACE virt-controller -o yaml
```

3. Check the virt-controller pods' events.

```
$ oc -n $NAMESPACE describe pods <virt-controller pod>
```

4. Check the virt-controller pods' logs.

```
$ oc -n $NAMESPACE logs <virt-controller pod>
```

5. Check the manager pod's logs to determine why creating the virt-controller pods fails.

```
$ oc get logs <virt-controller-pod>
```

An example of a virt-controller pod name in the logs is **virt-controller-7888c64d66-dzc9p**. However, there may be several pods that run virt-controller.

Resolution

There are several known reasons why the detection of no running virt-controller occurs. Identify the root cause from the list of possible reasons and take appropriate action.

- Node resource exhaustion
- Not enough memory on the cluster
- Nodes are down
- The API server overloads, such as when the scheduler is not 100% available)
- Networking issues

Otherwise, open a support issue and provide the information gathered in the troubleshooting process.

13.13.3.7. VirtControllerRESTErrorsBurst alert

Description

More than 80% of the REST calls failed in virt-controller in the last five minutes.

Reason

Virt-controller has potentially fully lost connectivity to the API server. This loss does not affect running workloads, but propagation of status updates and actions like migrations cannot occur.

Troubleshoot

There are two common error types associated with virt-controller REST call failure:

- The API server overloads, causing timeouts. Check the API server metrics and details like response times and overall calls.
- The virt-controller pod cannot reach the API server. Common causes are:
 - DNS issues on the node
 - Networking connectivity issues

Resolution

Check the virt-controller logs to determine if the virt-controller pod cannot connect to the API server at all. If so, delete the pod to force a restart.

Additionally, verify if node resource exhaustion or not having enough memory on the cluster is causing the connection failure.

The issue normally relates to DNS or CNI issues outside of the scope of this alert.

Otherwise, open a support issue and provide the information gathered in the troubleshooting process.

13.13.3.8. VirtHandlerRESTErrorsBurst alert

Description

More than 80% of the REST calls failed in virt-handler in the last five minutes.

Reason

Virt-handler lost the connection to the API server. Running workloads on the affected node still run, but status updates cannot propagate and actions such as migrations cannot occur.

Troubleshoot

There are two common error types associated with virt-operator REST call failure:

- The API server overloads, causing timeouts. Check the API server metrics and details like response times and overall calls.
- The virt-operator pod cannot reach the API server. Common causes are:
 - DNS issues on the node
 - Networking connectivity issues

Resolution

If the virt-handler cannot connect to the API server, delete the pod to force a restart. The issue normally relates to DNS or CNI issues outside of the scope of this alert. Identify the root cause and take appropriate action.

Otherwise, open a support issue and provide the information gathered in the troubleshooting process.

13.13.3.9. VirtOperatorDown alert

Description

This alert occurs when no virt-operator pod is in the **Running** state in the past 10 minutes. The virt-operator deployment has a default replica of two pods.

Reason

The virt-operator is the first Kubernetes Operator active in an OpenShift Container Platform cluster. Its primary responsibilities are:

- Installation
- Live-update
- Live-upgrade of a cluster
- Monitoring the lifecycle of top-level controllers such as virt-controller, virt-handler, and virt-launcher

- Managing the reconciliation of top-level controllers

In addition, the virt-operator is responsible for cluster-wide tasks such as certificate rotation and some infrastructure management.



NOTE

The virt-operator is not directly responsible for virtual machines in the cluster. The virt-operator's unavailability does not affect the custom workloads.

This alert indicates a failure at the cluster level. Critical cluster-wide management functionalities such as certification rotation, upgrade, and reconciliation of controllers are temporarily unavailable.

Troubleshoot

1. Modify the environment variable **NAMESPACE**.

```
$ export NAMESPACE="$(oc get kubevirt -A -o custom-columns='\"':.metadata.namespace)\""
```

2. Check the status of the virt-operator deployment.

```
$ oc get deployment -n $NAMESPACE virt-operator -o yaml
```

3. Check the virt-operator pods' events.

```
$ oc -n $NAMESPACE describe pods <virt-operator pod>
```

4. Check the virt-operator pods' logs.

```
$ oc -n $NAMESPACE logs <virt-operator pod>
```

5. Check the manager pod's logs to determine why creating the virt-operator pods fails.

```
$ oc get logs <virt-operator-pod>
```

An example of a virt-operator pod name in the logs is **virt-operator-7888c64d66-dzc9p**. However, there may be several pods that run virt-operator.

Resolution

There are several known reasons why the detection of no running virt-operator occurs. Identify the root cause from the list of possible reasons and take appropriate action.

- Node resource exhaustion
- Not enough memory on the cluster
- Nodes are down
- The API server overloads, such as when the scheduler is not 100% available)
- Networking issues

Otherwise, open a support issue and provide the information gathered in the troubleshooting process.

13.13.3.10. VirtOperatorRESTErrorsBurst alert

Description

More than 80% of the REST calls failed in virt-operator in the last five minutes.

Reason

Virt-operator lost the connection to the API server. Cluster-level actions such as upgrading and controller reconciliation do not function. There is no effect to customer workloads such as VMs and VMIs.

Troubleshoot

There are two common error types associated with virt-operator REST call failure:

- The API server overloads, causing timeouts. Check the API server metrics and details, such as response times and overall calls.
- The virt-operator pod cannot reach the API server. Common causes are network connectivity problems and DNS issues on the node. Check the virt-operator logs to verify that the pod can connect to the API server at all.

```
$ export NAMESPACE="$(oc get kubevirt -A -o custom-columns='":.metadata.namespace")"
```

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-operator
```

```
$ oc -n $NAMESPACE logs <pod-name>
```

```
$ oc -n $NAMESPACE describe pod <pod-name>
```

Resolution

If the virt-operator cannot connect to the API server, delete the pod to force a restart. The issue normally relates to DNS or CNI issues outside of the scope of this alert. Identify the root cause and take appropriate action.

Otherwise, open a support issue and provide the information gathered in the troubleshooting process.

13.13.4. Additional resources

- [Getting support](#)

13.14. 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.

13.14.1. 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 Data Foundation, 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.

13.14.1.1. 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 [Red Hat OpenShift Data Foundation](#)
- Collecting **must-gather** data for [OpenShift Virtualization](#)
- Collecting Prometheus metrics for [all projects](#) as a cluster administrator

13.14.2. 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.

13.14.2.1. 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](#)

13.14.3. 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}
```

13.14.3.1. 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

13.14.3.1.1. 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.

13.14.3.1.2. 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
--------	---------------------------------

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:v4.11.0 \
  -- <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:v4.11.0 \
  -- NS=mynamespace VM=my-vm gather_vms_details ❶
```

❶ 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:v4.11.0 \
  -- 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:v4.11.0 \  
-- gather_images
```

13.14.3.2. Additional resources

- [About the **must-gather** tool](#)

CHAPTER 14. BACKUP AND RESTORE

14.1. INSTALLING AND CONFIGURING OADP

As a cluster administrator, you install the OpenShift API for Data Protection (OADP) by installing the OADP Operator. The Operator installs [Velero 1.9](#).

You create a default **Secret** for your backup storage provider and then you install the Data Protection Application.

14.1.1. Installing the OADP Operator

You install the OpenShift API for Data Protection (OADP) Operator on OpenShift Container Platform 4.11 by using Operator Lifecycle Manager (OLM).

The OADP Operator installs [Velero 1.9](#).

Prerequisites

- You must be logged in as a user with **cluster-admin** privileges.

Procedure

1. In the OpenShift Container Platform web console, click **Operators → OperatorHub**.
2. Use the **Filter by keyword** field to find the **OADP Operator**.
3. Select the **OADP Operator** and click **Install**.
4. Click **Install** to install the Operator in the **openshift-adp** project.
5. Click **Operators → Installed Operators** to verify the installation.

14.1.2. About backup and snapshot locations and their secrets

You specify backup and snapshot locations and their secrets in the **DataProtectionApplication** custom resource (CR).

Backup locations

You specify S3-compatible object storage, such as Multicloud Object Gateway, Noobaa, or Minio, as a backup location.

Velero backs up OpenShift Container Platform resources, Kubernetes objects, and internal images as an archive file on object storage.

Snapshot locations

If you use your cloud provider's native snapshot API to back up persistent volumes, you must specify the cloud provider as the snapshot location.

If you use Container Storage Interface (CSI) snapshots, you do not need to specify a snapshot location because you will create a **VolumeSnapshotClass** CR to register the CSI driver.

If you use Restic, you do not need to specify a snapshot location because Restic backs up the file system on object storage.

Secrets

If the backup and snapshot locations use the same credentials or if you do not require a snapshot location, you create a default **Secret**.

If the backup and snapshot locations use different credentials, you create two secret objects:

- Custom **Secret** for the backup location, which you specify in the **DataProtectionApplication** CR.
- Default **Secret** for the snapshot location, which is not referenced in the **DataProtectionApplication** CR.



IMPORTANT

The Data Protection Application requires a default **Secret**. Otherwise, the installation will fail.

If you do not want to specify backup or snapshot locations during the installation, you can create a default **Secret** with an empty **credentials-velero** file.

14.1.2.1. Creating a default Secret

You create a default **Secret** if your backup and snapshot locations use the same credentials or if you do not require a snapshot location.



NOTE

The **DataProtectionApplication** custom resource (CR) requires a default **Secret**. Otherwise, the installation will fail. If the name of the backup location **Secret** is not specified, the default name is used.

If you do not want to use the backup location credentials during the installation, you can create a **Secret** with the default name by using an empty **credentials-velero** file.

Prerequisites

- Your object storage and cloud storage, if any, must use the same credentials.
- You must configure object storage for Velero.
- You must create a **credentials-velero** file for the object storage in the appropriate format.

Procedure

- Create a **Secret** with the default name:

```
$ oc create secret generic cloud-credentials -n openshift-adp --from-file cloud=credentials-velero
```

The **Secret** is referenced in the **spec.backupLocations.credential** block of the **DataProtectionApplication** CR when you install the Data Protection Application.

14.1.3. Configuring the Data Protection Application

You can configure the Data Protection Application by setting Velero resource allocations or enabling self-signed CA certificates.

14.1.3.1. Setting Velero CPU and memory resource allocations

You set the CPU and memory resource allocations for the **Velero** pod by editing the **DataProtectionApplication** custom resource (CR) manifest.

Prerequisites

- You must have the OpenShift API for Data Protection (OADP) Operator installed.

Procedure

- Edit the values in the **spec.configuration.velero.podConfig.ResourceAllocations** block of the **DataProtectionApplication** CR manifest, as in the following example:

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
spec:
  ...
  configuration:
    velero:
      podConfig:
        nodeSelector: <node selector> 1
        resourceAllocations:
          limits:
            cpu: "1"
            memory: 512Mi
          requests:
            cpu: 500m
            memory: 256Mi
```

- 1** Specify the node selector to be supplied to Velero podSpec

14.1.3.2. Enabling self-signed CA certificates

You must enable a self-signed CA certificate for object storage by editing the **DataProtectionApplication** custom resource (CR) manifest to prevent a **certificate signed by unknown authority** error.

Prerequisites

- You must have the OpenShift API for Data Protection (OADP) Operator installed.

Procedure

- Edit the **spec.backupLocations.velero.objectStorage.caCert** parameter and **spec.backupLocations.velero.config** parameters of the **DataProtectionApplication** CR manifest:


```

apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
spec:
  ...
  backupLocations:
    - name: default
      velero:
        provider: aws
        default: true
        objectStorage:
          bucket: <bucket>
          prefix: <prefix>
          caCert: <base64_encoded_cert_string> ❶
        config:
          insecureSkipTLSVerify: "false" ❷
  ...

```

- ❶ Specify the Base46-encoded CA certificate string.
- ❷ The **`insecureSkipTLSVerify`** configuration can be set to either **"true"** or **"false"**. If set to **"true"**, SSL/TLS security is disabled. If set to **"false"**, SSL/TLS security is enabled.

14.1.4. Installing the Data Protection Application

You install the Data Protection Application (DPA) by creating an instance of the **DataProtectionApplication** API.

Prerequisites

- You must install the OADP Operator.
- You must configure object storage as a backup location.
- If you use snapshots to back up PVs, your cloud provider must support either a native snapshot API or Container Storage Interface (CSI) snapshots.
- If the backup and snapshot locations use the same credentials, you must create a **Secret** with the default name, **cloud-credentials**.
- If the backup and snapshot locations use different credentials, you must create two **Secrets**:
 - **Secret** with a custom name for the backup location. You add this **Secret** to the **DataProtectionApplication** CR.
 - **Secret** with the default name, **cloud-credentials**, for the snapshot location. This **Secret** is not referenced in the **DataProtectionApplication** CR.



NOTE

If you do not want to specify backup or snapshot locations during the installation, you can create a default **Secret** with an empty **credentials-velero** file. If there is no default **Secret**, the installation will fail.

Procedure

1. Click **Operators** → **Installed Operators** and select the OADP Operator.
2. Under **Provided APIs**, click **Create instance** in the **DataProtectionApplication** box.
3. Click **YAML View** and update the parameters of the **DataProtectionApplication** manifest:

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
  namespace: openshift-adp
spec:
  configuration:
    velero:
      defaultPlugins:
        - kubevirt 1
        - gcp 2
        - csi 3
        - openshift 4
      restic:
        enable: true 5
      podConfig:
        nodeSelector: <node selector> 6
  backupLocations:
    - velero:
        provider: gcp 7
        default: true
        credential:
          key: cloud
          name: <default_secret> 8
        objectStorage:
          bucket: <bucket_name> 9
          prefix: <prefix> 10
```

- 1** The **kubevirt** plug-in is mandatory for OpenShift Virtualization.
- 2** Specify the plug-in for the backup provider, for example, **gcp**, if it exists.
- 3** The **csi** plug-in is mandatory for backing up PVs with CSI snapshots. The **csi** plug-in uses the [Velero CSI beta snapshot APIs](#). You do not need to configure a snapshot location.
- 4** The **openshift** plug-in is mandatory.
- 5** Set to **false** if you want to disable the Restic installation. Restic deploys a daemon set, which means that each worker node has **Restic** pods running. You configure Restic for backups by adding **spec.defaultVolumesToRestic: true** to the **Backup** CR.
- 6** Specify the node selector to be supplied to Restic podSpec
- 7** Specify the backup provider.
- 8** If you use a default plug-in for the backup provider, you must specify the correct default name for the **Secret**, for example, **cloud-credentials-gcp**. If you specify a custom name, the custom name is used for the backup location. If you do not specify a **Secret** name, the

default name is used.

- 9 Specify a bucket as the backup storage location. If the bucket is not a dedicated bucket for Velero backups, you must specify a prefix.
- 10 Specify a prefix for Velero backups, for example, **velero**, if the bucket is used for multiple purposes.

4. Click **Create**.

5. Verify the installation by viewing the OADP resources:

```
$ oc get all -n openshift-adp
```

Example output

```
NAME                                READY STATUS  RESTARTS  AGE
pod/oadp-operator-controller-manager-67d9494d47-6l8z8  2/2   Running  0         2m8s
pod/oadp-velero-sample-1-aws-registry-5d6968cbdd-d5w9k  1/1   Running  0         95s
pod/restic-9cq4q                                1/1   Running  0         94s
pod/restic-m4lts                                1/1   Running  0         94s
pod/restic-pv4kr                                1/1   Running  0         95s
pod/velero-588db7f655-n842v                    1/1   Running  0         95s

NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP
PORT(S)  AGE
service/oadp-operator-controller-manager-metrics-service  ClusterIP    172.30.70.140
<none>    8443/TCP  2m8s
service/oadp-velero-sample-1-aws-registry-svc              ClusterIP    172.30.130.230 <none>
5000/TCP  95s

NAME            DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE  NODE
SELECTOR  AGE
daemonset.apps/restic  3        3        3      3           3          <none>    96s

NAME            READY  UP-TO-DATE  AVAILABLE  AGE
deployment.apps/oadp-operator-controller-manager  1/1    1           1          2m9s
deployment.apps/oadp-velero-sample-1-aws-registry  1/1    1           1          96s
deployment.apps/velero                            1/1    1           1          96s

NAME            DESIRED  CURRENT  READY  AGE
replicaset.apps/oadp-operator-controller-manager-67d9494d47  1      1      1      2m9s
replicaset.apps/oadp-velero-sample-1-aws-registry-5d6968cbdd  1      1      1      96s
replicaset.apps/velero-588db7f655                            1      1      1      96s
```

14.1.4.1. Enabling CSI in the DataProtectionApplication CR

You enable the Container Storage Interface (CSI) in the **DataProtectionApplication** custom resource (CR) in order to back up persistent volumes with CSI snapshots.

Prerequisites

- The cloud provider must support CSI snapshots.

Procedure

- Edit the **DataProtectionApplication** CR, as in the following example:

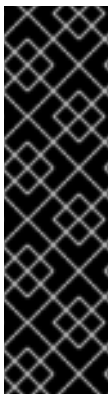
```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
...
spec:
  configuration:
    velero:
      defaultPlugins:
        - openshift
        - csi 1
      featureFlags:
        - EnableCSI 2
```

- 1 Add the **csi** default plug-in.
- 2 Add the **EnableCSI** feature flag.

14.1.5. Uninstalling OADP

You uninstall the OpenShift API for Data Protection (OADP) by deleting the OADP Operator. See [Deleting Operators from a cluster](#) for details.

14.2. BACKING UP AND RESTORING VIRTUAL MACHINES



IMPORTANT

OADP for OpenShift Virtualization 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/>.

You back up and restore virtual machines by using the [OpenShift API for Data Protection \(OADP\)](#).

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Install the [OADP Operator](#) according to the instructions for your storage provider.
2. Install the [Data Protection Application](#) with the **kubevirt** and **openshift** plugins.
3. Back up virtual machines by creating a [Backup](#) custom resource (CR).

4. Restore the **Backup** CR by creating a [Restore CR](#).

14.2.1. Additional resources

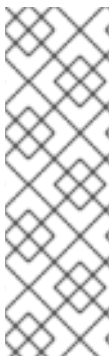
- [OADP features and plugins](#)
- [Troubleshooting](#)

14.3. BACKING UP VIRTUAL MACHINES

You back up virtual machines (VMs) by creating an OpenShift API for Data Protection (OADP) **Backup custom resource (CR)**.

The **Backup** CR performs the following actions:

- Backs up OpenShift Virtualization resources by creating an archive file on S3-compatible object storage, such as [Multicloud Object Gateway](#), Noobaa, or Minio.
- Backs up VM disks by using one of the following options:
 - [Container Storage Interface \(CSI\) snapshots](#) on CSI-enabled cloud storage, such as Ceph RBD or Ceph FS.
 - [Restic file system backups](#) on object storage.



NOTE

OADP provides backup hooks to freeze the VM file system before the backup operation and unfreeze it when the backup is complete.

The **kubevirt-controller** creates the **virt-launcher** pods with annotations that enable Velero to run the **virt-freezer** binary before and after the backup operation.

The **freeze** and **unfreeze** APIs are subresources of the VM snapshot API. See [About virtual machine snapshots](#) for details.

You can add [hooks](#) to the **Backup** CR to run commands on specific VMs before or after the backup operation.

You schedule a backup by creating a [Schedule CR](#) instead of a **Backup** CR.

14.3.1. Creating a Backup CR

You back up Kubernetes images, internal images, and persistent volumes (PVs) by creating a **Backup** custom resource (CR).

Prerequisites

- You must install the OpenShift API for Data Protection (OADP) Operator.
- The **DataProtectionApplication** CR must be in a **Ready** state.
- Backup location prerequisites:
 - You must have S3 object storage configured for Velero.

- You must have a backup location configured in the **DataProtectionApplication** CR.
- Snapshot location prerequisites:
 - Your cloud provider must have a native snapshot API or support Container Storage Interface (CSI) snapshots.
 - For CSI snapshots, you must create a **VolumeSnapshotClass** CR to register the CSI driver.
 - You must have a volume location configured in the **DataProtectionApplication** CR.

Procedure

1. Retrieve the **backupStorageLocations** CRs by entering the following command:

```
$ oc get backupStorageLocations
```

Example output

```
NAME          PHASE    LAST VALIDATED  AGE   DEFAULT
velero-sample-1 Available  11s            31m
```

2. Create a **Backup** CR, as in the following example:

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: <backup>
  labels:
    velero.io/storage-location: default
  namespace: openshift-adp
spec:
  hooks: {}
  includedNamespaces:
    - <namespace> 1
  includedResources: [] 2
  excludedResources: [] 3
  storageLocation: <velero-sample-1> 4
  ttl: 720h0m0s
  labelSelector: 5
    - matchLabels:
        app=<label_1>
    - matchLabels:
        app=<label_2>
    - matchLabels:
        app=<label_3>
  orlabelSelectors: 6
    - matchLabels:
        app=<label_1>
    - matchLabels:
        app=<label_2>
    - matchLabels:
        app=<label_3>
```

- 1 Specify an array of namespaces to back up.
- 2 Optional: Specify an array of resources to include in the backup. Resources might be shortcuts (for example, 'po' for 'pods') or fully-qualified. If unspecified, all resources are included.
- 3 5 Optional: Specify an array of resources to exclude from the backup. Resources might be shortcuts (for example, 'po' for 'pods') or fully-qualified.
- 4 6 Specify the name of the **backupStorageLocations** CR.

3. Verify that the status of the **Backup** CR is **Completed**:

```
$ oc get backup -n openshift-adp <backup> -o jsonpath='{.status.phase}'
```

14.3.1.1. Backing up persistent volumes with CSI snapshots

You back up persistent volumes with Container Storage Interface (CSI) snapshots by editing the **VolumeSnapshotClass** custom resource (CR) of the cloud storage before you create the **Backup** CR.

Prerequisites

- The cloud provider must support CSI snapshots.
- You must enable CSI in the **DataProtectionApplication** CR.

Procedure

- Add the **metadata.labels.velero.io/csi-volumesnapshot-class: "true"** key-value pair to the **VolumeSnapshotClass** CR:

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: <volume_snapshot_class_name>
  labels:
    velero.io/csi-volumesnapshot-class: "true"
driver: <csi_driver>
deletionPolicy: Retain
```

You can now create a **Backup** CR.

14.3.1.2. Backing up applications with Restic

You back up Kubernetes resources, internal images, and persistent volumes with Restic by editing the **Backup** custom resource (CR).

You do not need to specify a snapshot location in the **DataProtectionApplication** CR.



IMPORTANT

Restic does not support backing up **hostPath** volumes. For more information, see [additional Restic limitations](#).

Prerequisites

- You must install the OpenShift API for Data Protection (OADP) Operator.
- You must not disable the default Restic installation by setting **spec.configuration.restic.enable** to **false** in the **DataProtectionApplication** CR.
- The **DataProtectionApplication** CR must be in a **Ready** state.

Procedure

- Edit the **Backup** CR, as in the following example:

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: <backup>
  labels:
    velero.io/storage-location: default
  namespace: openshift-adp
spec:
  defaultVolumesToRestic: true 1
...
```

- 1 Add **defaultVolumesToRestic: true** to the **spec** block.

14.3.1.3. Creating backup hooks

You create backup hooks to run commands in a container in a pod by editing the **Backup** custom resource (CR).

Pre hooks run before the pod is backed up. *Post* hooks run after the backup.

Procedure

- Add a hook to the **spec.hooks** block of the **Backup** CR, as in the following example:

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: <backup>
  namespace: openshift-adp
spec:
  hooks:
    resources:
      - name: <hook_name>
        includedNamespaces:
          - <namespace> 1
        excludedNamespaces: 2
          - <namespace>
        includedResources: []
        - pods 3
        excludedResources: [] 4
      labelSelector: 5
```



```

matchLabels:
  app: velero
  component: server
pre: ❹
- exec:
  container: <container> ❺
  command:
  - /bin/uname ❸
  - -a
  onError: Fail ❽
  timeout: 30s ❿
post: ❾
...

```

- ❶ Optional: You can specify namespaces to which the hook applies. If this value is not specified, the hook applies to all namespaces.
- ❷ Optional: You can specify namespaces to which the hook does not apply.
- ❸ Currently, pods are the only supported resource that hooks can apply to.
- ❹ Optional: You can specify resources to which the hook does not apply.
- ❺ Optional: This hook only applies to objects matching the label. If this value is not specified, the hook applies to all namespaces.
- ❻ Array of hooks to run before the backup.
- ❼ Optional: If the container is not specified, the command runs in the first container in the pod.
- ❽ This is the entrypoint for the init container being added.
- ❾ Allowed values for error handling are **Fail** and **Continue**. The default is **Fail**.
- ❿ Optional: How long to wait for the commands to run. The default is **30s**.
- ⓫ This block defines an array of hooks to run after the backup, with the same parameters as the pre-backup hooks.

14.3.2. Scheduling backups

You schedule backups by creating a **Schedule** custom resource (CR) instead of a **Backup** CR.

**WARNING**

Leave enough time in your backup schedule for a backup to finish before another backup is created.

For example, if a backup of a namespace typically takes 10 minutes, do not schedule backups more frequently than every 15 minutes.

Prerequisites

- You must install the OpenShift API for Data Protection (OADP) Operator.
- The **DataProtectionApplication** CR must be in a **Ready** state.

Procedure

1. Retrieve the **backupStorageLocations** CRs:

```
$ oc get backupStorageLocations
```

Example output

NAME	PHASE	LAST VALIDATED	AGE	DEFAULT
velero-sample-1	Available	11s	31m	

2. Create a **Schedule** CR, as in the following example:

```
$ cat << EOF | oc apply -f -
apiVersion: velero.io/v1
kind: Schedule
metadata:
  name: <schedule>
  namespace: openshift-adp
spec:
  schedule: 0 7 * * * 1
  template:
    hooks: {}
    includedNamespaces:
      - <namespace> 2
    storageLocation: <velero-sample-1> 3
    defaultVolumesToRestic: true 4
    ttl: 720h0m0s
EOF
```

- 1 **cron** expression to schedule the backup, for example, **0 7 * * *** to perform a backup every day at 7:00.
- 2 Array of namespaces to back up.

- 3 Name of the **backupStorageLocations** CR.
- 4 Optional: Add the **defaultVolumesToRestic: true** key-value pair if you are backing up volumes with Restic.

3. Verify that the status of the **Schedule** CR is **Completed** after the scheduled backup runs:

```
$ oc get schedule -n openshift-adp <schedule> -o jsonpath='{.status.phase}'
```

14.3.3. Additional resources

- [Overview of CSI volume snapshots](#)

14.4. RESTORING VIRTUAL MACHINES

You restore an OpenShift API for Data Protection (OADP) **Backup** custom resource (CR) by creating a **Restore CR**.

You can add [hooks](#) to the **Restore** CR to run commands in init containers, before the application container starts, or in the application container itself.

14.4.1. Creating a Restore CR

You restore a **Backup** custom resource (CR) by creating a **Restore** CR.

Prerequisites

- You must install the OpenShift API for Data Protection (OADP) Operator.
- The **DataProtectionApplication** CR must be in a **Ready** state.
- You must have a Velero **Backup** CR.
- Adjust the requested size so the persistent volume (PV) capacity matches the requested size at backup time.

Procedure

1. Create a **Restore** CR, as in the following example:

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: <restore>
  namespace: openshift-adp
spec:
  backupName: <backup> 1
  includedResources: [] 2
  excludedResources:
    - nodes
    - events
    - events.events.k8s.io
    - backups.velero.io
```

```
- restores.velero.io
- resticrepositories.velero.io
restorePVs: true
```

- 1 Name of the **Backup** CR.
- 2 Optional. Specify an array of resources to include in the restore process. Resources might be shortcuts (for example, 'po' for 'pods') or fully-qualified. If unspecified, all resources are included.

2. Verify that the status of the **Restore** CR is **Completed** by entering the following command:

```
$ oc get restore -n openshift-adp <restore> -o jsonpath='{.status.phase}'
```

3. Verify that the backup resources have been restored by entering the following command:

```
$ oc get all -n <namespace> 1
```

- 1 Namespace that you backed up.

14.4.1.1. Creating restore hooks

You create restore hooks to run commands in a container in a pod while restoring your application by editing the **Restore** custom resource (CR).

You can create two types of restore hooks:

- An **init** hook adds an init container to a pod to perform setup tasks before the application container starts.
If you restore a Restic backup, the **restic-wait** init container is added before the restore hook init container.
- An **exec** hook runs commands or scripts in a container of a restored pod.

Procedure

- Add a hook to the **spec.hooks** block of the **Restore** CR, as in the following example:

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: <restore>
  namespace: openshift-adp
spec:
  hooks:
    resources:
      - name: <hook_name>
        includedNamespaces:
          - <namespace> 1
        excludedNamespaces:
          - <namespace>
        includedResources:
          - pods 2
```

```

excludedResources: []
labelSelector: ❸
  matchLabels:
    app: velero
    component: server
postHooks:
- init:
  initContainers:
  - name: restore-hook-init
    image: alpine:latest
    volumeMounts:
    - mountPath: /restores/pvc1-vm
      name: pvc1-vm
    command:
    - /bin/ash
    - -c
  timeout: ❹
- exec:
  container: <container> ❺
  command:
  - /bin/bash ❻
  - -c
  - "psql < /backup/backup.sql"
  waitTimeout: 5m ❼
  execTimeout: 1m ❽
  onError: Continue ❾

```

- ❶ Optional: Array of namespaces to which the hook applies. If this value is not specified, the hook applies to all namespaces.
- ❷ Currently, pods are the only supported resource that hooks can apply to.
- ❸ Optional: This hook only applies to objects matching the label selector.
- ❹ Optional: Timeout specifies the maximum amount of time Velero waits for **initContainers** to complete.
- ❺ Optional: If the container is not specified, the command runs in the first container in the pod.
- ❻ This is the entrypoint for the init container being added.
- ❼ Optional: How long to wait for a container to become ready. This should be long enough for the container to start and for any preceding hooks in the same container to complete. If not set, the restore process waits indefinitely.
- ❽ Optional: How long to wait for the commands to run. The default is **30s**.
- ❾ Allowed values for error handling are **Fail** and **Continue**:
 - **Continue**: Only command failures are logged.
 - **Fail**: No more restore hooks run in any container in any pod. The status of the **Restore** CR will be **PartiallyFailed**.

