



OpenShift Container Platform 4.5

OpenShift Virtualization

OpenShift Virtualization installation, usage, and release notes

OpenShift Container Platform 4.5 OpenShift Virtualization

OpenShift Virtualization installation, usage, and release notes

Legal Notice

Copyright © 2021 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	11
1.1. WHAT YOU CAN DO WITH OPENSIFT VIRTUALIZATION	11
1.1.1. OpenShift Virtualization supported cluster version	11
CHAPTER 2. OPENSIFT VIRTUALIZATION RELEASE NOTES	12
2.1. ABOUT RED HAT OPENSIFT VIRTUALIZATION	12
2.1.1. What you can do with OpenShift Virtualization	12
2.2. NEW AND CHANGED FEATURES	12
2.2.1. Supported guest operating systems	13
2.2.2. Networking	13
2.2.3. Storage	13
2.2.4. Web console	14
2.3. NOTABLE TECHNICAL CHANGES	14
2.4. KNOWN ISSUES	14
CHAPTER 3. OPENSIFT VIRTUALIZATION INSTALLATION	18
3.1. CONFIGURING YOUR CLUSTER FOR OPENSIFT VIRTUALIZATION	18
3.2. INSTALLING OPENSIFT VIRTUALIZATION USING THE WEB CONSOLE	18
3.2.1. Prerequisites	18
3.2.2. Subscribing to the OpenShift Virtualization catalog	18
3.2.3. Deploying OpenShift Virtualization	19
3.2.4. Next steps	20
3.3. INSTALLING OPENSIFT VIRTUALIZATION USING THE CLI	20
3.3.1. Prerequisites	20
3.3.2. Subscribing to the OpenShift Virtualization catalog by using the CLI	20
3.3.3. Deploying the OpenShift Virtualization Operator by using the CLI	21
3.3.4. Next steps	22
3.4. INSTALLING THE VIRTCTL CLIENT	22
3.4.1. Enabling OpenShift Virtualization repositories	22
3.4.2. Installing the virtctl client	23
3.5. UNINSTALLING OPENSIFT VIRTUALIZATION USING THE WEB CONSOLE	23
3.5.1. Prerequisites	23
3.5.2. Deleting the OpenShift Virtualization Operator Deployment custom resource	23
3.5.3. Deleting the OpenShift Virtualization catalog subscription	24
3.5.4. Deleting a namespace using the web console	24
3.6. UNINSTALLING OPENSIFT VIRTUALIZATION USING THE CLI	24
3.6.1. Prerequisites	25
3.6.2. Deleting OpenShift Virtualization	25
CHAPTER 4. UPGRADING OPENSIFT VIRTUALIZATION	27
4.1. ABOUT UPGRADING OPENSIFT VIRTUALIZATION	27
4.1.1. How OpenShift Virtualization upgrades work	27
4.1.2. How OpenShift Virtualization upgrades affect your cluster	27
4.2. UPGRADING OPENSIFT VIRTUALIZATION TO THE NEXT MINOR VERSION	27
4.3. MONITORING UPGRADE STATUS	28
4.4. ADDITIONAL RESOURCES	29
CHAPTER 5. ADDITIONAL SECURITY PRIVILEGES GRANTED FOR KUBEVIRT-CONTROLLER AND VIRT-LAUNCHER	30
5.1. EXTENDED SELINUX POLICIES FOR VIRT-LAUNCHER PODS	30
5.2. ADDITIONAL OPENSIFT CONTAINER PLATFORM SECURITY CONTEXT CONSTRAINTS AND LINUX CAPABILITIES FOR THE KUBEVIRT-CONTROLLER SERVICE ACCOUNT	30

5.2.1. Additional SCCs granted to the kubevirt-controller service account	30
5.2.2. Viewing the SCC and RBAC definitions for the kubevirt-controller	31
5.3. ADDITIONAL RESOURCES	31
CHAPTER 6. USING THE CLI TOOLS	32
6.1. PREREQUISITES	32
6.2. VIRTCTL CLIENT COMMANDS	32
6.3. OPENSIFT CONTAINER PLATFORM CLIENT COMMANDS	33
CHAPTER 7. VIRTUAL MACHINES	34
7.1. CREATING VIRTUAL MACHINES	34
7.1.1. Running the virtual machine wizard to create a virtual machine	34
7.1.1.1. Virtual machine wizard fields	35
7.1.1.2. Cloud-init fields	36
7.1.1.3. CD-ROM fields	37
7.1.1.4. Networking fields	37
7.1.1.5. Storage fields	38
Advanced storage settings	38
7.1.2. Pasting in a pre-configured YAML file to create a virtual machine	39
7.1.3. Using the CLI to create a virtual machine	39
7.1.4. Virtual machine storage volume types	40
7.1.5. Additional resources	42
7.2. EDITING VIRTUAL MACHINES	43
7.2.1. Editing a virtual machine in the web console	43
7.2.2. Editing a virtual machine YAML configuration using the web console	43
7.2.3. Editing a virtual machine YAML configuration using the CLI	44
7.2.4. Adding a virtual disk to a virtual machine	44
7.2.4.1. Storage fields	45
Advanced storage settings	45
7.2.5. Adding a network interface to a virtual machine	46
7.2.5.1. Networking fields	47
7.2.6. Editing CD-ROMs for Virtual Machines	47
7.3. EDITING BOOT ORDER	48
7.3.1. Adding items to a boot order list in the web console	48
7.3.2. Editing a boot order list in the web console	48
7.3.3. Editing a boot order list in the YAML configuration file	49
7.3.4. Removing items from a boot order list in the web console	49
7.4. DELETING VIRTUAL MACHINES	50
7.4.1. Deleting a virtual machine using the web console	50
7.4.2. Deleting a virtual machine by using the CLI	50
7.5. MANAGING VIRTUAL MACHINE INSTANCES	51
7.5.1. About virtual machine instances	51
7.5.2. Listing all virtual machine instances using the CLI	51
7.5.3. Listing standalone virtual machine instances using the web console	52
7.5.4. Editing a standalone virtual machine instance using the web console	52
7.5.5. Deleting a standalone virtual machine instance using the CLI	52
7.5.6. Deleting a standalone virtual machine instance using the web console	53
7.6. CONTROLLING VIRTUAL MACHINE STATES	53
7.6.1. Starting a virtual machine	53
7.6.2. Restarting a virtual machine	54
7.6.3. Stopping a virtual machine	54
7.6.4. Unpausing a virtual machine	55
7.7. ACCESSING VIRTUAL MACHINE CONSOLES	55

7.7.1. Virtual machine console sessions	56
7.7.2. Connecting to the virtual machine with the web console	56
7.7.2.1. Connecting to the terminal	56
7.7.2.2. Connecting to the serial console	56
7.7.2.3. Connecting to the VNC console	57
7.7.2.4. Connecting to the RDP console	57
7.7.3. Accessing virtual machine consoles by using CLI commands	58
7.7.3.1. Accessing a virtual machine instance via SSH	58
7.7.3.2. Accessing the serial console of a virtual machine instance	59
7.7.3.3. Accessing the graphical console of a virtual machine instances with VNC	59
7.7.3.4. Connecting to a Windows virtual machine with an RDP console	59
7.8. MANAGING CONFIGMAPS, SECRETS, AND SERVICE ACCOUNTS IN VIRTUAL MACHINES	61
7.8.1. Adding a secret, ConfigMap, or service account to a virtual machine	61
7.8.2. Removing a secret, ConfigMap, or service account from a virtual machine	62
7.8.3. Additional resources	62
7.9. INSTALLING VIRTIO DRIVER ON AN EXISTING WINDOWS VIRTUAL MACHINE	62
7.9.1. Understanding VirtIO drivers	63
7.9.2. Supported VirtIO drivers for Microsoft Windows virtual machines	63
7.9.3. Adding VirtIO drivers container disk to a virtual machine	63
7.9.4. Installing VirtIO drivers on an existing Windows virtual machine	64
7.9.5. Removing the VirtIO container disk from a virtual machine	65
7.10. INSTALLING VIRTIO DRIVER ON A NEW WINDOWS VIRTUAL MACHINE	65
7.10.1. Prerequisites	65
7.10.2. Understanding VirtIO drivers	65
7.10.3. Supported VirtIO drivers for Microsoft Windows virtual machines	66
7.10.4. Adding VirtIO drivers container disk to a virtual machine	66
7.10.5. Installing VirtIO drivers during Windows installation	67
7.10.6. Removing the VirtIO container disk from a virtual machine	68
7.11. ADVANCED VIRTUAL MACHINE MANAGEMENT	68
7.11.1. Automating management tasks	68
7.11.1.1. About Red Hat Ansible Automation	68
7.11.1.2. Automating virtual machine creation	68
7.11.1.3. Example: Ansible Playbook for creating virtual machines	70
7.11.2. Configuring PXE booting for virtual machines	70
7.11.2.1. Prerequisites	71
7.11.2.2. OpenShift Virtualization networking glossary	71
7.11.2.3. PXE booting with a specified MAC address	71
7.11.2.4. Template: virtual machine instance configuration file for PXE booting	74
7.11.3. Managing guest memory	75
7.11.3.1. Configuring guest memory overcommitment	75
7.11.3.2. Disabling guest memory overhead accounting	76
7.11.4. Using huge pages with virtual machines	77
7.11.4.1. Prerequisites	77
7.11.4.2. What huge pages do	77
7.11.4.3. Configuring huge pages for virtual machines	77
7.11.5. Enabling dedicated resources for virtual machines	78
7.11.5.1. About dedicated resources	78
7.11.5.2. Prerequisites	78
7.11.5.3. Enabling dedicated resources for a virtual machine	78
7.12. IMPORTING VIRTUAL MACHINES	79
7.12.1. TLS certificates for DataVolume imports	79
7.12.1.1. Adding TLS certificates for authenticating DataVolume imports	79
7.12.1.2. Example: ConfigMap created from a TLS certificate	79

7.12.2. Importing virtual machine images with DataVolumes	79
7.12.2.1. Prerequisites	80
7.12.2.2. CDI supported operations matrix	80
7.12.2.3. About DataVolumes	80
7.12.2.4. Importing a virtual machine image into a PersistentVolumeClaim by using a DataVolume	81
7.12.3. Importing virtual machine images to block storage with DataVolumes	84
7.12.3.1. Prerequisites	84
7.12.3.2. About DataVolumes	84
7.12.3.3. About block PersistentVolumes	84
7.12.3.4. Creating a local block PersistentVolume	84
7.12.3.5. Importing a virtual machine image to a block PersistentVolume using DataVolumes	85
7.12.3.6. CDI supported operations matrix	87
7.12.4. Importing a single Red Hat Virtualization virtual machine	87
7.12.4.1. OpenShift Virtualization storage feature matrix	88
7.12.4.2. Prerequisites for importing a virtual machine	88
7.12.4.3. Checking the default storage class	89
7.12.4.3.1. Checking the default storage class in the OpenShift Container Platform console	89
7.12.4.3.2. Checking the default storage class from the CLI	90
Changing the default storage class	90
7.12.4.4. Creating a ConfigMap for importing a Red Hat Virtualization virtual machine	91
7.12.4.5. Importing a virtual machine with the VM Import wizard	95
Virtual machine wizard fields	96
Networking fields	97
Storage fields	98
Advanced storage settings	98
7.12.4.6. Importing a Red Hat Virtualization virtual machine with the CLI	99
7.12.4.7. Canceling a virtual machine import	103
7.12.4.8. Troubleshooting a virtual machine import	103
7.12.4.8.1. Logs	103
7.12.4.8.2. Error messages	103
7.12.5. Importing a single VMware virtual machine or template	104
7.12.5.1. OpenShift Virtualization storage feature matrix	104
7.12.5.2. Preparing a VDDK image	104
7.12.5.2.1. Configuring an internal image registry	105
Changing the image registry's management state	105
Configuring registry storage for bare metal	105
Accessing registry directly from the cluster	106
Exposing a secure registry manually	108
7.12.5.2.2. Configuring an external image registry	109
Adding certificate authorities to the cluster	109
Allowing pods to reference images from other secured registries	109
7.12.5.2.3. Creating and using a VDDK image	110
7.12.5.3. Importing a virtual machine with the VM Import wizard	111
Virtual machine wizard fields	114
Cloud-init fields	115
Networking fields	115
Storage fields	116
Advanced storage settings	117
7.12.5.3.1. Updating the NIC name of an imported virtual machine	117
7.12.5.4. Troubleshooting a virtual machine import	118
7.12.5.4.1. Logs	118
7.12.5.4.2. Error messages	118
7.13. CLONING VIRTUAL MACHINES	119

7.13.1. Enabling user permissions to clone DataVolumes across namespaces	119
7.13.1.1. Prerequisites	119
7.13.1.2. About DataVolumes	119
7.13.1.3. Creating RBAC resources for cloning DataVolumes	119
7.13.2. Cloning a virtual machine disk into a new DataVolume	120
7.13.2.1. Prerequisites	120
7.13.2.2. About DataVolumes	121
7.13.2.3. Cloning the PersistentVolumeClaim of a virtual machine disk into a new DataVolume	121
7.13.2.4. Template: DataVolume clone configuration file	122
7.13.2.5. CDI supported operations matrix	122
7.13.3. Cloning a virtual machine by using a DataVolumeTemplate	123
7.13.3.1. Prerequisites	123
7.13.3.2. About DataVolumes	123
7.13.3.3. Creating a new virtual machine from a cloned PersistentVolumeClaim by using a DataVolumeTemplate	123
7.13.3.4. Template: DataVolume virtual machine configuration file	125
7.13.3.5. CDI supported operations matrix	126
7.13.4. Cloning a virtual machine disk into a new block storage DataVolume	126
7.13.4.1. Prerequisites	127
7.13.4.2. About DataVolumes	127
7.13.4.3. About block PersistentVolumes	127
7.13.4.4. Creating a local block PersistentVolume	127
7.13.4.5. Cloning the PersistentVolumeClaim of a virtual machine disk into a new DataVolume	128
7.13.4.6. CDI supported operations matrix	130
7.14. VIRTUAL MACHINE NETWORKING	130
7.14.1. Using the default Pod network for virtual machines	130
7.14.1.1. Configuring masquerade mode from the command line	130
7.14.1.2. Selecting binding method	131
7.14.1.2.1. Networking fields	131
7.14.1.3. Virtual machine configuration examples for the default network	132
7.14.1.3.1. Template: virtual machine configuration file	132
7.14.1.3.2. Template: Windows virtual machine instance configuration file	133
7.14.1.4. Creating a service from a virtual machine	134
7.14.2. Attaching a virtual machine to multiple networks	137
7.14.2.1. OpenShift Virtualization networking glossary	138
7.14.2.2. Creating a NetworkAttachmentDefinition	138
7.14.2.3. Prerequisites	138
7.14.2.3.1. Creating a Linux bridge NetworkAttachmentDefinition in the web console	138
7.14.2.3.2. Creating a Linux bridge NetworkAttachmentDefinition in the CLI	139
7.14.2.4. Creating a NIC for a virtual machine	141
7.14.2.5. Networking fields	141
7.14.3. Configuring an SR-IOV network device for virtual machines	142
7.14.3.1. Prerequisites	142
7.14.3.2. Automated discovery of SR-IOV network devices	142
7.14.3.2.1. Example SrioVNetworkNodeState object	142
7.14.3.3. Configuring SR-IOV network devices	144
7.14.3.4. Next steps	146
7.14.4. Defining an SR-IOV network	146
7.14.4.1. Prerequisites	146
7.14.4.2. Configuring SR-IOV additional network	146
7.14.4.3. Next steps	148
7.14.5. Attaching a virtual machine to an SR-IOV network	148
7.14.5.1. Prerequisites	148

7.14.5.2. Attaching a virtual machine to an SR-IOV network	148
7.14.6. Installing the QEMU guest agent on virtual machines	149
7.14.6.1. Installing QEMU guest agent on a Linux virtual machine	149
7.14.6.2. Installing QEMU guest agent on a Windows virtual machine	150
7.14.6.2.1. Installing VirtIO drivers on an existing Windows virtual machine	150
7.14.6.2.2. Installing VirtIO drivers during Windows installation	150
7.14.7. Viewing the IP address of NICs on a virtual machine	151
7.14.7.1. Prerequisites	151
7.14.7.2. Viewing the IP address of a virtual machine interface in the CLI	151
7.14.7.3. Viewing the IP address of a virtual machine interface in the web console	152
7.14.8. Using a MAC address pool for virtual machines	152
7.14.8.1. About KubeMacPool	152
7.14.8.2. Enabling a MAC address pool for a namespace in the CLI	153
7.14.8.3. Disabling a MAC address pool for a namespace in the CLI	153
7.15. VIRTUAL MACHINE DISKS	153
7.15.1. Storage features	153
7.15.1.1. OpenShift Virtualization storage feature matrix	153
7.15.2. Configuring local storage for virtual machines	154
7.15.2.1. About the hostpath provisioner	154
7.15.2.2. Configuring SELinux for the hostpath provisioner on Red Hat Enterprise Linux CoreOS 8	155
7.15.2.3. Using the hostpath provisioner to enable local storage	156
7.15.2.4. Creating a StorageClass object	157
7.15.3. Configuring CDI to work with namespaces that have a compute resource quota	158
7.15.3.1. About CPU and memory quotas in a namespace	158
7.15.3.2. Editing the CDIConfig object to override CPU and memory defaults	158
7.15.3.3. Additional resources	159
7.15.4. Uploading local disk images by using the virtctl tool	159
7.15.4.1. Prerequisites	159
7.15.4.2. About DataVolumes	159
7.15.4.3. Creating an upload DataVolume	159
7.15.4.4. Uploading a local disk image to a DataVolume	160
7.15.4.5. CDI supported operations matrix	161
7.15.5. Uploading a local disk image to a block storage DataVolume	162
7.15.5.1. Prerequisites	162
7.15.5.2. About DataVolumes	162
7.15.5.3. About block PersistentVolumes	162
7.15.5.4. Creating a local block PersistentVolume	162
7.15.5.5. Creating an upload DataVolume	163
7.15.5.6. Uploading a local disk image to a DataVolume	164
7.15.5.7. CDI supported operations matrix	165
7.15.6. Moving a local virtual machine disk to a different node	166
7.15.6.1. Cloning a local volume to another node	166
7.15.7. Expanding virtual storage by adding blank disk images	168
7.15.7.1. About DataVolumes	169
7.15.7.2. Creating a blank disk image with DataVolumes	169
7.15.7.3. Template: DataVolume configuration file for blank disk images	169
7.15.8. Storage defaults for DataVolumes	170
7.15.8.1. About storage settings for DataVolumes	170
7.15.8.1.1. Access modes	170
7.15.8.1.2. Volume modes	170
7.15.8.2. Editing the kubevirt-storage-class-defaults config map in the web console	171
7.15.8.3. Editing the kubevirt-storage-class-defaults config map in the CLI	171
7.15.8.4. Example of multiple storage class defaults	172

7.15.9. Using container disks with virtual machines	172
7.15.9.1. About container disks	173
7.15.9.1.1. Importing a container disk into a PVC by using a DataVolume	173
7.15.9.1.2. Attaching a container disk to a virtual machine as a containerDisk volume	173
7.15.9.2. Preparing a container disk for virtual machines	173
7.15.9.3. Disabling TLS for a container registry to use as insecure registry	174
7.15.9.4. Next steps	174
7.15.10. Preparing CDI scratch space	174
7.15.10.1. About DataVolumes	174
7.15.10.2. Understanding scratch space	175
Manual provisioning	175
7.15.10.3. CDI operations that require scratch space	175
7.15.10.4. Defining a StorageClass in the CDI configuration	176
7.15.10.5. CDI supported operations matrix	176
7.15.11. Re-using persistent volumes	177
7.15.11.1. About reclaiming statically provisioned persistent volumes	177
7.15.11.2. Reclaiming statically provisioned persistent volumes	177
7.15.12. Deleting DataVolumes	178
7.15.12.1. About DataVolumes	178
7.15.12.2. Listing all DataVolumes	179
7.15.12.3. Deleting a DataVolume	179
CHAPTER 8. VIRTUAL MACHINE TEMPLATES	180
8.1. CREATING VIRTUAL MACHINE TEMPLATES	180
8.1.1. Creating a virtual machine template with the interactive wizard in the web console	180
8.1.2. Virtual machine template interactive wizard fields	181
8.1.2.1. Virtual machine template wizard fields	181
8.1.2.2. Cloud-init fields	182
8.1.2.3. Networking fields	182
8.1.2.4. Storage fields	183
Advanced storage settings	184
8.2. EDITING VIRTUAL MACHINE TEMPLATES	184
8.2.1. Editing a virtual machine template in the web console	184
8.2.2. Editing virtual machine template YAML configuration in the web console	185
8.2.3. Adding a virtual disk to a virtual machine template	185
8.2.4. Adding a network interface to a virtual machine template	185
8.2.5. Editing CD-ROMs for Virtual Machine Templates	186
8.3. ENABLING DEDICATED RESOURCES FOR VIRTUAL MACHINE TEMPLATES	187
8.3.1. About dedicated resources	187
8.3.2. Prerequisites	187
8.3.3. Enabling dedicated resources for a virtual machine template	187
8.4. DELETING A VIRTUAL MACHINE TEMPLATE	187
8.4.1. Deleting a virtual machine template in the web console	187
CHAPTER 9. LIVE MIGRATION	189
9.1. VIRTUAL MACHINE LIVE MIGRATION	189
9.1.1. Prerequisites	189
9.1.2. Understanding live migration	189
9.1.3. Updating access mode for LiveMigration	189
9.2. LIVE MIGRATION LIMITS AND TIMEOUTS	189
9.2.1. Configuring live migration limits and timeouts	189
9.2.2. Cluster-wide live migration limits and timeouts	190
9.3. MIGRATING A VIRTUAL MACHINE INSTANCE TO ANOTHER NODE	191

9.3.1. Initiating live migration of a virtual machine instance in the web console	191
9.3.2. Initiating live migration of a virtual machine instance in the CLI	191
9.4. MONITORING LIVE MIGRATION OF A VIRTUAL MACHINE INSTANCE	192
9.4.1. Monitoring live migration of a virtual machine instance in the web console	192
9.4.2. Monitoring live migration of a virtual machine instance in the CLI	192
9.5. CANCELLING THE LIVE MIGRATION OF A VIRTUAL MACHINE INSTANCE	193
9.5.1. Cancelling live migration of a virtual machine instance in the web console	193
9.5.2. Cancelling live migration of a virtual machine instance in the CLI	193
9.6. CONFIGURING VIRTUAL MACHINE EVICTION STRATEGY	194
9.6.1. Configuring custom virtual machines with the LiveMigration eviction strategy	194
CHAPTER 10. NODE MAINTENANCE	195
10.1. AUTOMATIC RENEWAL OF TLS CERTIFICATES	195
10.1.1. Automatic renewal of TLS certificates	195
10.2. NODE MAINTENANCE MODE	195
10.2.1. Understanding node maintenance mode	195
10.3. SETTING A NODE TO MAINTENANCE MODE	195
10.3.1. Understanding node maintenance mode	195
10.3.2. Setting a node to maintenance mode in the web console	196
10.3.3. Setting a node to maintenance mode in the CLI	196
10.4. RESUMING A NODE FROM MAINTENANCE MODE	197
10.4.1. Resuming a node from maintenance mode in the web console	197
10.4.2. Resuming a node from maintenance mode in the CLI	197
CHAPTER 11. NODE NETWORKING	199
11.1. OBSERVING NODE NETWORK STATE	199
11.1.1. About nmstate	199
11.1.2. Viewing the network state of a node	199
11.2. UPDATING NODE NETWORK CONFIGURATION	200
11.2.1. About nmstate	200
11.2.2. Creating an interface on nodes	201
Additional resources	202
11.2.3. Confirming Policy updates on nodes	202
11.2.4. Removing an interface from nodes	202
11.2.5. Restoring node network configuration after removing an interface	203
11.2.6. Example Policy configurations for different interfaces	204
11.2.6.1. Example: Linux bridge interface NodeNetworkConfigurationPolicy	204
11.2.6.2. Example: VLAN interface NodeNetworkConfigurationPolicy	205
11.2.6.3. Example: Bond interface NodeNetworkConfigurationPolicy	206
11.2.6.4. Example: Ethernet interface NodeNetworkConfigurationPolicy	207
11.2.6.5. Example: Multiple interfaces in the same Policy	208
11.2.7. Examples: IP management	208
11.2.7.1. Static	208
11.2.7.2. No IP address	209
11.2.7.3. Dynamic host configuration	209
11.2.7.4. DNS	210
11.2.7.5. Static routing	210
11.3. TROUBLESHOOTING NODE NETWORK CONFIGURATION	210
11.3.1. Troubleshooting an incorrect NodeNetworkConfigurationPolicy configuration	211
CHAPTER 12. LOGGING, EVENTS, AND MONITORING	216
12.1. VIEWING VIRTUAL MACHINE LOGS	216
12.1.1. Understanding virtual machine logs	216
12.1.2. Viewing virtual machine logs in the CLI	216

12.1.3. Viewing virtual machine logs in the web console	216
12.2. VIEWING EVENTS	216
12.2.1. Understanding virtual machine events	217
12.2.2. Viewing the events for a virtual machine in the web console	217
12.2.3. Viewing namespace events in the CLI	217
12.2.4. Viewing resource events in the CLI	217
12.3. DIAGNOSING DATAVOLUMES USING EVENTS AND CONDITIONS	218
12.3.1. About conditions and events	218
12.3.2. Analyzing DataVolumes using conditions and events	218
12.4. VIEWING INFORMATION ABOUT VIRTUAL MACHINE WORKLOADS	220
12.4.1. About the Virtual Machines dashboard	220
12.5. MONITORING VIRTUAL MACHINE HEALTH	221
12.5.1. About liveness and readiness probes	221
12.5.2. Define an HTTP liveness probe	221
12.5.3. Define a TCP liveness probe	222
12.5.4. Define a readiness probe	223
12.6. USING THE OPENSIFT CONTAINER PLATFORM DASHBOARD TO GET CLUSTER INFORMATION	225
12.6.1. About the OpenShift Container Platform dashboards page	225
12.7. OPENSIFT CONTAINER PLATFORM CLUSTER MONITORING, LOGGING, AND TELEMETRY	226
12.7.1. About OpenShift Container Platform cluster monitoring	226
12.7.2. About cluster logging components	226
12.7.3. About Telemetry	227
12.7.3.1. Information collected by Telemetry	227
12.7.4. CLI troubleshooting and debugging commands	228
12.8. COLLECTING OPENSIFT VIRTUALIZATION DATA FOR RED HAT SUPPORT	228
12.8.1. About the must-gather tool	228
12.8.2. About collecting OpenShift Virtualization data	228
12.8.3. Gathering data about specific features	229

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 a feature of OpenShift Container Platform that you can use to run and manage virtual machine workloads alongside container workloads.

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

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

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

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

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

1.1.1. OpenShift Virtualization supported cluster version

OpenShift Virtualization 2.4 is supported for use on OpenShift Container Platform 4.5 clusters.

CHAPTER 2. OPENSIFT VIRTUALIZATION RELEASE NOTES

2.1. ABOUT RED HAT OPENSIFT VIRTUALIZATION

Red Hat OpenShift Virtualization is supported for use on OpenShift Container Platform 4.5 clusters. Previously known as container-native virtualization, 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 a new logo:



2.1.1. What you can do with OpenShift Virtualization

OpenShift Virtualization is a feature of OpenShift Container Platform that you can use to run and manage virtual machine workloads alongside container workloads.

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

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

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

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

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

2.2. NEW AND CHANGED FEATURES

- You can now [install OpenShift Virtualization by using the CLI](#) to apply manifests to your OpenShift Container Platform cluster.
- OpenShift Virtualization is certified in Microsoft's Windows Server Virtualization Validation Program (SVVP) to run Windows Server workloads.
The SVVP Certification applies to:
 - Red Hat Enterprise Linux CoreOS 8 workers. In the Microsoft SVVP Catalog, they are named *Red Hat OpenShift Container Platform 4 on RHEL CoreOS 8*.

- Intel and AMD CPUs.
- OpenShift Virtualization rotates and renews TLS certificates at regular intervals. This automatic process does not disrupt any operations.
- This release features significant security enhancements. OpenShift Virtualization now supports SELinux with Mandatory Access Control (MAC) for isolating virtual machines (VMs). Previously, all VMs were managed by using privileged [Security Context Constraints \(SCC\)](#). Now, you can use less privileged custom SCCs for VMs and limit the use of privileged SCCs to infrastructure containers in the cluster.
- You can now enable access to your Red Hat Enterprise Linux entitlement for RHEL virtual machines. Configure the **virt-who** daemon to report the running VMs in your OpenShift Container Platform cluster. This gives the [Red Hat Subscription Manager](#) in the RHEL VM access to your entitlements.

2.2.1. Supported guest operating systems

OpenShift Virtualization guests can use the following operating systems:

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

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

2.2.2. Networking

- OpenShift Virtualization is now integrated with the OpenShift Container Platform [Single Root I/O Virtualization \(SR-IOV\) Operator](#). You can now [attach virtual machines to SR-IOV networks](#) in your cluster.
- [MAC address pool](#) is now supported in OpenShift Virtualization. It is disabled by default in the cluster and can be enabled per namespace.

2.2.3. Storage

- You can now configure the **Volume Mode** and **Access Mode** for a virtual disk when you [add a disk to a virtual machine](#) in the web console. This is also possible when adding a disk to a [new virtual machine using the wizard](#).
- Using OpenShift Container Storage (OCS) with OpenShift Virtualization gives you the benefits of fault-tolerant storage and the ability to live migrate between nodes.
- You can now 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. The default compute resource limits are set to **0** but administrators can configure the resource limits applied to CDI worker Pods.
- The **virtctl** tool can now use a **DataVolume** when uploading virtual machine disks to the cluster. This helps prevent virtual machines from being inadvertently started before an upload has completed.

- OpenShift Container Storage DataVolumes have been enhanced with conditions and events that make it easier to understand the state of virtual disk imports, clones, and upload operations. Conditions and events also simplify troubleshooting.

2.2.4. Web console

- In the web console, the sidebar items **Virtual Machines** and **Virtual Machine Templates** have been replaced by a single sidebar menu item labeled **Virtualization**. When you click **Virtualization**, you have access to two tabs: **Virtual Machines** and **Virtual Machine Templates**.
- You can now configure the scheduling properties of virtual machines by accessing the **Scheduling and resources requirements** section of the **Virtual Machine Details** page. For example, you can view and manage affinity rules, dedicated resources, and tolerations for tainted nodes. You can also search for nodes with labels that match specific key/value pairs by using the **Node Selector**.
- You can now add [secrets](#), [ConfigMaps](#), and [service accounts](#) to a virtual machine on the **Virtual Machine Overview** → **Environment** page of the OpenShift Container Platform web console. You can also remove these resources on the same page.

2.3. NOTABLE TECHNICAL CHANGES

- OpenShift Virtualization can be installed on disconnected clusters in restricted network environments that do not have Internet connectivity. You can create local mirrors for the OpenShift Virtualization Operator and install the Operator from a local catalog image that is accessible to the disconnected cluster. Learn more about [Installing OpenShift Virtualization on a restricted network cluster](#).
- You can import a single Red Hat Virtualization virtual machine by using the virtual machine wizard or the CLI.
- Every component in OpenShift Virtualization now uses its own API subgroup, **<component_name>.kubevirt.io**.

2.4. KNOWN ISSUES

- If you enable a MAC address pool for a namespace by applying the KubeMacPool label and using the **io** attribute for virtual machines in that namespace, the **io** attribute configuration is not retained for the VMs. As a workaround, do not use the **io** attribute for VMs. Alternatively, you can disable KubeMacPool for the namespace. ([BZ#1869527](#))
- If container-native virtualization 2.3 is installed on your OpenShift Container Platform 4.4 cluster, upgrading the cluster to version 4.5 may cause a migrating virtual machine instance (VMI) to fail. This is because the virt-launcher Pod does not successfully notify the virt-handler Pod that migration has failed. The result is that the source VMI **migrationState** is not updated. ([BZ#1859661](#))
 - As a workaround, delete the virt-handler Pod on the source node where the VMI is running. This restarts the virt-handler Pod, which updates the VMI status and restarts VMI migration:

1. Find the name of the source node where the VMI is running:

```
$ oc get vmi -o wide
```

2. Delete the virt-handler Pod on the source node:

```
$ oc delete pod -n openshift-cnv --selector=kubevirt.io=virt-handler --field-selector=spec.nodeName=<source-node-name> 1
```

- 1** Where <source-node-name> is the name of the source node that the VMI is migrating from.

- Common templates in previous versions of OpenShift Virtualization had a default **spec.terminationGracePeriodSeconds** value of **0**. Virtual machines created from these older common templates can encounter disk issues from being forcefully terminated. If you upgrade to OpenShift Virtualization 2.4, both older and newer versions of common templates are available for each combination of operating system, workload, and flavor. When you create a virtual machine by using a common template, you must use the newer version of the template. Disregard the older version to avoid issues. ([BZ#1859235](#))

- To verify if a virtual machine is affected by this bug, run the following command in the namespace of the virtual machine to determine the **spec.terminationGracePeriodSeconds** value:

```
$ oc get vm <virtual-machine-name> -o yaml | grep "terminationGracePeriodSeconds"
```

- If the virtual machine has a **terminationGracePeriodSeconds** value of **0**, patch the virtual machine config with a **spec.terminationGracePeriodSeconds** value of **180** for Linux virtual machines, or a value of **3600** for Windows virtual machines.

```
$ oc patch vm <virtual-machine-name> --type merge -p '{"spec":{"template":{"spec":{"terminationGracePeriodSeconds":180}}}}'
```

- Running virtual machines that cannot be live migrated might block an OpenShift Container Platform cluster upgrade. This includes virtual machines that use hostpath-provisioner storage or SR-IOV network interfaces. As a workaround, you can reconfigure the virtual machines so that they can be powered off during a cluster upgrade. In the **spec** section of the virtual machine configuration file:
 - Remove the **evictionStrategy: LiveMigrate** field. See [Configuring virtual machine eviction strategy](#) for more information on how to configure eviction strategy.
 - Set the **runStrategy** field to **Always**.
- For unknown reasons, memory consumption for the **containerDisk** volume type might gradually increase until it exceeds the memory limit. To resolve this issue, restart the VM. ([BZ#1855067](#))
- Sometimes, when attempting to edit the subscription channel of the **OpenShift Virtualization Operator** in the web console, clicking the **Channel** button of the **Subscription Overview** results in a JavaScript error. ([BZ#1796410](#))
 - As a workaround, trigger the upgrade process to OpenShift Virtualization 2.4 from the CLI by running the following **oc** patch command:

```
$ export TARGET_NAMESPACE=openshift-cnv CNV_CHANNEL=2.4 && oc patch -n "${TARGET_NAMESPACE}" $(oc get subscription -n ${TARGET_NAMESPACE} --no-headers -o name) --type='json' -p='[{"op": "replace", "path": "/spec/channel", "value":"${CNV_CHANNEL}"}, {"op": "replace", "path": "/spec/installPlanApproval", "value":"Automatic"}]'
```

This command points your subscription to upgrade channel **2.4** and enables automatic updates.

- After migration, a virtual machine is assigned a new IP address. However, the commands **oc get vmi** and **oc describe vmi** still generate output containing the obsolete IP address. ([BZ#1686208](#))

- As a workaround, view the correct IP address by running the following command:

```
$ oc get pod -o wide
```

- Live migration fails when nodes have different CPU models. Even in cases where nodes have the same physical CPU model, differences introduced by microcode updates have the same effect. This is because the default settings trigger host CPU passthrough behavior, which is incompatible with live migration. ([BZ#1760028](#))
- As a workaround, set the default CPU model in the **kubevirt-config** ConfigMap, as shown in the following example:



NOTE

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

1. Open the **kubevirt-config** ConfigMap for editing by running the following command:

```
$ oc edit configmap kubevirt-config -n openshift-cnv
```

2. Edit the ConfigMap:

```
kind: ConfigMap
metadata:
  name: kubevirt-config
data:
  default-cpu-model: "<cpu-model>" 1
```

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

- OpenShift Virtualization cannot reliably identify node drains that are triggered by running either **oc adm drain** or **kubectl drain**. Do not run these commands on the nodes of any clusters where OpenShift Virtualization is deployed. The nodes might not drain if there are virtual machines running on top of them.
 - The current solution is to [put nodes into maintenance](#).
- You must create a custom ConfigMap in order to import a Red Hat Virtualization (RHV) VM into OpenShift Virtualization.
- You cannot import a RHV VM if the target VM name exceeds 63 characters. ([BZ#1857165](#))
- If the OpenShift Virtualization storage PV is not suitable for importing a RHV VM, the progress bar remains at 10% and the import does not complete. The VM Import Controller Pod log

displays the following error message: **Failed to bind volumes: provisioning failed for PVC.** ([BZ#1857784](#))

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

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

- If a RHV VM disk is in a **Locked** state, you must [unlock the disk](#) before you can import it.
- **cloud-init** settings are not imported with a RHV virtual machine. You must recreate **cloud-init** after the import process.
- OpenShift Virtualization does not support UEFI. If you import a VMware VM with UEFI BIOS into OpenShift Virtualization, the VM will not boot. ([BZ#1880083](#))

CHAPTER 3. OPENSIFT VIRTUALIZATION INSTALLATION

3.1. CONFIGURING YOUR CLUSTER FOR OPENSIFT VIRTUALIZATION

Before you install OpenShift Virtualization, ensure that your OpenShift Container Platform cluster meets the following requirements:

- Your cluster must be installed on [bare metal](#) infrastructure with Red Hat Enterprise Linux CoreOS workers.
- Shared storage is required to enable live migration.
- You must manage your Compute nodes according to the number and size of the virtual machines that you want to host in the cluster.
- To deploy OpenShift Virtualization in a disconnected environment, you must [configure Operator Lifecycle Manager on restricted networks](#).
- To use proxy with OpenShift Virtualization, you must [configure proxy support in Operator Lifecycle Manager](#).
- If your cluster uses worker nodes from multiple CPU vendors, live migration failures can occur. For example, a virtual machine with an AMD CPU might attempt to live-migrate to a node with an Intel CPU and likely fail migration. To avoid this, label nodes with a vendor-specific label, such as **Vendor=Intel** or **Vendor=AMD**, and set node affinity on your virtual machines to ensure successful migration. See [Configuring a required node affinity rule](#) for more information.

OpenShift Virtualization works with OpenShift Container Platform by default, but the following installation configurations are recommended:

- Configure [monitoring](#) in the cluster.



NOTE

To obtain an evaluation version of OpenShift Container Platform, download a trial from the OpenShift Container Platform home page.

3.2. 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 4.5 [web console](#) to subscribe to and deploy the OpenShift Virtualization Operators.

3.2.1. Prerequisites

- Install OpenShift Container Platform 4.5 on your cluster.
- Log in as a user with **cluster-admin** permissions.

3.2.2. Subscribing to the OpenShift Virtualization catalog

Before you install OpenShift Virtualization, subscribe to the **OpenShift Virtualization** catalog from the OpenShift Container Platform web console. Subscribing gives the **openshift-cnv** namespace access to the OpenShift Virtualization Operators.

Procedure

1. Open a browser window and log in to the OpenShift Container Platform web console.
2. Navigate to the **Operators → OperatorHub** page.
3. Search for **OpenShift Virtualization** and then select it.
4. Read the information about the Operator and click **Install**.
5. On the **Install Operator** page:
 - a. 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.

- b. Select **2.4** from the list of available **Update Channel** options.
 - c. For **Approval Strategy**, ensure that **Automatic**, which is the default value, is selected. OpenShift Virtualization automatically updates when a new z-stream release is available.
6. Click **Install** to make the Operator available to the **openshift-cnv** namespace. On the **Installed Operators** screen, the **Status** displays **Succeeded** when OpenShift Virtualization finishes installation.

3.2.3. Deploying OpenShift Virtualization

After subscribing to the **OpenShift Virtualization** catalog, create the **OpenShift Virtualization Operator Deployment** custom resource to deploy OpenShift Virtualization.

Prerequisites

- Subscribe to the **OpenShift Virtualization** catalog in the **openshift-cnv** namespace.

Procedure

1. Navigate to the **Operators → Installed Operators** page.
2. Click **OpenShift Virtualization**.
3. Click the **OpenShift Virtualization Operator Deployment** tab and click **Create HyperConverged Cluster**.

**WARNING**

To avoid deployment errors, do not rename the custom resource. Before you proceed to the next step, ensure that the custom resource is named the default **kubevirt-hyperconverged**.

4. Click **Create** to launch OpenShift Virtualization.
5. 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 access OpenShift Virtualization.

3.2.4. Next steps

You might want to additionally configure the following components:

- The *KubeMacPool* component provides a MAC address pool service for virtual machine NICs in designated namespaces. [Enable a MAC address pool in a namespace](#) by applying the KubeMacPool label to that namespace.
- 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.

3.3. 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.

3.3.1. Prerequisites

- Install OpenShift Container Platform 4.5 on your cluster.
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

3.3.2. 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.v2.4.9
  channel: "2.4"

```

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

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

3.3.3. 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/v1alpha1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  BareMetalPlatform: true

```

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 ClusterServiceVersion (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
kubvirt-hyperconverged-operator.v2.4.9  OpenShift Virtualization  2.4.9
Succeeded
```

3.3.4. Next steps

You might want to additionally configure the following components:

- The *KubeMacPool* component provides a MAC address pool service for virtual machine NICs in designated namespaces. [Enable a MAC address pool in a namespace](#) by applying the KubeMacPool label to that namespace.
- 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.

3.4. INSTALLING THE VIRTCTL CLIENT

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

Install the client to your system by enabling the OpenShift Virtualization repository and installing the **kubvirt-virtctl** package.

3.4.1. Enabling OpenShift Virtualization repositories

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

- Red Hat Enterprise Linux 8 repository: **cnv-2.4-for-rhel-8-x86_64-rpms**
- Red Hat Enterprise Linux 7 repository: **rhel-7-server-cnv-2.4-rpms**

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

Procedure

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

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

3.4.2. Installing the virtctl client

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

Procedure

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

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

See also: [Using the CLI tools](#) for OpenShift Virtualization.

3.5. UNINSTALLING OPENSIFT VIRTUALIZATION USING THE WEB CONSOLE

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

3.5.1. Prerequisites

- You must have OpenShift Virtualization 2.4 installed.
- You must delete all [virtual machines](#), [virtual machine instances](#), and [DataVolumes](#).



IMPORTANT

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

3.5.2. Deleting the OpenShift Virtualization Operator Deployment custom resource


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

Prerequisites

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

Procedure

1. From the OpenShift Container Platform web console, select **openshift-cnvm** 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
```

3.5.3. Deleting the OpenShift Virtualization catalog subscription

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

Prerequisites

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

Procedure

1. Navigate to the **Operators → OperatorHub** page.
2. Search for **OpenShift Virtualization** and then select it.
3. Click **Uninstall**.



NOTE

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

3.5.4. Deleting a namespace using the web console

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



NOTE

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

Procedure

1. Navigate to **Administration → Namespaces**.
2. Locate the namespace that you want to delete in the list of namespaces.
3. On the far right side of the namespace listing, select **Delete Namespace** from the Options

menu  .

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

3.6. UNINSTALLING OPENSIFT VIRTUALIZATION USING THE CLI

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

3.6.1. Prerequisites

- You must have OpenShift Virtualization 2.4 installed.
- You must delete all [virtual machines](#), [virtual machine instances](#), and [DataVolumes](#).



IMPORTANT

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

3.6.2. Deleting OpenShift Virtualization

You can delete OpenShift Virtualization by using the CLI.

Prerequisites

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



NOTE

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

Procedure

1. Delete the HyperConverged Custom Resource:

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

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

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

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

```
$ CSV_NAME=$(oc get csv -n openshift-cnv -o=custom-columns=: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.v2.4.9"  
deleted
```

CHAPTER 4. UPGRADING OPENSIFT VIRTUALIZATION

You can manually upgrade to the next minor version of OpenShift Virtualization and monitor the status of an update by using the web console.

4.1. ABOUT UPGRADING OPENSIFT VIRTUALIZATION

4.1.1. How OpenShift Virtualization upgrades work

- You can upgrade to the next minor version of OpenShift Virtualization by using the OpenShift Container Platform web console to change the channel of your Operator subscription.
- You can enable automatic *z-stream* updates during OpenShift Virtualization installation.
- Updates are delivered via the *Marketplace Operator*, which is deployed during OpenShift Container Platform installation. The Marketplace Operator makes external Operators available to your cluster.
- The amount of time an update takes to complete depends on your network connection. Most automatic updates complete within fifteen minutes.

4.1.2. How OpenShift Virtualization upgrades affect your cluster

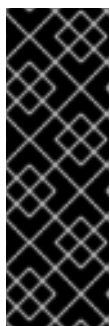
- Upgrading does not interrupt virtual machine workloads.
 - Virtual machine pods are not restarted or migrated during an upgrade. If you need to update the **virt-launcher** pod, you must restart or live migrate the virtual machine.



NOTE

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

- Upgrading does not interrupt network connections.
- DataVolumes and their associated PersistentVolumeClaims are preserved during upgrade.



IMPORTANT

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

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

4.2. UPGRADING OPENSIFT VIRTUALIZATION TO THE NEXT MINOR VERSION

You can manually upgrade OpenShift Virtualization to the next minor version by using the OpenShift Container Platform web console to change the channel of your Operator subscription.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.

Procedure

1. Access the OpenShift Container Platform web console and navigate to **Operators** → **Installed Operators**.
2. Click **OpenShift Virtualization** to open the **Operator Details** page.
3. Click the **Subscription** tab to open the **Subscription Overview** page.
4. In the **Channel** pane, click the pencil icon on the right side of the version number to open the **Change Subscription Update Channel** window.
5. Select the next minor version. For example, if you want to upgrade to OpenShift Virtualization 2.4, select **2.4**.
6. Click **Save**.
7. Check the status of the upgrade by navigating to **Operators** → **Installed Operators**. You can also check the status by running the following **oc** command:

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

4.3. MONITORING UPGRADE STATUS

The best way to monitor OpenShift Virtualization upgrade status is to watch the ClusterServiceVersion (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

- Access 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
```

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

Example output

VERSION REPLACES	PHASE
2.4.9 kubevirt-hyperconverged-operator.v2.4.6	Installing
2.4.6	Replacing

- 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
```

4.4. ADDITIONAL RESOURCES

- [ClusterServiceVersions \(CSVs\)](#)
- [Configuring virtual machine eviction strategy](#)

CHAPTER 5. ADDITIONAL SECURITY PRIVILEGES GRANTED FOR KUBEVIRT-CONTROLLER AND VIRT-LAUNCHER

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

5.1. EXTENDED SELINUX POLICIES FOR VIRT-LAUNCHER PODS

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

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

These rules enable the following virtualization features:

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

5.2. ADDITIONAL OPENSIFT CONTAINER PLATFORM SECURITY CONTEXT CONSTRAINTS AND LINUX CAPABILITIES FOR THE KUBEVIRT-CONTROLLER SERVICE ACCOUNT

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

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

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

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

- **scc.AllowHostDirVolumePlugin = true**
This allows virtual machines to use the hostPath volume 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**.

5.2.2. Viewing the SCC and RBAC definitions for the kubevirt-controller

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

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

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

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

5.3. ADDITIONAL RESOURCES

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

CHAPTER 6. 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

6.1. PREREQUISITES

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

6.2. VIRTCTL CLIENT COMMANDS

The **virtctl** client is a command-line utility for managing OpenShift Virtualization resources. The following table contains the **virtctl** commands used throughout the OpenShift Virtualization documentation.

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

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

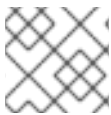
Table 6.1. **virtctl** client commands

Command	Description
virtctl start <vm_name>	Start a virtual machine.
virtctl stop <vm_name>	Stop a virtual machine.
virtctl pause vm vmi <object_name>	Pause a virtual machine or virtual machine instance. The machine state is kept in memory.
virtctl unpause vm vmi <object_name>	Unpause a virtual machine or virtual machine instance.
virtctl migrate <vm_name>	Migrate a virtual machine.
virtctl restart <vm_name>	Restart a virtual machine.
virtctl expose <vm_name>	Create a service that forwards a designated port of a virtual machine or virtual machine instance and expose the service on the specified port of the node.
virtctl console <vmi_name>	Connect to a serial console of a virtual machine instance.
virtctl vnc <vmi_name>	Open a VNC connection to a virtual machine instance.

Command	Description
virtctl image-upload dv <datavolume_name> --image-path= </path/to/image> --no-create	Upload a virtual machine image to a DataVolume 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 DataVolume.
virtctl version	Display the client and server version information.
virtctl help	Display a descriptive list of virtctl commands.

6.3. OPENSIFT CONTAINER PLATFORM CLIENT COMMANDS

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



NOTE

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

Table 6.2. **oc** commands

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

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

CHAPTER 7. VIRTUAL MACHINES

7.1. CREATING VIRTUAL MACHINES

Use one of these procedures to create a virtual machine:

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



WARNING

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

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

The web console features an interactive wizard that guides you through **General**, **Networking**, **Storage**, **Advanced**, and **Review** steps to simplify the process of creating virtual machines. All required fields are marked by a *. When the required fields are completed, you can review and create your virtual machine.

Network Interface Cards (NICs) and storage disks can be created and attached to virtual machines after they have been created.

Bootable Disk

If either **URL** or **Container** are selected as the **Source** in the **General** step, a **rootdisk** disk is created and attached to the virtual machine as the **Bootable Disk**. You can modify the **rootdisk** but you cannot remove it.




A **Bootable Disk** is not required for virtual machines provisioned from a **PXE** source if there are no disks attached to the virtual machine. If one or more disks are attached to the virtual machine, you must select one as the **Bootable Disk**.

Prerequisites

- When you create your virtual machine using the wizard, your virtual machine's storage medium must support Read-Write-Many (RWX) PVCs.

Procedure

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

4. Fill in all required fields in the **General** step. Selecting a **Template** automatically fills in these fields.
5. Click **Next** to progress to the **Networking** step. A **nic0** NIC is attached by default.
 - a. Optional: Click **Add Network Interface** to create additional NICs.
 - b. Optional: You can remove any or all NICs by clicking the Options menu  and selecting **Delete**. A virtual machine does not need a NIC attached to be created. NICs can be created after the virtual machine has been created.
6. Click **Next** to progress to the **Storage** screen.
 - a. Optional: Click **Add Disk** to create additional disks. These disks can be removed by clicking the Options menu  and selecting **Delete**.
 - b. Optional: Click the Options menu  to edit the disk and save your changes.
7. Click **Review and Create**. The **Results** screen displays the JSON configuration file for the virtual machine.

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

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

7.1.1.1. Virtual machine wizard fields

Name	Parameter	Description
Template		Template from which to create the virtual machine. Selecting a template will automatically complete other fields.
Source	PXE	Provision virtual machine from PXE menu. Requires a PXE-capable NIC in the cluster.
	URL	Provision virtual machine from an image available from an HTTP or S3 endpoint.
	Container	Provision virtual machine from a bootable operating system container located in a registry accessible from the cluster. Example: kubevirt/cirros-registry-disk-demo .

Name	Parameter	Description
	Disk	Provision virtual machine from a disk.
Operating System		The primary operating system that is selected for the virtual machine.
Flavor	small, medium, large, tiny, Custom	Presets that determine the amount of CPU and memory allocated to the virtual machine. The presets displayed for Flavor are determined by the operating system.
Memory		Size in GiB of the memory allocated to the virtual machine.
CPUs		The amount of CPU allocated to the virtual machine.
Workload Profile	High Performance	A virtual machine configuration that is optimized for high-performance workloads.
	Server	A profile optimized to run server workloads.
	Desktop	A virtual machine configuration for use on a desktop.
Name		The name can contain lowercase letters (a-z), numbers (0-9), and hyphens (-), up to a maximum of 253 characters. The first and last characters must be alphanumeric. The name must not contain uppercase letters, spaces, periods (.), or special characters.
Description		Optional description field.
Start virtual machine on creation		Select to automatically start the virtual machine upon creation.

7.1.1.2. Cloud-init fields

Name	Description
Hostname	Sets a specific host name for the virtual machine.
Authenticated 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.

7.1.1.3. CD-ROM fields

Source	Description
Container	Specify the container path. For example: kubevirt/fedora-registry-disk: latest .
URL	Specify the URL path and size in GiB. Then, select the storage class for this URL from the drop-down list.
Attach Disk	Select the virtual machine disk that you want to attach.

7.1.1.4. Networking fields

Name	Description
Name	Name for the Network Interface Card.
Model	Indicates the model of the Network Interface Card. Supported values are e1000 , e1000e , ne2k_pci , pcnet , rtl8139 , and virtIO .
Network	List of available NetworkAttachmentDefinition objects.
Type	List of available binding methods. For the default Pod network, masquerade is the only recommended binding method. For secondary networks, use the bridge binding method. The masquerade method is not supported for non-default networks.
MAC Address	MAC address for the Network Interface Card. If a MAC address is not specified, an ephemeral address is generated for the session.


7.1.1.5. Storage fields

Name	Description
Source	Select a blank disk for the virtual machine or choose from the options available: URL , Container , Attach Cloned Disk , or Attach Disk . To select an existing disk and attach it to the virtual machine, choose Attach Cloned Disk or Attach Disk from a list of available PersistentVolumeClaims (PVCs).
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 (GiB)	Size, in GiB, of the disk.
Interface	Type of disk device. Supported interfaces are virtIO , SATA , and SCSI .
Storage Class	The StorageClass that is used to create the disk.
Advanced → Volume Mode	
Defines whether the persistent volume uses a formatted file system or raw block state. Default is Filesystem .	Advanced → Access Mode
	Access mode of the persistent volume. Supported access modes are Single User (RWO) , Shared Access (RWX) , and Read Only (ROX) .

Advanced storage settings

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

Name	Parameter	Description
Volume Mode	Filesystem	Stores the virtual disk on a filesystem-based volume.
	Block	Stores the virtual disk directly on the block volume. Only use Block if the underlying storage supports it.
Access Mode	Single User (RWO)	The disk can be mounted as read/write by a single node.

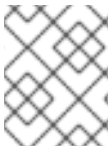
Name	Parameter	Description
	Shared Access (RWX)	<p>The disk can be mounted as read/write by many nodes.</p>  <p>NOTE</p> <p>This is required for some features, such as live migration of virtual machines between nodes.</p>
	Read Only (ROX)	<p>The disk can be mounted as read-only by many nodes.</p>

For more information on the **kubevirt-storage-class-defaults** ConfigMap, see [Storage defaults for DataVolumes](#).

7.1.2. 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 **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Click **Create Virtual Machine** and select **New from YAML**.
4. Write or paste your virtual machine configuration in the editable window.
 - a. Alternatively, use the **example** virtual machine provided by default in the YAML screen.
5. Optional: Click **Download** to download the YAML configuration file in its present state.
6. Click **Create** to create the virtual machine.

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

7.1.3. Using the CLI to create a virtual machine

Procedure

The **spec** object of the VirtualMachine configuration file references the virtual machine settings, such as the number of cores and the amount of memory, the disk type, and the volumes to use.

1. Attach the virtual machine disk to the virtual machine by referencing the relevant PVC **claimName** as a volume.
2. To create a virtual machine with the OpenShift Container Platform client, run this command:

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

3. Since virtual machines are created in a **Stopped** state, run a virtual machine instance by starting it.



NOTE

A [ReplicaSet](#)'s purpose is often used to guarantee the availability of a specified number of identical pods. ReplicaSet is not currently supported in OpenShift Virtualization.

Table 7.1. Domain settings

Setting	Description
Cores	The number of cores inside the virtual machine. Must be a value greater than or equal to 1.
Memory	The amount of RAM that is allocated to the virtual machine by the node. Specify a value in M for Megabyte or Gi for Gigabyte.
Disks	The name of the volume that is referenced. Must match the name of a volume.

Table 7.2. Volume settings

Setting	Description
Name	The name of the volume, which must be a DNS label and unique within the virtual machine.
PersistentVolumeClaim	The PVC to attach to the virtual machine. The claimName of the PVC must be in the same project as the virtual machine.

7.1.4. Virtual machine storage volume types

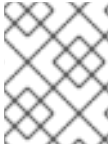
Storage volume type	Description
---------------------	-------------

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

Storage volume type	Description
containerDisk	<p>References an image, such as a virtual machine disk, that is stored in the container image registry. The image is pulled from the registry and attached to the virtual machine as a disk when the virtual machine is launched.</p> <p>A containerDisk volume is not limited to a single virtual machine and is useful for creating large numbers of virtual machine clones that do not require persistent storage.</p> <p>Only RAW and QCOW2 formats are supported disk types for the container image registry. QCOW2 is recommended for reduced image size.</p> <div data-bbox="815 757 922 1041" style="float: left; width: 60px; height: 127px; background: repeating-linear-gradient(45deg, transparent, transparent 2px, gray 2px, gray 4px); border: 1px solid gray; margin-bottom: 10px;"></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 filesystems such as CD-ROMs or for disposable virtual machines.</p>
emptyDisk	<p>Creates an additional sparse QCOW2 disk that is tied to the life-cycle of the virtual machine interface. The data survives guest-initiated reboots in the virtual machine but is discarded when the virtual machine stops or is restarted from the web console. The empty disk is used to store application dependencies and data that otherwise exceeds the limited temporary file system of an ephemeral disk.</p> <p>The disk capacity size must also be provided.</p>

7.1.5. Additional resources

The **VirtualMachineSpec** definition in the [KubeVirt v0.30.5 API Reference](#) provides broader context for the parameters and hierarchy of the virtual machine specification.

**NOTE**

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

- [Prepare a container disk](#) before adding it to a virtual machine as a **containerDisk** volume.

7.2. EDITING VIRTUAL MACHINES

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

7.2.1. Editing a virtual machine in the web console

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

Procedure

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

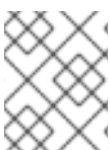
If the virtual machine is running, changes will not take effect until you reboot the virtual machine.

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

Using the web console, edit the YAML configuration of a virtual machine.

Not all parameters can be updated. If you edit values that cannot be changed and click **Save**, an error message indicates the parameter that was not able to be updated.

The YAML configuration can be edited while the virtual machine is **Running**, however the changes will only take effect after the virtual machine has been stopped and started again.

**NOTE**

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

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.

3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **YAML** tab to display the editable configuration.
5. Optional: You can click **Download** to download the YAML file locally in its current state.
6. 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.

7.2.3. 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>
```

7.2.4. Adding a virtual disk to a virtual machine

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

Procedure

1. From the **Virtual Machines** tab, select your virtual machine.
2. Select the **Disks** tab.
3. Click **Add Disks** to open the **Add Disk** window.
4. In the **Add Disk** window, specify **Source**, **Name**, **Size**, **Interface**, and **Storage Class**.

- a. Optional: In the **Advanced** list, specify the **Volume Mode** and **Access Mode** for the virtual disk. If you do not specify these parameters, the system uses the default values from the **kubevirt-storage-class-defaults** ConfigMap.
5. Use the drop-down lists and check boxes to edit the disk configuration.
6. Click **OK**.


For more information on the **kubevirt-storage-class-defaults** ConfigMap, see [Storage defaults for DataVolumes](#).

7.2.4.1. Storage fields

Name	Description
Source	Select a blank disk for the virtual machine or choose from the options available: URL , Container , Attach Cloned Disk , or Attach Disk . To select an existing disk and attach it to the virtual machine, choose Attach Cloned Disk or Attach Disk from a list of available PersistentVolumeClaims (PVCs).
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 (GiB)	Size, in GiB, of the disk.
Interface	Type of disk device. Supported interfaces are virtIO , SATA , and SCSI .
Storage Class	The StorageClass that is used to create the disk.
Advanced → Volume Mode	
Defines whether the persistent volume uses a formatted file system or raw block state. Default is Filesystem .	Advanced → Access Mode
	Access mode of the persistent volume. Supported access modes are Single User (RWO) , Shared Access (RWX) , and Read Only (ROX) .

Advanced storage settings

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

Name	Parameter	Description
Volume Mode	Filesystem	Stores the virtual disk on a filesystem-based volume.
	Block	Stores the virtual disk directly on the block volume. Only use Block if the underlying storage supports it.
Access Mode	Single User (RWO)	The disk can be mounted as read/write by a single node.
	Shared Access (RWX)	The disk can be mounted as read/write by many nodes.  NOTE This is required for some features, such as live migration of virtual machines between nodes.
	Read Only (ROX)	The disk can be mounted as read-only by many nodes.

7.2.5. Adding a network interface to a virtual machine

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

Procedure

1. From the **Virtual Machines** tab, select the virtual machine.
2. Select the **Network Interfaces** tab.
3. Click **Add Network Interface**.
4. In the **Add Network Interface** window, specify the **Name**, **Model**, **Network**, **Type**, and **MAC Address** of the network interface.
5. Click **Add** to add the network interface.
6. Restart the virtual machine to enable access.
7. Edit the drop-down lists and check boxes to configure the network interface.
8. Click **Save Changes**.
9. Click **OK**.

The new network interface displays at the top of the **Create Network Interface** list until the user restarts it.

The new network interface has a **Pending VM restart** Link State until you restart the virtual machine. Hover over the Link State to display more detailed information.

The **Link State** is set to **Up** by default when the network interface card is defined on the virtual machine and connected to the network.

7.2.5.1. Networking fields

Name	Description
Name	Name for the Network Interface Card.
Model	Indicates the model of the Network Interface Card. Supported values are e1000 , e1000e , ne2k_pci , pcnet , rtl8139 , and virtIO .
Network	List of available NetworkAttachmentDefinition objects.
Type	List of available binding methods. For the default Pod network, masquerade is the only recommended binding method. For secondary networks, use the bridge binding method. The masquerade method is not supported for non-default networks.
MAC Address	MAC address for the Network Interface Card. If a MAC address is not specified, an ephemeral address is generated for the session.

7.2.6. Editing CD-ROMs for Virtual Machines

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

Procedure

1. From the **Virtual Machines** tab, select your virtual machine.
2. Select the **Overview** tab.
3. To add or edit a CD-ROM configuration, click the pencil icon to the right of the **CD-ROMs** label. The **Edit CD-ROM** window opens.
 - If CD-ROMs are unavailable for editing, the following message displays: **The virtual machine doesn't have any CD-ROMs attached.**
 - If there are CD-ROMs available, you can remove a CD-ROM by clicking **-**.
4. In the **Edit CD-ROM** window, do the following:
 - a. Select the type of CD-ROM configuration from the drop-down list for **Media Type**. CD-ROM configuration types are **Container**, **URL**, and **Persistent Volume Claim**
 - b. Complete the required information for each **Type**.
 - c. When all CD-ROMs are added, click **Save**.

7.3. 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 Card (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.

7.3.1. Adding items to a boot order list in the web console

Add items to a boot order list by using the web console.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Details** tab.
5. Click the pencil icon that is located on the right side of **Boot Order**. If a YAML configuration does not exist, or if this is the first time that you are creating a boot order list, the following message displays: **No resource selected. VM will attempt to boot disks from YAML by order of appearance in YAMLv file. Please select a boot source.**
6. Click **Add Source** and select a bootable disk or Network Interface Card (NIC) for the virtual machine.
7. Add any additional disks or NICs to the boot order list.
8. Click **Save**.

7.3.2. Editing a boot order list in the web console

Edit the boot order list in the web console.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Details** tab.
5. Click the pencil icon that is located on the right side of **Boot Order**.
6. Choose the appropriate method to move the item in the boot order list:

- If you do not use a screen reader, hover over the arrow icon next to the item that you want to move, drag the item up or down, and drop it in a location of your choice.
- If you use a screen reader, press the Up Arrow key or Down Arrow key to move the item in the boot order list. Then, press the **Tab** key to drop the item in a location of your choice.

7. Click **Save**.

7.3.3. 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 Card (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 Card.

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.

7.3.4. Removing items from a boot order list in the web console

Remove items from a boot order list by using the web console.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Details** tab.
5. Click the pencil icon that is located on the right side of **Boot Order**.
6. Click the **Remove** icon next to the item. The item is removed from the boot order list and saved in the list of available boot sources. If you remove all items from the boot order list, the following message displays: **No resource selected. VM will attempt to boot disks from YAML by order of appearance in YAML file. Please select a boot source.**

7.4. DELETING VIRTUAL MACHINES

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

7.4.1. Deleting a virtual machine using the web console

Deleting a virtual machine permanently removes it from the cluster.



NOTE

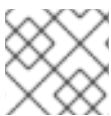
When you delete a virtual machine, the DataVolume it uses is automatically deleted.

Procedure

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

7.4.2. 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 DataVolume it uses is automatically deleted.

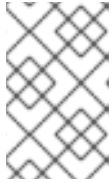
Prerequisites

- 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.

7.5. MANAGING VIRTUAL MACHINE INSTANCES

If you have standalone virtual machine instances (VMIs) that were created independently outside of the OpenShift Virtualization environment, you can manage them by using the web console or the command-line interface (CLI).

7.5.1. 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.

7.5.2. 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
```

7.5.3. Listing standalone virtual machine instances using the web console

Using the web console, you can list and view standalone virtual machine instances (VMIs) in your cluster that are not owned by virtual machines (VMs).



NOTE

VMIs that are owned by VMs or other objects are not displayed in the web console. The web console displays only standalone VMIs. If you want to list all VMIs in your cluster, you must use the CLI.

Procedure

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

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

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

Procedure

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



NOTE

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

7.5.5. 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>
```


7.5.6. Deleting a standalone virtual machine instance using the web console

Delete a standalone virtual machine instance (VMI) from the web console.

Procedure

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

7.6. CONTROLLING VIRTUAL MACHINE STATES

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




NOTE

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

7.6.1. Starting a virtual machine

You can start a virtual machine from the web console.

Procedure

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

- In the confirmation window, click **Start** to start the virtual machine.

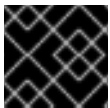


NOTE

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.

7.6.2. 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


- Click **Workloads** → **Virtualization** from the side menu.
- Click the **Virtual Machines** tab.
- Find the row that contains the virtual machine that you want to restart.
- Navigate to the appropriate menu for your use case:
 - To stay on this page, where you can perform actions on multiple virtual machines:
 - 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:
 - Access the **Virtual Machine Overview** screen by clicking the name of the virtual machine.
 - Click **Actions**.
- Select **Restart Virtual Machine**
- In the confirmation window, click **Restart** to restart the virtual machine.

7.6.3. Stopping a virtual machine

You can stop a virtual machine from the web console.

Procedure

- Click **Workloads** → **Virtualization** from the side menu.
- Click the **Virtual Machines** tab.
- Find the row that contains the virtual machine that you want to stop.

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

7.6.4. Unpausing a virtual machine

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

Prerequisites

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



NOTE

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

Procedure

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

7.7. 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 web console and by using CLI commands.

7.7.1. Virtual machine console sessions

You can connect to the VNC and serial consoles of a running virtual machine from the **Consoles** tab in the **Virtual Machine Details** screen of the web console.

There are two consoles available: the graphical **VNC Console** and the **Serial Console**. The **VNC Console** opens by default whenever you navigate to the **Consoles** tab. You can switch between the consoles using the **VNC Console Serial Console** list.

Console sessions remain active in the background unless they are disconnected. When the **Disconnect before switching** checkbox is active and you switch consoles, the current console session is disconnected and a new session with the selected console connects to the virtual machine. This ensures only one console session is open at a time.

Options for the VNC Console

The **Send Key** button lists key combinations to send to the virtual machine.

Options for the Serial Console

Use the **Disconnect** button to manually disconnect the **Serial Console** session from the virtual machine. Use the **Reconnect** button to manually open a **Serial Console** session to the virtual machine.

7.7.2. Connecting to the virtual machine with the web console

7.7.2.1. Connecting to the terminal

You can connect to a virtual machine by using the web console.

Procedure

1. Ensure you are in the correct project. If not, click the **Project** list and select the appropriate project.
2. Click **Workloads** → **Virtualization** from the side menu.
3. Click the **Virtual Machines** tab.
4. Select a virtual machine to open the **Virtual Machine Overview** screen.
5. In the **Details** tab, click the **virt-launcher-`<vm-name>`** pod.
6. Click the **Terminal** tab. If the terminal is blank, select the terminal and press any key to initiate connection.

7.7.2.2. Connecting to the serial console

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

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click **Console**. The VNC console opens by default.
5. Click the **VNC Console** drop-down list and select **Serial Console**.

7.7.2.3. Connecting to the VNC console

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

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Console** tab. The VNC console opens by default.

7.7.2.4. Connecting to the RDP console

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

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

Prerequisites

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

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a Windows virtual machine to open the **Virtual Machine Overview** screen.
4. Click the **Console** tab.
5. In the **Console** list, select **Desktop Viewer**.
6. In the **Network Interface** list, select the layer-2 NIC.

7. Click **Launch Remote Desktop** to download the **console.rdp** file.
8. Open an RDP client and reference the **console.rdp** file. For example, using **remmina**:


```
$ remmina --connect /path/to/console.rdp
```
9. Enter the **Administrator** user name and password to connect to the Windows virtual machine.

7.7.3. Accessing virtual machine consoles by using CLI commands

7.7.3.1. Accessing a virtual machine instance via SSH

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

The **virtctl expose** command forwards a virtual machine instance port to a node port and creates a service for enabled access. The following example creates the **fedora-vm-ssh** service that forwards port 22 of the **<fedora-vm>** virtual machine to a port on the node:

Prerequisites

- You must be in the same project as the virtual machine instance.
- The virtual machine instance you want to access must be connected to the default Pod network by using the **masquerade** binding method.
- The virtual machine instance 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=20022 --target-port=22 --name=fedora-vm-ssh --
type=NodePort 1
```

- 1** **<fedora-vm>** is the name of the virtual machine 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>       20022:32551/TCP 6s
```

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

1. Log in to the virtual machine instance via SSH. Use the **ipAddress** of the node and the port that you found in the previous step:

■

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

7.7.3.2. 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**:

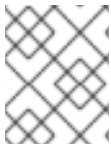
```
$ virtctl console <VMI>
```

7.7.3.3. 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
```

7.7.3.4. 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.

7.8. MANAGING CONFIGMAPS, SECRETS, AND SERVICE ACCOUNTS IN VIRTUAL MACHINES

You can use secrets, ConfigMaps, 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 ConfigMap 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, ConfigMaps, and service accounts as virtual machine disks so that you can use them across platforms without additional overhead.

7.8.1. Adding a secret, ConfigMap, or service account to a virtual machine

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

Prerequisites

- The secret, ConfigMap, or service account that you want to add must exist in the same namespace as the target virtual machine.
- The virtual machine must be powered off.

Procedure

1. From the side menu, click **Virtualization**.
2. Click the **Virtual Machine** tab.
3. Select a virtual machine to open its **Virtual Machine Overview** page.
4. Click the **Environment** tab.
5. Click **Select a resource** and select a secret, ConfigMap, or service account from the list.
6. Click **Save**.
7. Optional. Add another object by clicking **Add Config Map, Secret or Service Account**



NOTE

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

Verification

1. From the **Virtual Machine Overview** page, click the **Disks** tab.

2. Check to ensure that the secret, ConfigMap, or service account is included in the list of disks.
3. Optional. Start the virtual machine by clicking **Actions** → **Start Virtual Machine**. You can now mount the secret, ConfigMap, or service account as you would mount any other disk.


7.8.2. Removing a secret, ConfigMap, or service account from a virtual machine

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

Prerequisites

- You must have at least one secret, ConfigMap, or service account that is attached to a virtual machine.
- The virtual machine must be powered off.

Procedure

1. From the side menu, click **Virtualization**.
2. Click the **Virtual Machine** tab.
3. Select a virtual machine to open its **Virtual Machine Overview** page.
4. Click the **Environment** tab.
5. Find the item that you want to delete in the list, and click the **Delete** button  on the right side of the item.
6. Click **Save**.



NOTE

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

Verification

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

7.8.3. Additional resources

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

7.9. INSTALLING VIRTIO DRIVER ON AN EXISTING WINDOWS VIRTUAL MACHINE

7.9.1. Understanding VirtIO drivers

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

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

After the drivers are installed, the **container-native-virtualization/virtio-win** container disk can be removed from the virtual machine.

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

7.9.2. Supported VirtIO drivers for Microsoft Windows virtual machines

Table 7.3. 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.
viornng	VEN_1AF4&DEV_1005 VEN_1AF4&DEV_1044	The entropy source driver. Sometimes displays as a PCI Device in the Other devices group.
NetKVM	VEN_1AF4&DEV_1000 VEN_1AF4&DEV_1041	The network driver. Sometimes displays as an Ethernet Controller in the Other devices group. Available only if a VirtIO NIC is configured.

7.9.3. Adding VirtIO drivers container disk to a virtual machine

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

Prerequisites

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

Procedure

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

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

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

2. The disk is available once the virtual machine has started:

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

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

7.9.4. 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. Refer to 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.

7.9.5. 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.

7.10. INSTALLING VIRTIO DRIVER ON A NEW WINDOWS VIRTUAL MACHINE

7.10.1. 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.

7.10.2. Understanding VirtIO drivers

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

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

After the drivers are installed, the **container-native-virtualization/virtio-win** container disk can be removed from the virtual machine.

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

7.10.3. Supported VirtIO drivers for Microsoft Windows virtual machines

Table 7.4. 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.
viornng	VEN_1AF4&DEV_1005 VEN_1AF4&DEV_1044	The entropy source driver. Sometimes displays as a PCI Device in the Other devices group.
NetKVM	VEN_1AF4&DEV_1000 VEN_1AF4&DEV_1041	The network driver. Sometimes displays as an Ethernet Controller in the Other devices group. Available only if a VirtIO NIC is configured.

7.10.4. 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.

7.10.5. 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. Refer to 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.

7.10.6. 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.

7.11. ADVANCED VIRTUAL MACHINE MANAGEMENT

7.11.1. 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.

7.11.1.1. 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.

7.11.1.2. 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.

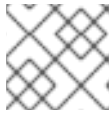
Prerequisites

- [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 ❶
  cpu_cores: 1
```

- ❶ 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
```

-
- 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
```

- 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](#).

7.11.1.3. 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](#)

7.11.2. 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.

7.11.2.1. Prerequisites

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

7.11.2.2. 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.

NetworkAttachmentDefinition

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

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.

7.11.2.3. 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 NetworkAttachmentDefinition 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 NetworkAttachmentDefinition file for PXE network **pxe-net-conf**:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: pxe-net-conf
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "pxe-net-conf",
    "plugins": [
      {
        "type": "cnv-bridge",
        "bridge": "br1",
        "vlan": 1 1
      },
      {
        "type": "cnv-tuning" 2
      }
    ]
  }'
```

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



NOTE

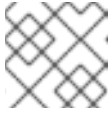
The virtual machine instance will be attached to the bridge **br1** through an access port with the requested VLAN.

2. Create the NetworkAttachmentDefinition object 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. However, note that at this time, MAC addresses assigned automatically are not persistent. 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 NetworkAttachmentDefinition. In this scenario, **<pxe-net>** is connected to the NetworkAttachmentDefinition 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
```

7.11.2.4. Template: virtual machine instance configuration file for PXE booting

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstance
metadata:
  creationTimestamp: null
  labels:
    special: vmi-pxe-boot
  name: vmi-pxe-boot
spec:
  domain:
    devices:
      disks:
        - disk:
            bus: virtio
            name: containerdisk
            bootOrder: 2
        - disk:
            bus: virtio
            name: cloudinitdisk
      interfaces:
        - masquerade: {}
          name: default
        - bridge: {}
          name: pxe-net
          macAddress: de:00:00:00:00:de
          bootOrder: 1
      machine:
        type: ""
    resources:
      requests:
        memory: 1024M
    networks:
      - name: default
        pod: {}
      - multus:
          networkName: pxe-net-conf
          name: pxe-net
  terminationGracePeriodSeconds: 0
  volumes:
    - name: containerdisk
      containerDisk:
        image: kubevirt/fedora-cloud-container-disk-demo
    - cloudInitNoCloud:
```

```

userData: |
  #!/bin/bash
  echo "fedora" | passwd fedora --stdin
name: cloudinitdisk
status: {}

```

7.11.3. Managing guest memory

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



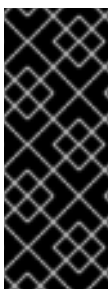
WARNING

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

7.11.3.1. Configuring guest memory overcommitment

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

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



IMPORTANT

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

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

Procedure

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

```

kind: VirtualMachine
spec:
  template:
    domain:

```

```
resources:
  requests:
    memory: 1024M
memory:
  guest: 2048M
```

**NOTE**

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

2. Create the virtual machine:

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

7.11.3.2. Disabling guest memory overhead accounting

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

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

**IMPORTANT**

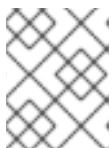
Disabling guest memory overhead accounting increases the potential for virtual machine processes to be killed due to memory pressure (OOM killed).

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

Procedure

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

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

**NOTE**

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

2. Create the virtual machine:


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

7.11.4. Using huge pages with virtual machines

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

7.11.4.1. Prerequisites

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

7.11.4.2. 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. In order 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.

7.11.4.3. 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
```

7.11.5. Enabling dedicated resources for virtual machines

Virtual machines can have resources of a node, such as CPU, dedicated to them in order to improve performance.

7.11.5.1. 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.

7.11.5.2. Prerequisites

- The [CPU Manager](#) must be configured on the node. Verify that the node has the **cpumanager = true** label before scheduling virtual machine workloads.

7.11.5.3. Enabling dedicated resources for a virtual machine

You can enable dedicated resources for a virtual machine in the **Virtual Machine Overview** page of the web console.

Procedure

1. Click **Workloads** → **Virtual Machines** from the side menu.

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

7.12. IMPORTING VIRTUAL MACHINES

7.12.1. TLS certificates for DataVolume imports

7.12.1.1. Adding TLS certificates for authenticating DataVolume imports

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

Create the ConfigMap by referencing the relative file path for the TLS certificate.

Procedure

1. Ensure you are in the correct namespace. The ConfigMap can only be referenced by DataVolumes if it is in the same namespace.

```
$ oc get ns
```

2. Create the ConfigMap:

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

7.12.1.2. Example: ConfigMap created from a TLS certificate

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

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

7.12.2. Importing virtual machine images with DataVolumes

Use the Containerized Data Importer (CDI) to import a virtual machine image into a PersistentVolumeClaim (PVC) by using a DataVolume. You can attach a DataVolume 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. Refer to the operating system documentation for details.

7.12.2.1. Prerequisites

- If the endpoint requires a TLS certificate, the certificate must be [included in a ConfigMap](#) in the same namespace as the DataVolume and referenced in the DataVolume 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 cdi-insecure-registries ConfigMap](#) before you can import a container disk from it.
- You might need to [define a StorageClass or prepare CDI scratch space](#) for this operation to complete successfully.

7.12.2.2. CDI supported operations matrix

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

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

✓ Supported operation

Unsupported operation

* Requires scratch space

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

7.12.2.3. About DataVolumes

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

7.12.2.4. Importing a virtual machine image into a PersistentVolumeClaim by using a DataVolume

You can import a virtual machine image into a PersistentVolumeClaim (PVC) by using a DataVolume.

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.

To create a virtual machine from an imported virtual machine image, specify the image or container disk endpoint in the **VirtualMachine** configuration file before you create the virtual machine.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- Your cluster has at least one available PersistentVolume.
- 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 endpoint where the image is hosted, along with any authentication credentials needed to access the data source. For example: <http://www.example.com/path/to/data>
- To import a container disk you must have the following:
 - A container disk built from a virtual machine image stored in your container image registry, along with any authentication credentials needed to access the data source. For example: **docker://registry.example.com/container-image**

Procedure

1. Optional: If your data source requires authentication credentials, edit the **endpoint-secret.yaml** file, and apply the updated configuration to the cluster:

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

- 1** Optional: your key or user name, base64 encoded
- 2** Optional: your secret or password, base64 encoded

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

2. Edit the virtual machine configuration file, specifying the data source for the virtual machine image you want to import. In this example, a Fedora image is imported from an **http** source:

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  creationTimestamp: null
  labels:
    kubevirt.io/vm: vm-fedora-datavolume
  name: vm-fedora-datavolume
spec:
  dataVolumeTemplates:
  - metadata:
    creationTimestamp: null
    name: fedora-dv
    spec:
      pvc:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 10Gi
        storageClassName: local
      source:
        http: 1
        url:
          "https://download.fedoraproject.org/pub/fedora/linux/releases/33/Cloud/x86_64/images/Fedora-
          Cloud-Base-33-1.2.x86_64.qcow2" 2
        secretRef: "" 3
        certConfigMap: "" 4
      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: "" 5
          resources:
            requests:
              memory: 1.5Gi
          terminationGracePeriodSeconds: 60
        volumes:
        - dataVolume:
```

```

name: fedora-dv
name: datavolumedisk1
status: {}

```

- 1 The source type to import the image from. This example uses an HTTP endpoint. To import a container disk from a registry, replace **http** with **registry**.
- 2 The source of the virtual machine image you want to import. This example references a virtual machine image at an HTTP endpoint. An example of a container registry endpoint is **url: "docker://kubevirt/fedora-cloud-container-disk-demo:latest"**.
- 3 The **secretRef** parameter is optional.
- 4 The **certConfigMap** is required for communicating with servers that use self-signed certificates or certificates not signed by the system CA bundle. The referenced ConfigMap must be in the same namespace as the DataVolume.
- 5 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.

3. Create the virtual machine:

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



NOTE

The **oc create** command creates the DataVolume and the virtual machine. The CDI controller creates an underlying PVC with the correct annotation, and the import process begins. When the import completes, the DataVolume status changes to **Succeeded**, and the virtual machine is allowed to start.

DataVolume provisioning happens in the background, so there is no need to monitor it. You can start the virtual machine, and it will not run until the import is complete.

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 DataVolume status until it shows **Succeeded** by running the following command:

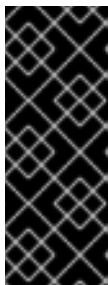
```
$ oc describe dv <datavolume-name> 1
```

- 1 The name of the DataVolume as specified under **dataVolumeTemplates.metadata.name** in the virtual machine configuration file. In the example configuration above, this is **fedora-dv**.
3. To verify that provisioning is complete and that the VMI has started, try accessing its serial console by running the following command:

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

7.12.3. Importing virtual machine images to block storage with DataVolumes

You can import an existing virtual machine image into your OpenShift Container Platform cluster. OpenShift Virtualization uses DataVolumes to automate the import of data and the creation of an underlying PersistentVolumeClaim (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. Refer to the operating system documentation for details.

7.12.3.1. Prerequisites

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

7.12.3.2. About DataVolumes

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

7.12.3.3. About block PersistentVolumes

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

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

7.12.3.4. Creating a local block PersistentVolume

Create a local block PersistentVolume (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 configuration 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** configuration 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 StorageClass 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 filename of the PersistentVolume created in the previous step.

7.12.3.5. Importing a virtual machine image to a block PersistentVolume using DataVolumes

You can import an existing virtual machine image into your OpenShift Container Platform cluster. OpenShift Virtualization uses DataVolumes to automate the importing data and the creation of an

underlying PersistentVolumeClaim (PVC). You can then reference the DataVolume in a virtual machine configuration.

Prerequisites

- A virtual machine disk image, in RAW, ISO, or QCOW2 format, optionally compressed by using **xz** or **gz**.
- An **HTTP** or **s3** endpoint where the image is hosted, along with any authentication credentials needed to access the data source
- At least one available block PV.

Procedure

1. If your data source requires authentication credentials, edit the **endpoint-secret.yaml** file, and apply the updated configuration to the cluster.

- a. Edit the **endpoint-secret.yaml** file with your preferred text editor:

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

- 1** Optional: your key or user name, base64 encoded
- 2** Optional: your secret or password, base64 encoded

- b. Update the secret by running the following command:

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

2. Create a **DataVolume** configuration that specifies the data source for the image you want to import and **volumeMode: Block** so that an available block PV is used.

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: <import-pv-datavolume> 1
spec:
  storageClassName: local 2
  source:
    http:
      url:
        <http://download.fedoraproject.org/pub/fedora/linux/releases/28/Cloud/x86_64/images/Fedora-Cloud-Base-28-1.1.x86_64.qcow2> 3
      secretRef: <endpoint-secret> 4
```

```
pvc:
  volumeMode: Block 5
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: <2Gi>
```

- 1** The name of the DataVolume.
- 2** Optional: Set the storage class or omit it to accept the cluster default.
- 3** The **HTTP** source of the image to import.
- 4** Only required if the data source requires authentication.
- 5** Required for importing to a block PV.

3. Create the DataVolume to import the virtual machine image by running the following command:

```
$ oc create -f <import-pv-datavolume.yaml> 1
```

- 1** The file name of the DataVolume that you created in the previous step.

7.12.3.6. CDI supported operations matrix

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

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

✓ Supported operation

Unsupported operation

* Requires scratch space

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

7.12.4. Importing a single Red Hat Virtualization virtual machine

You can import a single Red Hat Virtualization (RHV) virtual machine into your OpenShift Container Platform cluster by using the virtual machine wizard or the CLI.

7.12.4.1. OpenShift Virtualization storage feature matrix

The following table describes local and shared persistent storage that support VM import.

Table 7.5. OpenShift Virtualization storage feature matrix

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

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

7.12.4.2. Prerequisites for importing a virtual machine

Importing a virtual machine into OpenShift Virtualization has the following prerequisites:

- You must have admin user privileges.
- Storage:
 - The OpenShift Virtualization local and shared persistent storage classes must support VM import.
 - If you are using Ceph RBD block-mode volumes, the storage must be large enough to accommodate the virtual disk. If the disk is too large for the available storage, the import process fails and the PV that is used to copy the virtual disk is not released.
- Networks:
 - The source and target networks must either have the same name or be mapped to each other.
 - The source network interface must be **e1000**, **rtl8139**, or **virtio**.
- VM disks:
 - The disk interface must be **sata**, **virtio_scsi**, or **virtio**.
 - The disk must not be configured as a direct LUN.
 - The disk status must not be **illegal** or **locked**.

- The storage type must be **image**.
- SCSI reservation must be disabled.
- **ScsiGenericIO** must be disabled.
- VM configuration:
 - If the VM uses GPU resources, the nodes providing the GPUs must be configured.
 - The VM must not be configured for vGPU resources.
 - The VM must not have snapshots with disks in an **illegal** state.
 - The VM must not have been created with OpenShift Container Platform and subsequently added to RHV.
 - The VM must not be configured for USB devices.
 - The watchdog model must not be **diag288**.

7.12.4.3. Checking the default storage class

You must check the default storage class to ensure that it is NFS.



Cinder, the default storage class, does not support VM import.


7.12.4.3.1. Checking the default storage class in the OpenShift Container Platform console

You can check the default storage class in the OpenShift Container Platform console. If the default storage class is not NFS, you can change the default storage class so that it is no longer the default and change the NFS storage class so that it is the default.

If more than one default storage class is defined, the VirtualMachineImport CR uses the default storage class that is first in alphabetical order.

Procedure

1. Navigate to **Storage → Storage Classes**.
2. Check the default storage class in the **Storage Classes** list.
3. If the default storage class is not NFS, edit the default storage class so that it is no longer the default:
 - a. Click the Options menu  of the default storage class and select **Edit Storage Class**
 - b. In the **Details** tab, click the Edit button beside **Annotations**.
 - c. Click the Delete button  on the right side of the **storageclass.kubernetes.io/is-default-class** annotation and then click **Save**.
4. Change an existing NFS storage class to be the default:

- a. Click the Options menu  of an existing NFS storage class and select **Edit Storage Class**.
 - b. In the **Details** tab, click the Edit button beside **Annotations**.
 - c. Enter **storageclass.kubernetes.io/is-default-class** in the **Key** field and **true** in the **Value** field and then click **Save**.
5. Navigate to **Storage** → **Storage Classes** to verify that the NFS storage class is the only default storage class.

7.12.4.3.2. Checking the default storage class from the CLI

You can check the default storage class from the CLI.

If the default storage class is not NFS, you must change the default storage class to NFS and change the existing default storage class so that it is not the default. If more than one default storage class is defined, the VirtualMachineImport CR uses the default storage class that is first in alphabetical order.

Procedure

- Get the storage classes by entering the following command:

```
$ oc get sc
```

The **default** storage class is displayed in the output:

Example output

```
NAME                PROVISIONER          RECLAIMPOLICY  VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANS
...
standard (default)  kubernetes.io/cinder Delete          WaitForFirstConsumer true
```

Changing the default storage class

If you are using AWS, use the following process to change the default storage class. This process assumes you have two storage classes defined, **gp2** and **standard**, and you want to change the default storage class from **gp2** to **standard**.

1. List the storage class:

```
$ oc get storageclass
```

Example output

```
NAME                TYPE
gp2 (default)       kubernetes.io/aws-ebs 1
standard            kubernetes.io/aws-ebs
```

1 **(default)** denotes the default storage class.

2. Change the value of the annotation **storageclass.kubernetes.io/is-default-class** to **false** for the default storage class:

```
$ oc patch storageclass gp2 -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "false"}}}'
```

3. Make another storage class the default by adding or modifying the annotation as **storageclass.kubernetes.io/is-default-class=true**.

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

4. Verify the changes:

```
$ oc get storageclass
```

Example output

```
NAME                TYPE
gp2                 kubernetes.io/aws-ebs
standard (default) kubernetes.io/aws-ebs
```

7.12.4.4. Creating a ConfigMap for importing a Red Hat Virtualization virtual machine

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

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

Table 7.6. Operating system and template mapping

RHV VM operating system	OpenShift Virtualization template
rhel_6_9_plus_ppc64	rhel6.9
rhel_6_ppc64	rhel6.9
rhel_6	rhel6.9
rhel_6x64	rhel6.9
rhel_7_ppc64	rhel7.7
rhel_7_s390x	rhel7.7
rhel_7x64	rhel7.7
rhel_8x64	rhel8.1

RHV VM operating system	OpenShift Virtualization template
sles_11_ppc64	opensuse15.0
sles_11	opensuse15.0
sles_12_s390x	opensuse15.0
ubuntu_12_04	ubuntu18.04
ubuntu_12_10	ubuntu18.04
ubuntu_13_04	ubuntu18.04
ubuntu_13_10	ubuntu18.04
ubuntu_14_04_ppc64	ubuntu18.04
ubuntu_14_04	ubuntu18.04
ubuntu_16_04_s390x	ubuntu18.04
windows_10	win10
windows_10x64	win10
windows_2003	win10
windows_2003x64	win10
windows_2008R2x64	win2k8
windows_2008	win2k8
windows_2008x64	win2k8
windows_2012R2x64	win2k12r2
windows_2012x64	win2k12r2
windows_2016x64	win2k16
windows_2019x64	win2k19
windows_7	win10
windows_7x64	win10

RHV VM operating system	OpenShift Virtualization template
windows_8	win10
windows_8x64	win10
windows_xp	win10

Procedure

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

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

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

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

Example output

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

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

```
$ cat <<EOF | oc create -f -
apiVersion: v1
kind: ConfigMap
metadata:
  name: os-configmap
  namespace: default ❶
data:
  guestos2common: |
    "Red Hat Enterprise Linux Server": "rhel"
    "CentOS Linux": "centos"
    "Fedora": "fedora"
```

```
"Ubuntu": "ubuntu"
"openSUSE": "opensuse"
osinfo2common: |
  "<rhv-operating-system>": "<vm-template>" 2
EOF
```

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

ConfigMap example

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

5. Verify that the custom ConfigMap was created:

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

6. Edit the **kubevirt-hyperconverged-operator.v2.4.9.yaml** file:

```
$ oc edit clusterserviceversion -n openshift-cnv kubevirt-hyperconverged-operator.v2.4.9
```

7. Update the following parameters of the **vm-import-operator** deployment manifest:

```
...
spec:
  containers:
  - env:
    ...
    - name: OS_CONFIGMAP_NAME
      value: os-configmap 1
    - name: OS_CONFIGMAP_NAMESPACE
      value: default 2
```

- 1 Add **value: os-configmap** to the **name: OS_CONFIGMAP_NAME** parameter.
 - 2 Optional: You can add this value if you changed the namespace in the ConfigMap.
8. Save the **kubevirt-hyperconverged-operator.v2.4.9.yaml** file.
 - Updating the **vm-import-operator** deployment updates the **vm-import-controller** ConfigMap.
 9. Verify that the template appears in the OpenShift Virtualization web console:

- a. Click **Workloads** → **Virtualization** from the side menu.
- b. Click the **Virtual Machine Templates** tab and find the template in the list.

7.12.4.5. Importing a virtual machine with the VM Import wizard

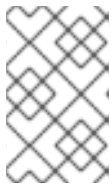
You can import a single virtual machine with the VM Import wizard.

Procedure

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


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

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



NOTE

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

6. Select a cluster and a virtual machine.
7. Click **Next**.
8. In the **Review** screen, review your settings.
9. Optional: You can select **Start virtual machine on creation**
10. Click **Edit** to update the following settings:
 - **General** → **Name:** The VM name is limited to 63 characters.

- **General → Description:** Optional description of the VM.
 - **Storage Class:** Select **NFS** or **ocs-storagecluster-ceph-rbd**.
If you select **ocs-storagecluster-ceph-rbd**, you must set the **Volume Mode** of the disk to **Block**.
 - **Advanced → Volume Mode:** Select **Block**.
 - **Advanced → Volume Mode:** Select **Block**.
 - **Networking → Network:** You can select a network from a list of available network attachment definition objects.
11. Click **Import** or **Review and Import**, if you have edited the import settings.
A **Successfully created virtual machine** message and a list of resources created for the virtual machine are displayed. The virtual machine appears in **Workloads → Virtual Machines**.

Virtual machine wizard fields

Name	Parameter	Description
Template		Template from which to create the virtual machine. Selecting a template will automatically complete other fields.
Source	PXE	Provision virtual machine from PXE menu. Requires a PXE-capable NIC in the cluster.
	URL	Provision virtual machine from an image available from an HTTP or S3 endpoint.
	Container	Provision virtual machine from a bootable operating system container located in a registry accessible from the cluster. Example: kubevirt/cirros-registry-disk-demo .
	Disk	Provision virtual machine from a disk.
Operating System		The primary operating system that is selected for the virtual machine.
Flavor	small, medium, large, tiny, Custom	Presets that determine the amount of CPU and memory allocated to the virtual machine. The presets displayed for Flavor are determined by the operating system.

Name	Parameter	Description
Memory		Size in GiB of the memory allocated to the virtual machine.
CPUs		The amount of CPU allocated to the virtual machine.
Workload Profile	High Performance	A virtual machine configuration that is optimized for high-performance workloads.
	Server	A profile optimized to run server workloads.
	Desktop	A virtual machine configuration for use on a desktop.
Name		The name can contain lowercase letters (a-z), numbers (0-9), and hyphens (-), up to a maximum of 253 characters. The first and last characters must be alphanumeric. The name must not contain uppercase letters, spaces, periods (.), or special characters.
Description		Optional description field.
Start virtual machine on creation		Select to automatically start the virtual machine upon creation.

Networking fields

Name	Description
Name	Name for the Network Interface Card.
Model	Indicates the model of the Network Interface Card. Supported values are e1000 , e1000e , ne2k_pci , pcnet , rtl8139 , and virtIO .
Network	List of available NetworkAttachmentDefinition objects.

Name	Description
Type	List of available binding methods. For the default Pod network, masquerade is the only recommended binding method. For secondary networks, use the bridge binding method. The masquerade method is not supported for non-default networks.
MAC Address	MAC address for the Network Interface Card. If a MAC address is not specified, an ephemeral address is generated for the session.

Storage fields

Name	Description
Source	Select a blank disk for the virtual machine or choose from the options available: URL , Container , Attach Cloned Disk , or Attach Disk . To select an existing disk and attach it to the virtual machine, choose Attach Cloned Disk or Attach Disk from a list of available PersistentVolumeClaims (PVCs).
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 (GiB)	Size, in GiB, of the disk.
Interface	Type of disk device. Supported interfaces are virtIO , SATA , and SCSI .
Storage Class	The StorageClass that is used to create the disk.
Advanced → Volume Mode	

Advanced storage settings

Name	Parameter	Description
Volume Mode	Filesystem	Stores the virtual disk on a filesystem-based volume.
	Block	Stores the virtual disk directly on the block volume. Only use Block if the underlying storage supports it.

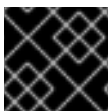
Name	Parameter	Description
Access Mode [1]	Single User (RWO)	The disk can be mounted as read/write by a single node.
	Shared Access (RWX)	The disk can be mounted as read/write by many nodes.
	Read Only (ROX)	The disk can be mounted as read-only by many nodes.

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

7.12.4.6. Importing a Red Hat Virtualization virtual machine with the CLI

You can import a Red Hat Virtualization (RHV) virtual machine with the CLI by creating the Secret and VirtualMachineImport Custom Resources (CRs). The Secret CR stores the RHV Manager credentials and CA certificate. The VirtualMachineImport CR defines the parameters of the VM import process.

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



IMPORTANT

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

Procedure

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

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

- 1 Optional. You can specify a different namespace in all the CRs.
- 2 Specify the API endpoint of the RHV Manager, for example, `"https://www.example.com:8443/ovirt-engine/api"`

- 3 Specify the password for **admin@internal**.
- 4 Specify the RHV Manager CA certificate. You can obtain the CA certificate by running the following command:

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

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

```
$ cat <<EOF | kubectl create -f -
apiVersion: v2v.kubevirt.io/v1alpha1
kind: ResourceMapping
metadata:
  name: resourcemapping_example
  namespace: default
spec:
  ovirt:
    networkMappings:
      - source:
          name: <rhv_logical_network>/<vnic_profile> 1
        target:
          name: <target_network> 2
        type: pod
    storageMappings: 3
      - source:
          name: <rhv_storage_domain> 4
        target:
          name: <target_storage_class> 5
          volumeMode: <volume_mode> 6
EOF
```

- 1 Specify the RHV logical network and vNIC profile.
- 2 Specify the OpenShift Virtualization network.
- 3 If storage mappings are specified in both the **ResourceMapping** and the **VirtualMachineImport** CRs, the **VirtualMachineImport** CR takes precedence.
- 4 Specify the RHV storage domain.
- 5 Specify **nfs** or **ocs-storagecluster-ceph-rbd**.
- 6 If you specified the **ocs-storagecluster-ceph-rbd** storage class, you must specify **Block** as the volume mode.

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

```
$ cat <<EOF | oc create -f -
apiVersion: v2v.kubevirt.io/v1alpha1
kind: VirtualMachineImport
metadata:
  name: vm-import
```



```

namespace: default
spec:
  providerCredentialsSecret:
    name: rhv-credentials
    namespace: default
  # resourceMapping: 1
  # name: resourcemapping-example
  # namespace: default
  targetVmName: vm_example 2
  startVm: true
  source:
    ovirt:
      vm:
        id: <source_vm_id> 3
        name: <source_vm_name> 4
      cluster:
        name: <source_cluster_name> 5
    mappings: 6
    networkMappings:
      - source:
          name: <source_logical_network>/<vnic_profile> 7
        target:
          name: <target_network> 8
        type: pod
    storageMappings: 9
      - source:
          name: <source_storage_domain> 10
        target:
          name: <target_storage_class> 11
          accessMode: <volume_access_mode> 12
    diskMappings:
      - source:
          id: <source_vm_disk_id> 13
        target:
          name: <target_storage_class> 14
EOF

```

- 1 If you create a ResourceMapping CR, uncomment the **resourceMapping** section.
- 2 Specify the target VM name.
- 3 Specify the source VM ID, for example, **80554327-0569-496b-bdeb-fcbbf52b827b**. You can obtain the VM ID by entering **<https://www.example.com/ovirt-engine/api/vms/>** in a web browser on the Manager machine to list all VMs. Locate the VM you want to import and its corresponding VM ID. You do not need to specify a VM name or cluster name.
- 4 If you specify the source VM name, you must also specify the source cluster. Do not specify the source VM ID.
- 5 If you specify the source cluster, you must also specify the source VM name. Do not specify the source VM ID.
- 6 If you create a ResourceMapping CR, comment out the **mappings** section.

- 7 Specify the logical network and vNIC profile of the source VM.
- 8 Specify the OpenShift Virtualization network.
- 9 If storage mappings are specified in both the **ResourceMapping** and the **VirtualMachineImport** CRs, the **VirtualMachineImport** CR takes precedence.
- 10 Specify the source storage domain.
- 11 Specify the target storage class.
- 12 Specify **ReadWriteOnce**, **ReadWriteMany**, or **ReadOnlyMany**. If no access mode is specified, {virt} determines the correct volume access mode based on the **Host** → **Migration mode** setting of the RHV VM or on the virtual disk access mode:
 - If the RHV VM migration mode is **Allow manual and automatic migration**, the default access mode is **ReadWriteMany**.
 - If the RHV virtual disk access mode is **ReadOnly**, the default access mode is **ReadOnlyMany**.
 - For all other settings, the default access mode is **ReadWriteOnce**.

Specify the source VM disk ID, for example, **8181ecc1-5db8-4193-9c92-3ddab3be7b05**. You can obtain the disk ID by entering <https://www.example.com/ovirt-engine/api/vms/vm23> in a web browser on the Manager machine and reviewing the VM details.

Specify the target storage class.

1. Follow the progress of the virtual machine import to verify that the import was successful:

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

The output indicating a successful import resembles the following example:

Example output

```
...
status:
conditions:
- lastHeartbeatTime: "2020-07-22T08:58:52Z"
  lastTransitionTime: "2020-07-22T08:58:52Z"
  message: Validation completed successfully
  reason: ValidationCompleted
  status: "True"
  type: Valid
- lastHeartbeatTime: "2020-07-22T08:58:52Z"
  lastTransitionTime: "2020-07-22T08:58:52Z"
  message: 'VM specifies IO Threads: 1, VM has NUMA tune mode specified:
interleave'
  reason: MappingRulesVerificationReportedWarnings
  status: "True"
  type: MappingRulesVerified
- lastHeartbeatTime: "2020-07-22T08:58:56Z"
  lastTransitionTime: "2020-07-22T08:58:52Z"
```

```

message: Copying virtual machine disks
reason: CopyingDisks
status: "True"
type: Processing
dataVolumes:
- name: fedora32-b870c429-11e0-4630-b3df-21da551a48c0
targetVmName: fedora32


```

7.12.4.7. Canceling a virtual machine import

You can cancel a virtual machine import in progress by using the web console.

Procedure

1. Click **Workloads** → **Virtual Machines**.

2. Click the Options menu  of the virtual machine you are importing and select **Delete Virtual Machine**.

3. In the **Delete Virtual Machine** window, click **Delete**.
The virtual machine is removed from the list of virtual machines.

7.12.4.8. Troubleshooting a virtual machine import

7.12.4.8.1. Logs

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

Procedure

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

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

- 1** Specify the namespace of your imported virtual machine.

Example output

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

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

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

- 1** Specify the VM Import Controller Pod name and the namespace.

7.12.4.8.2. Error messages

The following error message might appear:

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

```
Failed to bind volumes: provisioning failed for PVC
```

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

7.12.5. Importing a single VMware virtual machine or template

You can import a VMware vSphere 6.5, 6.7, or 7.0 VM or VM template into OpenShift Virtualization by using the VM Import wizard.

If you import a VM template, OpenShift Virtualization creates a virtual machine based on the template.

7.12.5.1. OpenShift Virtualization storage feature matrix

The following table describes local and shared persistent storage that support VM import.

Table 7.7. OpenShift Virtualization storage feature matrix

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

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

7.12.5.2. Preparing a VDDK image

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

You can download the VDDK SDK, create a VDDK image, upload the image to an image registry, and add it to the **v2v-vmware** ConfigMap.

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

**NOTE**

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

7.12.5.2.1. Configuring an internal image registry

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

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

Changing the image registry's management state

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

Procedure

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

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

Configuring registry storage for bare metal

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

Prerequisites

- Cluster administrator permissions.
- A cluster on bare metal.
- Persistent storage provisioned for your cluster, such as Red Hat OpenShift Container Storage.

**IMPORTANT**

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

- Must have 100Gi capacity.

Procedure

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

**NOTE**

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

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

```
$ oc get pod -n openshift-image-registry
```



NOTE

If the storage type is **emptyDIR**, the replica number cannot be greater than **1**.

3. Check the registry configuration:

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

Example output

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

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

4. Check the **clusteroperator** status:

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

Accessing registry directly from the cluster

You can access the registry from inside the cluster.

Procedure

Access the registry from the cluster by using internal routes:

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

```
$ oc get nodes
```

```
$ oc debug nodes/<node_address>
```

2. To enable access to tools such as **oc** and **podman** on the node, run the following command:

```
sh-4.2# chroot /host
```

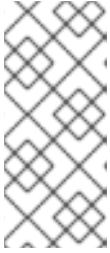
3. Log in to the container image registry by using your access token:

```
sh-4.2# oc login -u kubeadmin -p <password_from_install_log> https://api-int.
<cluster_name>.<base_domain>:6443
```

```
sh-4.2# podman login -u kubeadmin -p $(oc whoami -t) image-registry.openshift-image-
registry.svc:5000
```

You should see a message confirming login, such as:

Login Succeeded!



NOTE

You can pass any value for the user name; the token contains all necessary information. Passing a user name that contains colons will result in a login failure.

Since the Image Registry Operator creates the route, it will likely be similar to **default-route-openshift-image-registry.<cluster_name>**.

4. Perform **podman pull** and **podman push** operations against your registry:



IMPORTANT

You can pull arbitrary images, but if you have the **system:registry** role added, you can only push images to the registry in your project.

In the following examples, use:

Component	Value
<registry_ip>	172.30.124.220
<port>	5000
<project>	openshift
<image>	image
<tag>	omitted (defaults to latest)

- a. Pull an arbitrary image:

```
$ podman pull name.io/image
```

- b. Tag the new image with the form **<registry_ip>:<port>/<project>/<image>**. The project name must appear in this pull specification for OpenShift Container Platform to correctly place and later access the image in the registry:

```
$ podman tag name.io/image image-registry.openshift-image-registry.svc:5000/openshift/image
```



NOTE

You must have the **system:image-builder** role for the specified project, which allows the user to write or push an image. Otherwise, the **podman push** in the next step will fail. To test, you can create a new project to push the image.

- c. Push the newly tagged image to your registry:

```
$ podman push image-registry.openshift-image-registry.svc:5000/openshift/image
```

Exposing a secure registry manually

Instead of logging in to the OpenShift Container Platform registry from within the cluster, you can gain external access to it by exposing it with a route. This allows you to log in to the registry from outside the cluster using the route address, and to tag and push images using the route host.

Prerequisites:

- The following prerequisites are automatically performed:
 - Deploy the Registry Operator.
 - Deploy the Ingress Operator.

Procedure

You can expose the route by using **DefaultRoute** parameter in the **configs.imageregistry.operator.openshift.io** resource or by using custom routes.

To expose the registry using **DefaultRoute**:

1. Set **DefaultRoute** to **True**:

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster --patch '{"spec": {"defaultRoute":true}}' --type=merge
```

2. Log in with **podman**:

```
$ HOST=$(oc get route default-route -n openshift-image-registry --template='{{ .spec.host }}')
```

```
$ podman login -u kubeadmin -p $(oc whoami -t) --tls-verify=false $HOST 1
```

- 1** **--tls-verify=false** is needed if the cluster's default certificate for routes is untrusted. You can set a custom, trusted certificate as the default certificate with the Ingress Operator.

To expose the registry using custom routes:

1. Create a secret with your route's TLS keys:

```
$ oc create secret tls public-route-tls \
  -n openshift-image-registry \
  --cert=</path/to/tls.crt> \
  --key=</path/to/tls.key>
```

This step is optional. If you do not create a secret, the route uses the default TLS configuration from the Ingress Operator.

2. On the Registry Operator:

```
spec:
  routes:
```



```
- name: public-routes
  hostname: myregistry.mycorp.organization
  secretName: public-route-tls
```

...



NOTE

Only set **secretName** if you are providing a custom TLS configuration for the registry's route.

7.12.5.2.2. Configuring an external image registry

If you use an external image registry for the VDDK image, you can add the external image registry's certificate authorities to the OpenShift Container Platform cluster.

Optionally, you can create a pull secret from your Docker credentials and add it to your service account.

Adding certificate authorities to the cluster

You can add certificate authorities (CA) to the cluster for use when pushing and pulling images with the following procedure.

Prerequisites

- You must have cluster administrator privileges.
- You must have access to the public certificates of the registry, usually a **hostname/ca.crt** file located in the **/etc/docker/certs.d/** directory.

Procedure

1. Create a **ConfigMap** in the **openshift-config** namespace containing the trusted certificates for the registries that use self-signed certificates. For each CA file, ensure the key in the **ConfigMap** is the hostname of the registry in the **hostname[..port]** format:

```
$ oc create configmap registry-cas -n openshift-config \
  --from-file=myregistry.corp.com..5000=/etc/docker/certs.d/myregistry.corp.com:5000/ca.crt \
  --from-file=otherregistry.com=/etc/docker/certs.d/otherregistry.com/ca.crt
```

2. Update the cluster image configuration:

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
{"name":"registry-cas"}}}' --type=merge
```

Allowing pods to reference images from other secured registries

The **.dockercfg \$HOME/.docker/config.json** file for Docker clients is a Docker credentials file that stores your authentication information if you have previously logged into a secured or insecure registry.

To pull a secured container image that is not from OpenShift Container Platform's internal registry, you must create a pull secret from your Docker credentials and add it to your service account.

Procedure

- If you already have a **.dockercfg** file for the secured registry, you can create a secret from that file by running:

```
$ oc create secret generic <pull_secret_name> \
  --from-file=.dockercfg=<path/to/.dockercfg> \
  --type=kubernetes.io/dockercfg
```

- Or if you have a **\$HOME/.docker/config.json** file:

```
$ oc create secret generic <pull_secret_name> \
  --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
  --type=kubernetes.io/dockerconfigjson
```

- If you do not already have a Docker credentials file for the secured registry, you can create a secret by running:

```
$ oc create secret docker-registry <pull_secret_name> \
  --docker-server=<registry_server> \
  --docker-username=<user_name> \
  --docker-password=<password> \
  --docker-email=<email>
```

- To use a secret for pulling images for pods, you must add the secret to your service account. The name of the service account in this example should match the name of the service account the pod uses. The default service account is **default**:

```
$ oc secrets link default <pull_secret_name> --for=pull
```

7.12.5.2.3. Creating and using a VDDK image

You can download the VMware Virtual Disk Development Kit (VDDK), build a VDDK image, and push the VDDK image to your image registry. You then add the VDDK image to the **v2v-vmware** ConfigMap.

Prerequisites

- You must have access to an OpenShift Container Platform internal image registry or a secure external registry.

Procedure

1. Create and navigate to a temporary directory:

```
$ mkdir /tmp/<dir_name> && cd /tmp/<dir_name>
```

2. In a browser, navigate to [VMware code](#) and click **SDKs**.
3. Under **Compute Virtualization**, click **Virtual Disk Development Kit (VDDK)**
4. Select the VDDK version that corresponds to your VMware vSphere version, for example, VDDK 7.0 for vSphere 7.0, click **Download**, and then save the VDDK archive in the temporary directory.
5. Extract the VDDK archive:

```
$ tar -xzf VMware-vix-disklib-<version>.x86_64.tar.gz
```

6. Create a **Dockerfile**:

```
$ cat > Dockerfile <<EOF
FROM busybox:latest
COPY vmware-vix-disklib-distrib /vmware-vix-disklib-distrib
RUN mkdir -p /opt
ENTRYPOINT ["cp", "-r", "/vmware-vix-disklib-distrib", "/opt"]
EOF
```

7. Build the image:

```
$ podman build . -t <registry_route_or_server_path>/vddk:<tag> 1
```

1 Specify your image registry:

- For an internal OpenShift Container Platform registry, use the internal registry route, for example, **image-registry.openshift-image-registry.svc:5000/openshift/vddk:<tag>**.
- For an external registry, specify the server name, path, and tag, for example, **server.example.com:5000/vddk:<tag>**.

8. Push the image to the registry:

```
$ podman push <registry_route_or_server_path>/vddk:<tag>
```

9. Ensure that the image is accessible to your OpenShift Virtualization environment.

10. Edit the **v2v-vmware** ConfigMap in the **openshift-cnv** project:

```
$ oc edit configmap v2v-vmware -n openshift-cnv
```

11. Add the **vddk-init-image** parameter to the **data** stanza:

```
...
data:
  vddk-init-image: <registry_route_or_server_path>/vddk:<tag>
```

7.12.5.3. Importing a virtual machine with the VM Import wizard

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

You can also import a VM template. If you import a VM template, OpenShift Virtualization creates a virtual machine based on the template.

Prerequisites

- You must have admin user privileges.
- The VMware Virtual Disk Development Kit (VDDK) image must be in an image registry that is accessible to your OpenShift Virtualization environment.
- The VDDK image must be added to the **v2v-vmware** ConfigMap.

- The VM must be powered off.
- Virtual disks must be connected to IDE or SCSI controllers. If virtual disks are connected to a SATA controller, you can change them to IDE controllers and then migrate the VM.
- The OpenShift Virtualization local and shared persistent storage classes must support VM import.
- The OpenShift Virtualization storage must be large enough to accommodate the virtual disk.



WARNING

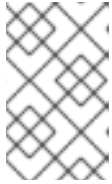
If you try to import a virtual machine with a disk that is larger than the available storage space, the operation cannot complete. You will not be able to import another virtual machine or to clean up the storage because there are insufficient resources to support object deletion. To resolve this situation, you must add more object storage devices to the storage back end.

- The OpenShift Virtualization egress network policy must allow the following traffic:


Destination	Protocol	Port
VMware ESXi hosts	TCP	443
VMware ESXi hosts	TCP	902
VMware vCenter	TCP	5840

Procedure

1. In the web console, click **Workloads → Virtual Machines**.
2. Click **Create Virtual Machine** and select **Import with Wizard**.
3. Select **VMware** from the **Provider** list.
4. Select **Connect to New Instance** or a saved vCenter instance.
 - If you select **Connect to New Instance**, enter the **vCenter hostname**, **Username**, and **Password**.
 - If you select a saved vCenter instance, the wizard connects to the vCenter instance using the saved credentials.
5. Click **Check and Save** and wait for the connection to complete.

**NOTE**

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

6. Select a virtual machine or a template.
7. Click **Next**.
8. In the **Review** screen, review your settings.
9. Click **Edit** to update the following settings:
 - **General:**
 - **Description**
 - **Operating System**
 - **Flavor**
 - **Memory**
 - **CPUs**
 - **Workload Profile**
 - **Networking:**
 - **Name**
 - **Model**
 - **Network**
 - **Type:** You must select the **masquerade** binding method.
 - **MAC Address**
 - **Storage:** Click the Options menu  of the VM disk and select **Edit** to update the following fields:
 - **Name**
 - **Source:** For example, **Import Disk**.
 - **Size**
 - **Interface**
 - **Storage Class:** Select **NFS** or **ocs-storagecluster-ceph-rbd (ceph-rbd)**. If you select **ocs-storagecluster-ceph-rbd**, you must set the **Volume Mode** of the disk to **Block**.

Other storage classes might work, but they are not officially supported.

- **Advanced → Volume Mode:** Select **Block**.
- **Advanced → Access Mode**
- **Advanced → Cloud-init:**
 - **Form:** Enter the **Hostname** and **Authenticated SSH Keys**.
 - **Custom script** Enter the **cloud-init** script in the text field.
- **Advanced → Virtual Hardware:** You can attach a virtual CD-ROM to the imported virtual machine.

10. Click **Import** or **Review and Import**, if you have edited the import settings.

A **Successfully created virtual machine** message and a list of resources created for the virtual machine are displayed. The virtual machine appears in **Workloads → Virtual Machines**.

Virtual machine wizard fields

Name	Parameter	Description
Template		Template from which to create the virtual machine. Selecting a template will automatically complete other fields.
Source	PXE	Provision virtual machine from PXE menu. Requires a PXE-capable NIC in the cluster.
	URL	Provision virtual machine from an image available from an HTTP or S3 endpoint.
	Container	Provision virtual machine from a bootable operating system container located in a registry accessible from the cluster. Example: kubevirt/cirros-registry-disk-demo .
	Disk	Provision virtual machine from a disk.
Operating System		The primary operating system that is selected for the virtual machine.
Flavor	small, medium, large, tiny, Custom	Presets that determine the amount of CPU and memory allocated to the virtual machine. The presets displayed for Flavor are determined by the operating system.

Name	Parameter	Description
Memory		Size in GiB of the memory allocated to the virtual machine.
CPUs		The amount of CPU allocated to the virtual machine.
Workload Profile	High Performance	A virtual machine configuration that is optimized for high-performance workloads.
	Server	A profile optimized to run server workloads.
	Desktop	A virtual machine configuration for use on a desktop.
Name		The name can contain lowercase letters (a-z), numbers (0-9), and hyphens (-), up to a maximum of 253 characters. The first and last characters must be alphanumeric. The name must not contain uppercase letters, spaces, periods (.), or special characters.
Description		Optional description field.
Start virtual machine on creation		Select to automatically start the virtual machine upon creation.

Cloud-init fields

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

Networking fields

Name	Description
Name	Name for the Network Interface Card.

Name	Description
Model	Indicates the model of the Network Interface Card. Supported values are e1000 , e1000e , ne2k_pci , pcnet , rtl8139 , and virtIO .
Network	List of available NetworkAttachmentDefinition objects.
Type	List of available binding methods. For the default Pod network, masquerade is the only recommended binding method. For secondary networks, use the bridge binding method. The masquerade method is not supported for non-default networks.
MAC Address	MAC address for the Network Interface Card. If a MAC address is not specified, an ephemeral address is generated for the session.


Storage fields

Name	Description
Source	Select a blank disk for the virtual machine or choose from the options available: URL , Container , Attach Cloned Disk , or Attach Disk . To select an existing disk and attach it to the virtual machine, choose Attach Cloned Disk or Attach Disk from a list of available PersistentVolumeClaims (PVCs).
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 (GiB)	Size, in GiB, of the disk.
Interface	Type of disk device. Supported interfaces are virtIO , SATA , and SCSI .
Storage Class	The StorageClass that is used to create the disk.
Advanced → Volume Mode	
Defines whether the persistent volume uses a formatted file system or raw block state. Default is Filesystem .	Advanced → Access Mode

Name	Description
	Access mode of the persistent volume. Supported access modes are Single User (RWO) , Shared Access (RWX) , and Read Only (ROX) .

Advanced storage settings

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

Name	Parameter	Description
Volume Mode	Filesystem	Stores the virtual disk on a filesystem-based volume.
	Block	Stores the virtual disk directly on the block volume. Only use Block if the underlying storage supports it.
Access Mode	Single User (RWO)	The disk can be mounted as read/write by a single node.
	Shared Access (RWX)	The disk can be mounted as read/write by many nodes.  NOTE This is required for some features, such as live migration of virtual machines between nodes.
	Read Only (ROX)	The disk can be mounted as read-only by many nodes.

7.12.5.3.1. Updating the NIC name of an imported virtual machine

You must update the NIC name of a virtual machine that you imported from VMware to conform to OpenShift Virtualization naming conventions.

Procedure

1. Log in to the virtual machine.
2. Navigate to the **/etc/sysconfig/network-scripts** directory.
3. Rename the network configuration file:

```
$ mv vmnic0 ifcfg-eth0 1
```

- 1** The first network configuration file is named **ifcfg-eth0**. Additional network configuration files are numbered sequentially, for example, **ifcfg-eth1**, **ifcfg-eth2**.

- Update the **NAME** and **DEVICE** parameters in the network configuration file:

```
NAME=eth0
DEVICE=eth0
```

- Restart the network:

```
$ systemctl restart network
```

7.12.5.4. Troubleshooting a virtual machine import

7.12.5.4.1. Logs

You can check the V2V Conversion Pod log for errors.

Procedure

- View the V2V Conversion Pod name by running the following command:

```
$ oc get pods -n <namespace> | grep v2v 1
```

- Specify the namespace of your imported virtual machine.

Example output

```
kubevirt-v2v-conversion-f66f7d-zqkz7      1/1   Running   0      4h49m
```

- View the V2V Conversion Pod log by running the following command:

```
$ oc logs <kubevirt-v2v-conversion-f66f7d-zqkz7> -f -n <namespace> 1
```

- Specify the VM Conversion Pod name and the namespace.

7.12.5.4.2. Error messages

The following error messages might appear:

- If the VMware VM is not shut down before import, the imported virtual machine displays the error message, **Readiness probe failed** in the OpenShift Container Platform console and the V2V Conversion Pod log displays the following error message:

```
INFO - have error: ('virt-v2v error: internal error: invalid argument: libvirt domain
'v2v_migration_vm_1' is running or paused. It must be shut down in order to perform virt-v2v
conversion',)
```

- The following error message is displayed in the OpenShift Container Platform console if a non-admin user tries to import a VM:

```
Could not load ConfigMap vmware-to-kubevirt-os in kube-public namespace
Restricted Access: configmaps "vmware-to-kubevirt-os" is forbidden: User cannot get
resource "configmaps" in API group "" in the namespace "kube-public"
```

Only an admin user can import a VM.

7.13. CLONING VIRTUAL MACHINES

7.13.1. Enabling user permissions to clone DataVolumes 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 ClusterRole. Bind this ClusterRole to a user to enable them to clone virtual machines to the destination namespace.

7.13.1.1. Prerequisites

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

7.13.1.2. About DataVolumes

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

7.13.1.3. Creating RBAC resources for cloning DataVolumes

Create a new ClusterRole 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 ClusterRole.

2. Create the ClusterRole 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 ClusterRole 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 RoleBinding.
- 2 The namespace for the source DataVolume.
- 3 The namespace to which the DataVolume is cloned.
- 4 The name of the ClusterRole created in the previous step.

4. Create the RoleBinding in the cluster:

```
$ oc create -f <datavolume-cloner.yaml> 1
```

- 1 The file name of the RoleBinding manifest created in the previous step.

7.13.2. Cloning a virtual machine disk into a new DataVolume

You can clone the PersistentVolumeClaim (PVC) of a virtual machine disk into a new DataVolume by referencing the source PVC in your DataVolume configuration file.



WARNING

Cloning operations between different volume modes are not supported. The **volumeMode** values must match in both the source and target specifications.

For example, if you attempt to clone from a PersistentVolume (PV) with **volumeMode: Block** to a PV with **volumeMode: Filesystem**, the operation fails with an error message.

7.13.2.1. Prerequisites

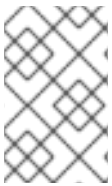
- Users need [additional permissions](#) to clone the PVC of a virtual machine disk into another namespace.

7.13.2.2. About DataVolumes

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

7.13.2.3. Cloning the PersistentVolumeClaim of a virtual machine disk into a new DataVolume

You can clone a PersistentVolumeClaim (PVC) of an existing virtual machine disk into a new DataVolume. The new DataVolume can then be used for a new virtual machine.



NOTE

When a DataVolume is created independently of a virtual machine, the lifecycle of the DataVolume is independent of the virtual machine. If the virtual machine is deleted, neither the DataVolume 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 DataVolume object that specifies the name of the new DataVolume, the name and namespace of the source PVC, and the size of the new DataVolume.

For example:

```
apiVersion: cdi.kubevirt.io/v1alpha1
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 DataVolume.
- 2 The namespace where the source PVC exists.
- 3 The name of the source PVC.
- 4 The size of the new DataVolume. 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 DataVolume:

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



NOTE

DataVolumes prevent a virtual machine from starting before the PVC is prepared, so you can create a virtual machine that references the new DataVolume while the PVC clones.

7.13.2.4. Template: DataVolume clone configuration file

example-clone-dv.yaml

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: "example-clone-dv"
spec:
  source:
    pvc:
      name: source-pvc
      namespace: example-ns
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: "1G"
```

7.13.2.5. CDI supported operations matrix

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

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

Content types	HTTP	HTTPS	HTTP basic auth	Registry	Upload
KubeVirt (RAW)	<input checked="" type="checkbox"/> RAW <input checked="" type="checkbox"/> GZ <input checked="" type="checkbox"/> XZ	<input checked="" type="checkbox"/> RAW <input checked="" type="checkbox"/> GZ <input checked="" type="checkbox"/> XZ	<input checked="" type="checkbox"/> RAW <input checked="" type="checkbox"/> GZ <input checked="" type="checkbox"/> XZ	<input checked="" type="checkbox"/> RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	<input checked="" type="checkbox"/> RAW* <input checked="" type="checkbox"/> GZ* <input checked="" type="checkbox"/> XZ*

Supported operation

Unsupported operation

* Requires scratch space

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

7.13.3. Cloning a virtual machine by using a DataVolumeTemplate

You can create a new virtual machine by cloning the PersistentVolumeClaim (PVC) of an existing VM. By including a **dataVolumeTemplate** in your virtual machine configuration file, you create a new DataVolume from the original PVC.



WARNING

Cloning operations between different volume modes are not supported. The **volumeMode** values must match in both the source and target specifications.

For example, if you attempt to clone from a PersistentVolume (PV) with **volumeMode: Block** to a PV with **volumeMode: Filesystem**, the operation fails with an error message.

7.13.3.1. Prerequisites

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

7.13.3.2. About DataVolumes

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

7.13.3.3. Creating a new virtual machine from a cloned PersistentVolumeClaim by using a DataVolumeTemplate

You can create a virtual machine that clones the PersistentVolumeClaim (PVC) of an existing virtual

machine into a DataVolume. By referencing a **dataVolumeTemplate** in the virtual machine **spec**, the **source** PVC is cloned to a DataVolume, which is then automatically used for the creation of the virtual machine.



NOTE

When a DataVolume is created as part of the DataVolumeTemplate of a virtual machine, the lifecycle of the DataVolume is then dependent on the virtual machine. If the virtual machine is deleted, the DataVolume 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** DataVolume called **favorite-clone** is created from **my-favorite-vm-disk**.

For example:

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm-dv-clone
  name: vm-dv-clone 1
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm-dv-clone
    spec:
      domain:
        devices:
          disks:
            - disk:
                bus: virtio
                name: root-disk
      resources:
        requests:
          memory: 64M
      volumes:
        - dataVolume:
            name: favorite-clone
            name: root-disk
  dataVolumeTemplates:
    - metadata:
```



```

name: favorite-clone
spec:
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 2Gi
  source:
    pvc:
      namespace: "source-namespace"
      name: "my-favorite-vm-disk"

```

1 The virtual machine to create.

3. Create the virtual machine with the PVC-cloned DataVolume:

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

7.13.3.4. Template: DataVolume virtual machine configuration file

example-dv-vm.yaml

```

apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: example-vm
  name: example-vm
spec:
  dataVolumeTemplates:
  - metadata:
      name: example-dv
    spec:
      pvc:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 1G
      source:
        http:
          url: "" 1
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: example-vm
    spec:
      domain:
        cpu:
          cores: 1
      devices:
        disks:

```

```

- disk:
  bus: virtio
  name: example-dv-disk
machine:
  type: q35
resources:
  requests:
    memory: 1G
terminationGracePeriodSeconds: 0
volumes:
- dataVolume:
  name: example-dv
  name: example-dv-disk

```

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

7.13.3.5. CDI supported operations matrix

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

Content types	HTTP	HTTPS	HTTP basic auth	Registry	Upload
KubeVirt(QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
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

7.13.4. Cloning a virtual machine disk into a new block storage DataVolume

You can clone the PersistentVolumeClaim (PVC) of a virtual machine disk into a new block DataVolume by referencing the source PVC in your DataVolume configuration file.



WARNING

Cloning operations between different volume modes are not supported. The **volumeMode** values must match in both the source and target specifications.

For example, if you attempt to clone from a PersistentVolume (PV) with **volumeMode: Block** to a PV with **volumeMode: Filesystem**, the operation fails with an error message.

7.13.4.1. Prerequisites

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

7.13.4.2. About DataVolumes

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

7.13.4.3. About block PersistentVolumes

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

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

7.13.4.4. Creating a local block PersistentVolume

Create a local block PersistentVolume (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 configuration 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** configuration 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 StorageClass 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 filename of the PersistentVolume created in the previous step.

7.13.4.5. Cloning the PersistentVolumeClaim of a virtual machine disk into a new DataVolume

You can clone a PersistentVolumeClaim (PVC) of an existing virtual machine disk into a new DataVolume. The new DataVolume can then be used for a new virtual machine.



NOTE

When a DataVolume is created independently of a virtual machine, the lifecycle of the DataVolume is independent of the virtual machine. If the virtual machine is deleted, neither the DataVolume 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 PersistentVolume (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 DataVolume object that specifies the name of the new DataVolume, the name and namespace of the source PVC, **volumeMode: Block** so that an available block PV is used, and the size of the new DataVolume.

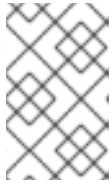
For example:

```
apiVersion: cdi.kubevirt.io/v1alpha1
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
    volumeMode: Block 5
```

- 1 The name of the new DataVolume.
- 2 The namespace where the source PVC exists.
- 3 The name of the source PVC.
- 4 The size of the new DataVolume. You must allocate enough space, or the cloning operation fails. The size must be the same as or larger than the source PVC.
- 5 Specifies that the destination is a block PV

- Start cloning the PVC by creating the DataVolume:

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



NOTE

DataVolumes prevent a virtual machine from starting before the PVC is prepared, so you can create a virtual machine that references the new DataVolume while the PVC clones.

7.13.4.6. CDI supported operations matrix

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

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

✓ Supported operation

Unsupported operation

* Requires scratch space

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

7.14. VIRTUAL MACHINE NETWORKING

7.14.1. Using the default Pod network for virtual machines

You can use the default Pod network with OpenShift Virtualization. To do so, you must use the **masquerade** binding method. It is the only recommended binding method for use with the default Pod network. Do not use **masquerade** mode with non-default networks.

For [secondary networks](#), use the **bridge** binding method.



NOTE

The *KubeMacPool* component provides a MAC address pool service for virtual machine NICs in designated namespaces. It is not enabled by default. [Enable a MAC address pool in a namespace](#) by applying the KubeMacPool label to that namespace.

7.14.1.1. Configuring masquerade mode from the command line

You can use masquerade mode to hide a virtual machine's outgoing traffic behind the Pod IP address. Masquerade mode uses Network Address Translation (NAT) to connect virtual machines to the Pod network backend through a Linux bridge.

Enable masquerade mode and allow traffic to enter the virtual machine by editing your virtual machine configuration file.

Prerequisites

- The virtual machine must be configured to use DHCP to acquire IPv4 addresses. The examples below are configured to use DHCP.

Procedure

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

```
kind: VirtualMachine
spec:
  domain:
    devices:
      interfaces:
        - name: red
          masquerade: {} 1
        ports:
          - port: 80 2
      networks:
        - name: red
          pod: {}
```

- 1 Connect using masquerade mode
- 2 Allow incoming traffic on port 80

2. Create the virtual machine:

```
$ oc create -f <vm-name>.yaml
```

7.14.1.2. Selecting binding method

If you create a virtual machine from the OpenShift Virtualization [web console wizard](#), select the required binding method from the **Networking** screen.

7.14.1.2.1. Networking fields

Name	Description
Name	Name for the Network Interface Card.
Model	Indicates the model of the Network Interface Card. Supported values are e1000 , e1000e , ne2k_pci , pcnet , rtl8139 , and virtIO .

Name	Description
Network	List of available NetworkAttachmentDefinition objects.
Type	List of available binding methods. For the default Pod network, masquerade is the only recommended binding method. For secondary networks, use the bridge binding method. The masquerade method is not supported for non-default networks.
MAC Address	MAC address for the Network Interface Card. If a MAC address is not specified, an ephemeral address is generated for the session.

7.14.1.3. Virtual machine configuration examples for the default network

7.14.1.3.1. Template: virtual machine configuration file

```

apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: default
spec:
  running: false
  template:
    spec:
      domain:
        devices:
          disks:
            - name: containerdisk
              disk:
                bus: virtio
            - name: cloudinitdisk
              disk:
                bus: virtio
          interfaces:
            - masquerade: {}
              name: default
      resources:
        requests:
          memory: 1024M
      networks:
        - name: default
          pod: {}
      volumes:
        - name: containerdisk
          containerDisk:
            image: kubevirt/fedora-cloud-container-disk-demo

```



```

- name: cloudinitdisk
  cloudInitNoCloud:
    userData: |
      #!/bin/bash
      echo "fedora" | passwd fedora --stdin

```

7.14.1.3.2. Template: Windows virtual machine instance configuration file

```

apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstance
metadata:
  labels:
    special: vmi-windows
  name: vmi-windows
spec:
  domain:
    clock:
      timer:
        hpet:
          present: false
        hyperv: {}
        pit:
          tickPolicy: delay
        rtc:
          tickPolicy: catchup
        utc: {}
    cpu:
      cores: 2
    devices:
      disks:
        - disk:
            bus: sata
            name: pvcdisk
      interfaces:
        - masquerade: {}
          model: e1000
          name: default
    features:
      acpi: {}
      apic: {}
      hyperv:
        relaxed: {}
        spinlocks:
          spinlocks: 8191
        vapic: {}
    firmware:
      uuid: 5d307ca9-b3ef-428c-8861-06e72d69f223
    machine:
      type: q35
    resources:
      requests:
        memory: 2Gi
  networks:
    - name: default
      pod: {}

```

```

terminationGracePeriodSeconds: 0
volumes:
- name: pvcdisk
  persistentVolumeClaim:
    claimName: disk-windows

```

7.14.1.4. Creating a service from a virtual machine

Create a service from a running virtual machine by first creating a **Service** object to expose the virtual machine.

The **ClusterIP** service type exposes the virtual machine internally, within the cluster. The **NodePort** or **LoadBalancer** service types expose the virtual machine externally, outside of the cluster.

This procedure presents an example of how to create, connect to, and expose a **Service** object of **type: ClusterIP** as a virtual machine-backed service.



NOTE

ClusterIP is the default service **type**, if the service **type** is not specified.

Procedure

1. Edit the virtual machine YAML as follows:

```

apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  name: vm-ephemeral
  namespace: example-namespace
spec:
  running: false
  template:
    metadata:
      labels:
        special: key 1
    spec:
      domain:
        devices:
          disks:
            - name: containerdisk
              disk:
                bus: virtio
            - name: cloudinitdisk
              disk:
                bus: virtio
          interfaces:
            - masquerade: {}
              name: default
      resources:
        requests:
          memory: 1024M
      networks:
        - name: default
          pod: {}

```

```
volumes:
- name: containerdisk
  containerDisk:
    image: kubevirt/fedora-cloud-container-disk-demo
- name: cloudinitdisk
  cloudInitNoCloud:
    userData: |
      #!/bin/bash
      echo "fedora" | passwd fedora --stdin
```

- 1 Add the label **special: key** in the **spec.template.metadata.labels** section.



NOTE

Labels on a virtual machine are passed through to the pod. The labels on the **VirtualMachine**, for example **special: key**, must match the labels in the **Service** YAML **selector** attribute, which you create later in this procedure.

2. Save the virtual machine YAML to apply your changes.
3. Edit the **Service** YAML to configure the settings necessary to create and expose the **Service** object:

```
apiVersion: v1
kind: Service
metadata:
  name: vmervice 1
  namespace: example-namespace 2
spec:
  ports:
  - port: 27017
    protocol: TCP
    targetPort: 22 3
  selector:
    special: key 4
  type: ClusterIP 5
```

- 1 Specify the **name** of the service you are creating and exposing.
- 2 Specify **namespace** in the **metadata** section of the **Service** YAML that corresponds to the **namespace** you specify in the virtual machine YAML.
- 3 Add **targetPort: 22**, exposing the service on SSH port **22**.
- 4 In the **spec** section of the **Service** YAML, add **special: key** to the **selector** attribute, which corresponds to the **labels** you added in the virtual machine YAML configuration file.
- 5 In the **spec** section of the **Service** YAML, add **type: ClusterIP** for a ClusterIP service. To create and expose other types of services externally, outside of the cluster, such as **NodePort** and **LoadBalancer**, replace **type: ClusterIP** with **type: NodePort** or **type: LoadBalancer**, as appropriate.

4. Save the **Service** YAML to store the service configuration.

5. Create the **ClusterIP** service:

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

6. Start the virtual machine. If the virtual machine is already running, restart it.
7. Query the **Service** object to verify it is available and is configured with type **ClusterIP**.

Verification

- Run the **oc get service** command, specifying the **namespace** that you reference in the virtual machine and **Service** YAML files.

```
$ oc get service -n example-namespace
```

Example output

```
NAME      TYPE      CLUSTER-IP    EXTERNAL-IP  PORT(S)    AGE
vmervice  ClusterIP  172.30.3.149  <none>       27017/TCP  2m
```

- As shown from the output, **vmervice** is running.
 - The **TYPE** displays as **ClusterIP**, as you specified in the **Service** YAML.
8. Establish a connection to the virtual machine that you want to use to back your service. Connect from an object inside the cluster, such as another virtual machine.
 - a. Edit the virtual machine YAML as follows:

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  name: vm-connect
  namespace: example-namespace
spec:
  running: false
  template:
    spec:
      domain:
        devices:
          disks:
            - name: containerdisk
              disk:
                bus: virtio
            - name: cloudinitdisk
              disk:
                bus: virtio
          interfaces:
            - masquerade: {}
              name: default
      resources:
        requests:
          memory: 1024M
      networks:
        - name: default
```

```

pod: {}
volumes:
- name: containerdisk
  containerDisk:
    image: kubevirt/fedora-cloud-container-disk-demo
- name: cloudinitdisk
  cloudInitNoCloud:
    userData: |
      #!/bin/bash
      echo "fedora" | passwd fedora --stdin

```

- b. Run the **oc create** command to create a second virtual machine, where **file.yaml** is the name of the virtual machine YAML:

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

- c. Start the virtual machine.
- d. Connect to the virtual machine by running the following **virtctl** command:

```
$ virtctl -n example-namespace console <new-vm-name>
```



NOTE

For service type **LoadBalancer**, use the **vinagre** client to connect your virtual machine by using the public IP and port. External ports are dynamically allocated when using service type **LoadBalancer**.

- e. Run the **ssh** command to authenticate the connection, where **172.30.3.149** is the ClusterIP of the service and **fedora** is the user name of the virtual machine:

```
$ ssh fedora@172.30.3.149 -p 27017
```

Verification

- You receive the command prompt of the virtual machine backing the service you want to expose. You now have a service backed by a running virtual machine.

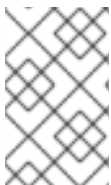
Additional resources

- [Configuring external IPs](#)

7.14.2. Attaching a virtual machine to multiple networks

OpenShift Virtualization provides layer-2 networking capabilities that allow you to connect virtual machines to multiple networks. You can import virtual machines with existing workloads that depend on access to multiple interfaces. You can also configure a PXE network so that you can boot machines over the network.

To get started, a network administrator configures a bridge NetworkAttachmentDefinition for a namespace in the web console or CLI. Users can then create a NIC to attach Pods and virtual machines in that namespace to the bridge network.

**NOTE**

The *KubeMacPool* component provides a MAC address pool service for virtual machine NICs in designated namespaces. It is not enabled by default. [Enable a MAC address pool in a namespace](#) by applying the KubeMacPool label to that namespace.

7.14.2.1. 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.

NetworkAttachmentDefinition

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

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.

7.14.2.2. Creating a NetworkAttachmentDefinition

7.14.2.3. Prerequisites

- A Linux bridge must be configured and attached on every node. See the [node networking](#) section for more information.

**WARNING**

Configuring ipam in a NetworkAttachmentDefinition for virtual machines is not supported.

7.14.2.3.1. Creating a Linux bridge NetworkAttachmentDefinition in the web console

The NetworkAttachmentDefinition is a custom resource that exposes layer-2 devices to a specific namespace in your OpenShift Virtualization cluster.

Network administrators can create NetworkAttachmentDefinitions to provide existing layer-2 networking to pods and virtual machines.

Procedure

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

7.14.2.3.2. Creating a Linux bridge NetworkAttachmentDefinition in the CLI

As a network administrator, you can configure a NetworkAttachmentDefinition of type **cnv-bridge** to provide Layer-2 networking to pods and virtual machines.



NOTE

The NetworkAttachmentDefinition must be in the same namespace as the Pod or virtual machine.

Procedure

1. Create a new file for the NetworkAttachmentDefinition in any local directory. The file must have the following contents, modified to match your configuration:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: a-bridge-network
  annotations:
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/br0 1
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "a-bridge-network", 2
    "plugins": [
      {
        "type": "cnv-bridge", 3
        "bridge": "br0", 4
        "vlan": 1 5
      },
      {
        "type": "cnv-tuning" 6
      }
    ]
  }
```

```
}
]
}'
```

- 1 If you add this annotation to your NetworkAttachmentDefinition, your virtual machine instances will only run on nodes that have the **br0** bridge connected.
- 2 Required. A name for the configuration. It is recommended to match the configuration name to the **name** value of the NetworkAttachmentDefinition.
- 3 The actual name of the Container Network Interface (CNI) plug-in that provides the network for this NetworkAttachmentDefinition. Do not change this field unless you want to use a different CNI.
- 4 You must substitute the actual name of the bridge, if it is not **br0**.
- 5 Optional: The VLAN tag.
- 6 Required. This allows the MAC pool manager to assign a unique MAC address to the connection.

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

2. Edit the configuration file of a virtual machine or virtual machine instance that you want to connect to the bridge network, for example:

```
apiVersion: v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  domain:
    devices:
      interfaces:
        - masquerade: {}
          name: default
        - bridge: {}
          name: bridge-net 1
  ...
  networks:
    - name: default
      pod: {}
    - name: bridge-net 2
      multus:
        networkName: a-bridge-network 3
  ...
```

- 1 2 The **name** value for the bridge interface and network must be the same.
- 3 You must substitute the actual **name** value from the NetworkAttachmentDefinition.

**NOTE**

The virtual machine instance will be connected to both the **default** Pod network and **bridge-net**, which is defined by a NetworkAttachmentDefinition named **a-bridge-network**.

3. Apply the configuration file to the resource:

```
$ oc create -f <local/path/to/network-attachment-definition.yaml>
```

**NOTE**

When defining the NIC in the next section, ensure that the **NETWORK** value is the bridge network name from the NetworkAttachmentDefinition you created in the previous section.

7.14.2.4. Creating a NIC for a virtual machine

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

Procedure

1. In the correct project in the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click **Network Interfaces** to display the NICs already attached to the virtual machine.
5. Click **Add Network Interface** to create a new slot in the list.
6. Fill in the **Name**, **Model**, **Network**, **Type**, and **MAC Address** for the new NIC.
7. Click the **Add** button to save and attach the NIC to the virtual machine.

7.14.2.5. Networking fields

Name	Description
Name	Name for the Network Interface Card.
Model	Indicates the model of the Network Interface Card. Supported values are e1000 , e1000e , ne2k_pci , pcnet , rtl8139 , and virtIO .
Network	List of available NetworkAttachmentDefinition objects.

Name	Description
Type	List of available binding methods. For the default Pod network, masquerade is the only recommended binding method. For secondary networks, use the bridge binding method. The masquerade method is not supported for non-default networks.
MAC Address	MAC address for the Network Interface Card. If a MAC address is not specified, an ephemeral address is generated for the session.

Install the optional [QEMU guest agent](#) on the virtual machine so that the host can display relevant information about the additional networks.

7.14.3. 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.

7.14.3.1. Prerequisites

- You must have [installed the SR-IOV Operator](#).
- You must have [configured the SR-IOV Operator](#).

7.14.3.2. 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.

7.14.3.2.1. 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

```

❶ The value of the **name** field is the same as the name of the worker node.

❷ The **interfaces** stanza includes a list of all of the SR-IOV devices discovered by the Operator on the worker node.

7.14.3.3. Configuring SR-IOV network devices

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



NOTE

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

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

Prerequisites

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

Procedure

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

```

apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" 4
  priority: <priority> 5
  mtu: <mtu> 6
  numVfs: <num> 7
  nicSelector: 8
    vendor: "<vendor_code>" 9
    deviceID: "<device_id>" 10
    pfNames: ["<pf_name>", ...] 11
    rootDevices: ["<pci_bus_id>", "..."] 12
  deviceType: vfio-pci 13
  isRdma: false 14

```

- 1 Specify a name for the CR object.

- 2 Specify the namespace where the SR-IOV Operator is installed.
- 3 Specify the resource name of the SR-IOV device plug-in. You can create multiple **SriovNetworkNodePolicy** objects for a resource name.
- 4 Specify the node selector to select which nodes are configured. Only SR-IOV network devices on selected nodes are configured. The SR-IOV Container Network Interface (CNI) plug-in and device plug-in are deployed only on selected nodes.
- 5 Optional: Specify an integer value between **0** and **99**. A smaller number gets higher priority, so a priority of **10** is higher than a priority of **99**. The default value is **99**.
- 6 Optional: Specify a value for the maximum transmission unit (MTU) of the virtual function. The maximum MTU value can vary for different NIC models.
- 7 Specify the number of the virtual functions (VF) to create for the SR-IOV physical network device. For an Intel Network Interface Card (NIC), the number of VFs cannot be larger than the total VFs supported by the device. For a Mellanox NIC, the number of VFs cannot be larger than **128**.
- 8 The **nicSelector** mapping selects the Ethernet device for the Operator to configure. You do not need to specify values for all the parameters. It is recommended to identify the Ethernet adapter with enough precision to minimize the possibility of selecting an Ethernet device unintentionally. If you specify **rootDevices**, you must also specify a value for **vendor**, **deviceID**, or **pfNames**. If you specify both **pfNames** and **rootDevices** at the same time, ensure that they point to an identical device.
- 9 Optional: Specify the vendor hex code of the SR-IOV network device. The only allowed values are either **8086** or **15b3**.
- 10 Optional: Specify the device hex code of SR-IOV network device. The only allowed values are **158b**, **1015**, **1017**.
- 11 Optional: The parameter accepts an array of one or more physical function (PF) names for the Ethernet device.
- 12 The parameter accepts an array of one or more PCI bus addresses for the physical function of the Ethernet device. Provide the address in the following format: **0000:02:00.1**.
- 13 The **vfio-pci** driver type is required for virtual functions in OpenShift Virtualization.
- 14 Optional: Specify whether to enable remote direct memory access (RDMA) mode. For a Mellanox card, set **isRdma** to **false**. The default value is **false**.



NOTE

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

2. 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.

- 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}'
```

7.14.3.4. Next steps

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

7.14.4. 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.

7.14.4.1. Prerequisites

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

7.14.4.2. 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

- 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

```

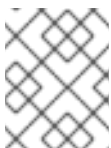
- 1 Replace **<name>** with a name for the object. The SR-IOV Network Operator creates a **NetworkAttachmentDefinition** object with same name.
- 2 Specify the namespace where the SR-IOV Network Operator is installed.
- 3 Replace **<sriov_resource_name>** with the value for the **.spec.resourceName** 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.

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

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

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

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

7.14.4.3. Next steps

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

7.14.5. 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.

7.14.5.1. Prerequisites

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

7.14.5.2. 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

- 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.

7.14.6. Installing the QEMU guest agent on virtual machines

The QEMU guest agent is a daemon that runs on the virtual machine. The agent passes network information on the virtual machine, notably the IP address of additional networks, to the host.

7.14.6.1. Installing QEMU guest agent on a Linux virtual machine

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

Procedure

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

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

3. Start the QEMU guest agent service:

```
$ systemctl start qemu-guest-agent
```

4. Ensure the service is persistent:

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

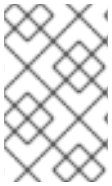
You can also install and start the QEMU guest agent by using the **custom script** field in the **cloud-init** section of the wizard when creating either virtual machines or virtual machines templates in the web console.

7.14.6.2. Installing QEMU guest agent on a Windows virtual machine

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

7.14.6.2.1. 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

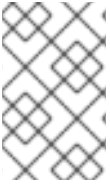
This procedure uses a generic approach to adding drivers to Windows. The process might differ slightly between versions of Windows. Refer to the installation documentation for your version of Windows for specific installation steps.

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.

7.14.6.2.2. 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. Refer to 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.

7.14.7. Viewing the IP address of NICs on a virtual machine

The QEMU guest agent runs on the virtual machine and passes the IP address of attached NICs to the host, allowing you to view the IP address from both the web console and the **oc** client.

7.14.7.1. Prerequisites

1. Verify that the guest agent is installed and running by entering the following command:

```
$ systemctl status qemu-guest-agent
```

2. If the guest agent is not installed and running, [install and run the guest agent on the virtual machine](#).

7.14.7.2. 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:fef4:25/64
Mac: 0a:58:0a:f4:00:25
Name: default
Interface Name: v2
Ip Address: 1.1.1.7/24
Ip Addresses:
  1.1.1.7/24
  fe80::f4d9:70ff:fe13:9089/64
Mac: f6:d9:70:13:90:89
Interface Name: v1
Ip Address: 1.1.1.1/24
Ip Addresses:
  1.1.1.1/24
  1.1.1.2/24
  1.1.1.4/24
  2001:de7:0:f101::1/64
  2001:db8:0:f101::1/64
  fe80::1420:84ff:fe10:17aa/64
Mac: 16:20:84:10:17:aa

```

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

The IP information displays in the **Virtual Machine Overview** screen for the virtual machine.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine name to open the **Virtual Machine Overview** screen.

The information for each attached NIC is displayed under **IP Address**.

7.14.8. Using a MAC address pool for virtual machines

The *KubeMacPool* component provides a MAC address pool service for virtual machine NICs in designated namespaces. Enable a MAC address pool in a namespace by applying the KubeMacPool label to that namespace.

7.14.8.1. About KubeMacPool

If you enable the KubeMacPool component for a namespace, virtual machine NICs in that namespace are allocated MAC addresses from a MAC address 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 disabled by default. Enable a MAC address pool for a namespace by applying the KubeMacPool label to the namespace.

7.14.8.2. Enabling a MAC address pool for a namespace in the CLI

Enable a MAC address pool for virtual machines in a namespace by applying the **mutatevirtualmachines.kubemacpool.io=allocate** label to the namespace.

Procedure

- Add the KubeMacPool label to the namespace. The following example adds the KubeMacPool label to two namespaces, **<namespace1>** and **<namespace2>**:

```
$ oc label namespace <namespace1> <namespace2>
mutatevirtualmachines.kubemacpool.io=allocate
```

7.14.8.3. Disabling a MAC address pool for a namespace in the CLI

Disable a MAC address pool for virtual machines in a namespace by removing the **mutatevirtualmachines.kubemacpool.io** label.

Procedure

- Remove the KubeMacPool label from the namespace. The following example removes the KubeMacPool label from two namespaces, **<namespace1>** and **<namespace2>**:

```
$ oc label namespace <namespace1> <namespace2>
mutatevirtualmachines.kubemacpool.io-
```

7.15. VIRTUAL MACHINE DISKS

7.15.1. Storage features

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

7.15.1.1. OpenShift Virtualization storage feature matrix

Table 7.8. OpenShift Virtualization storage feature matrix

	Virtual machine live migration	Host-assisted virtual machine disk cloning	Storage-assisted virtual machine disk cloning	Virtual machine snapshots
OpenShift Container Storage: RBD block-mode volumes	Yes	Yes	Yes	Yes
OpenShift Virtualization hostpath provisioner	No	Yes	No	No
Other multi-node writable storage	Yes ^[1]	Yes	Yes ^[2]	Yes ^[2]
Other single-node writable storage	No	Yes	Yes ^[2]	Yes ^[2]

1. PVCs must request a ReadWriteMany access mode.
2. 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 SRI-OV and GPU

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

7.15.2. Configuring local storage for virtual machines

You can configure local storage for your virtual machines by using the hostpath provisioner feature.

7.15.2.1. About the hostpath provisioner

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

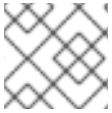
When you install the OpenShift Virtualization Operator, the hostpath provisioner Operator is automatically installed. To use it, you must:

- Configure SELinux:
 - If you use Red Hat Enterprise Linux CoreOS 8 workers, you must create a MachineConfig object on each node.
 - Otherwise, apply the SELinux label **container_file_t** to the PersistentVolume (PV) backing directory on each node.
- Create a HostPathProvisioner custom resource.
- Create a **StorageClass** object for the hostpath provisioner.

The hostpath provisioner Operator deploys the provisioner as a *DaemonSet* on each node when you create its custom resource. In the custom resource file, you specify the backing directory for the PersistentVolumes that the hostpath provisioner creates.

7.15.2.2. Configuring SELinux for the hostpath provisioner on Red Hat Enterprise Linux CoreOS 8

You must configure SELinux before you create the HostPathProvisioner custom resource. To configure SELinux on Red Hat Enterprise Linux CoreOS 8 workers, you must create a **MachineConfig** object on each node.



NOTE

If you do not use Red Hat Enterprise Linux CoreOS workers, skip this procedure.

Prerequisites

- Create a backing directory on each node for the PersistentVolumes (PVs) that the hostpath provisioner creates.

Procedure

1. Create the MachineConfig file. For example:

```
$ touch machineconfig.yaml
```

2. Edit the file, ensuring that you include the directory where you want the hostpath provisioner to create PVs. For example:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: 50-set-selinux-for-hostpath-provisioner
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  config:
    ignition:
      version: 2.2.0
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Set SELinux chcon for hostpath provisioner
            Before=kubelet.service

            [Service]
            ExecStart=/usr/bin/chcon -Rt container_file_t <path/to/backing/directory> 1

            [Install]
            WantedBy=multi-user.target
          enabled: true
          name: hostpath-provisioner.service
```

- 1 Specify the backing directory where you want the provisioner to create PVs.

3. Create the **MachineConfig** object:

```
$ oc create -f machineconfig.yaml -n <namespace>
```

7.15.2.3. Using the `hostpath` provisioner to enable local storage

To deploy the `hostpath` provisioner and enable your virtual machines to use local storage, first create a `HostPathProvisioner` custom resource.

Prerequisites

- Create a backing directory on each node for the PersistentVolumes (PVs) that the `hostpath` provisioner creates.
- Apply the SELinux context **`container_file_t`** to the PV backing directory on each node. For example:

```
$ sudo chcon -t container_file_t -R </path/to/backing/directory>
```



NOTE

If you use Red Hat Enterprise Linux CoreOS 8 workers, you must configure SELinux by using a `MachineConfig` manifest instead.

Procedure

1. Create the `HostPathProvisioner` custom resource file. For example:

```
$ touch hostpathprovisioner_cr.yaml
```

2. Edit the file, ensuring that the **`spec.pathConfig.path`** value is the directory where you want the `hostpath` provisioner to create PVs. For example:

```
apiVersion: hostpathprovisioner.kubevirt.io/v1alpha1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
  pathConfig:
    path: "</path/to/backing/directory>" 1
    useNamingPrefix: "false" 2
```

- 1 Specify the backing directory where you want the provisioner to create PVs.
- 2 Change this value to **`true`** if you want to use the name of the PersistentVolumeClaim (PVC) that is bound to the created PV as the prefix of the directory name.

**NOTE**

If you did not create the backing directory, the provisioner attempts to create it for you. If you did not apply the **container_file_t** SELinux context, this can cause **Permission denied** errors.

3. Create the custom resource in the **openshift-cnv** namespace:

```
$ oc create -f hostpathprovisioner_cr.yaml -n openshift-cnv
```

7.15.2.4. Creating a StorageClass object

When you create a **StorageClass** object, you set parameters that affect the dynamic provisioning of PersistentVolumes (PVs) that belong to that storage class.

**NOTE**

You cannot update a **StorageClass** object's parameters after you create it.

Procedure

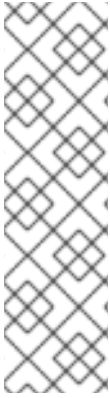
1. Create a YAML file for defining the storage class. For example:

```
$ touch storageclass.yaml
```

2. Edit the file. For example:

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

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



NOTE

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

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

1. Create the **StorageClass** object:

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

Additional information

- [Storage Classes](#)

7.15.3. 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.

7.15.3.1. 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 **CDIConfig** object defines the user configuration for the Containerized Data Importer (CDI). The CPU and memory request and limit values for the **CDIConfig** object are set to a default value of 0. This ensures that pods created by CDI that make no compute resource requirements are given the default values and are allowed to run in a namespace that is restricted with a quota.

7.15.3.2. Editing the **CDIConfig** object to override CPU and memory defaults

Modify the default settings for CPU and memory requests and limits for your use case by editing the **spec** attribute of the **CDIConfig** object.

Prerequisites

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

Procedure

1. Edit the **cdiconfig/config** by running the following command:

```
$ oc edit cdiconfig/config
```

2. Change the default CPU and memory requests and limits by editing the **spec: podResourceRequirements** property of the **CDIConfig** object:

```

apiVersion: cdi.kubevirt.io/v1alpha1
kind: CDIConfig
metadata:
  labels:
    app: containerized-data-importer
    cdi.kubevirt.io: ""
  name: config
spec:
  podResourceRequirements:
    limits:
      cpu: "4"
      memory: "1Gi"
    requests:
      cpu: "1"
      memory: "250Mi"
  ...

```

3. Save and exit the editor to update the **CDIConfig** object.

Verification

- View the **CDIConfig** status and verify your changes by running the following command:

```
$ oc get cdiconfig config -o yaml
```

7.15.3.3. Additional resources

- [Resource quotas per project](#)

7.15.4. Uploading local disk images by using the virtctl tool

You can upload a locally stored disk image to a new or existing DataVolume by using the **virtctl** command-line utility.

7.15.4.1. Prerequisites

- [Install](#) the **kubevirt-virtctl** package.
- If you require scratch space according to the [CDI supported operations matrix](#), you must first [define a StorageClass or prepare CDI scratch space](#) for this operation to complete successfully.

7.15.4.2. About DataVolumes

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

7.15.4.3. Creating an upload DataVolume

You can manually create a DataVolume with an **upload** data source to use for uploading local disk images.

Procedure

1. Create a DataVolume configuration that specifies **spec: source: upload{}**:

```

apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: <upload-datavolume> 1
spec:
  source:
    upload: {}
  pvc:
    accessModes:
      - ReadWriteOnce
  resources:
    requests:
      storage: <2Gi> 2

```

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

2. Create the DataVolume by running the following command:

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

7.15.4.4. Uploading a local disk image to a DataVolume

You can use the **virtctl** CLI utility to upload a local disk image from a client machine to a DataVolume (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

- A virtual machine disk image, in RAW, ISO, or QCOW2 format, optionally compressed by using **xz** or **gz**.
- 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 DataVolume that you want to use. If this DataVolume does not exist, it is created automatically.

- The size of the DataVolume, 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 DataVolume.
- 2** The size of the DataVolume. 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 DataVolume, 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 DataVolume was created, view all DataVolume objects by running the following command:

```
$ oc get dvs
```

7.15.4.5. CDI supported operations matrix

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

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

✓ Supported operation

□ Unsupported operation

* Requires scratch space

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

7.15.5. Uploading a local disk image to a block storage DataVolume

You can upload a local disk image into a block DataVolume by using the **virtctl** command-line utility.

In this workflow, you create a local block device to use as a PersistentVolume, associate this block volume with an **upload** DataVolume, and use **virtctl** to upload the local disk image into the DataVolume.

7.15.5.1. Prerequisites

- [Install](#) the **kubevirt-virtctl** package.
- If you require scratch space according to the [CDI supported operations matrix](#), you must first [define a StorageClass or prepare CDI scratch space](#) for this operation to complete successfully.

7.15.5.2. About DataVolumes

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

7.15.5.3. About block PersistentVolumes

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

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

7.15.5.4. Creating a local block PersistentVolume

Create a local block PersistentVolume (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 configuration 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** configuration 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 StorageClass 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 filename of the PersistentVolume created in the previous step.

7.15.5.5. Creating an upload DataVolume

You can manually create a DataVolume with an **upload** data source to use for uploading local disk images.

Procedure

1. Create a DataVolume configuration that specifies **spec: source: upload{}**:

```

apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: <upload-datavolume> 1
spec:
  source:
    upload: {}
  pvc:
    accessModes:
      - ReadWriteOnce
  resources:
    requests:
      storage: <2Gi> 2

```

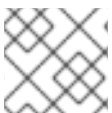
- 1 The name of the DataVolume.
- 2 The size of the DataVolume. Ensure that this value is greater than or equal to the size of the disk that you upload.

2. Create the DataVolume by running the following command:

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

7.15.5.6. Uploading a local disk image to a DataVolume

You can use the **virtctl** CLI utility to upload a local disk image from a client machine to a DataVolume (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

- A virtual machine disk image, in RAW, ISO, or QCOW2 format, optionally compressed by using **xz** or **gz**.
- 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 DataVolume that you want to use. If this DataVolume does not exist, it is created automatically.

- The size of the DataVolume, 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 DataVolume.
- 2** The size of the DataVolume. 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 DataVolume, 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 DataVolume was created, view all DataVolume objects by running the following command:

```
$ oc get dvs
```

7.15.5.7. CDI supported operations matrix

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

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

✓ Supported operation

□ Unsupported operation

* Requires scratch space

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

7.15.6. 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 DataVolume. After the cloning operation is complete, you can [edit the virtual machine configuration](#) so that it uses the new DataVolume, or [add the new DataVolume to another virtual machine](#).



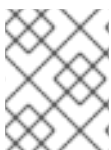
NOTE

Users without the **cluster-admin** role require [additional user permissions](#) in order to clone volumes across namespaces.

7.15.6.1. 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 PersistentVolumeClaim (PVC).

To ensure the virtual machine disk is cloned to the correct node, you must either create a new PersistentVolume (PV) or identify one on the correct node. Apply a unique label to the PV so that it can be referenced by the DataVolume.



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> ❶
  annotations:
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 10Gi ❷
  local:
    path: /mnt/local-storage/local/disk1 ❸
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
            - node01 ❹
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local
  volumeMode: Filesystem

```

- ❶ The name of the PV.
 - ❷ 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.
 - ❸ The mount path on the node.
 - ❹ The name of the node where you want to create the PV.
- Identify a PV that already exists on the target node. You can identify the node where a PV is provisioned by viewing the **nodeAffinity** field in its configuration:

```
$ oc get pv <destination-pv> -o yaml
```

The following snippet shows that the PV is on **node01**:

Example output

```

...
spec:
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname ❶
          operator: In
          values:
            - node01 ❷
...

```

- ❶ The **kubernetes.io/hostname** key uses the node host name to select a node.

2 The host name of the node.

2. Add a unique label to the PV:

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

3. Create a DataVolume 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/v1alpha1
kind: DataVolume
metadata:
  name: <clone-datavolume> 1
spec:
  source:
    pvc:
      name: "<source-vm-disk>" 2
      namespace: "<source-namespace>" 3
    pvc:
      accessModes:
        - ReadWriteOnce
      selector:
        matchLabels:
          node: node01 4
      resources:
        requests:
          storage: <10Gi> 5
```

1 The name of the new DataVolume.

2 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**.

3 The namespace where the source PVC exists.

4 The label that you applied to the PV in the previous step.

5 The size of the destination PV.

4. Start the cloning operation by applying the DataVolume manifest to your cluster:

```
$ oc apply -f <clone-datavolume.yaml>
```

The DataVolume clones the PVC of the virtual machine into the PV on the specific node.

7.15.7. 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.

7.15.7.1. About DataVolumes

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

7.15.7.2. Creating a blank disk image with DataVolumes

You can create a new blank disk image in a PersistentVolumeClaim by customizing and deploying a DataVolume configuration file.

Prerequisites

- At least one available PersistentVolume.
- Install the OpenShift CLI (**oc**).

Procedure

1. Edit the DataVolume configuration file:

```
apiVersion: cdi.kubevirt.io/v1alpha1
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
```

7.15.7.3. Template: DataVolume configuration file for blank disk images

blank-image-datavolume.yaml

```
apiVersion: cdi.kubevirt.io/v1alpha1
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

```

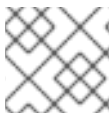
7.15.8. Storage defaults for DataVolumes

The **kubevirt-storage-class-defaults** ConfigMap provides *access mode* and *volume mode* defaults for DataVolumes. You can edit or add storage class defaults to the ConfigMap in order to create DataVolumes in the web console that better match the underlying storage.

7.15.8.1. About storage settings for DataVolumes

DataVolumes require a defined *access mode* and *volume mode* to be created in the web console. These storage settings are configured by default with a **ReadWriteOnce** access mode and **Filesystem** volume mode.

You can modify these settings by editing the **kubevirt-storage-class-defaults** ConfigMap in the **openshift-cnv** namespace. You can also add settings for other storage classes in order to create DataVolumes in the web console for different storage types.



NOTE

You must configure storage settings that are supported by the underlying storage.

All DataVolumes that you create in the web console use the default storage settings unless you specify a storage class that is also defined in the ConfigMap.

7.15.8.1.1. Access modes

DataVolumes support the following access modes:

- **ReadWriteOnce**: The volume can be mounted as read-write by a single node. **ReadWriteOnce** has greater versatility and is the default setting.
- **ReadWriteMany**: The volume can be mounted as read-write by many nodes. **ReadWriteMany** is required for some features, such as live migration of virtual machines between nodes.

ReadWriteMany is recommended if the underlying storage supports it.

7.15.8.1.2. Volume modes

The volume mode defines if a volume is intended to be used with a formatted filesystem or to remain in raw block state. DataVolumes support the following volume modes:

- **Filesystem**: Creates a filesystem on the DataVolume. This is the default setting.
- **Block**: Creates a block DataVolume. Only use **Block** if the underlying storage supports it.

7.15.8.2. Editing the kubevirt-storage-class-defaults config map in the web console

Modify the storage settings for DataVolumes by editing the **kubevirt-storage-class-defaults** ConfigMap in the **openshift-cnv** namespace. You can also add settings for other storage classes in order to create DataVolumes in the web console for different storage types.



NOTE

You must configure storage settings that are supported by the underlying storage.

Procedure

1. Click **Workloads** → **Config Maps** from the side menu.
2. In the **Project** list, select **openshift-cnv**.
3. Click **kubevirt-storage-class-defaults** to open the **Config Map Overview**.
4. Click the **YAML** tab to display the editable configuration.
5. Update the **data** values with the storage configuration that is appropriate for your underlying storage:

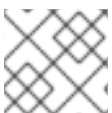
```
...
data:
  accessMode: ReadWriteOnce 1
  volumeMode: Filesystem 2
  <new>.accessMode: ReadWriteMany 3
  <new>.volumeMode: Block 4
```

- 1 The default accessMode is **ReadWriteOnce**.
- 2 The default volumeMode is **Filesystem**.
- 3 If you add an access mode for a storage class, replace the **<new>** part of the parameter with the storage class name.
- 4 If you add a volume mode for a storage class, replace the **<new>** part of the parameter with the storage class name.

6. Click **Save** to update the config map.

7.15.8.3. Editing the kubevirt-storage-class-defaults config map in the CLI

Modify the storage settings for DataVolumes by editing the **kubevirt-storage-class-defaults** ConfigMap in the **openshift-cnv** namespace. You can also add settings for other storage classes in order to create DataVolumes in the web console for different storage types.



NOTE

You must configure storage settings that are supported by the underlying storage.

Procedure

1. Edit the ConfigMap by running the following command:

```
$ oc edit configmap kubevirt-storage-class-defaults -n openshift-cnv
```

2. Update the **data** values of the ConfigMap:

```
...
data:
  accessMode: ReadWriteOnce 1
  volumeMode: Filesystem 2
  <new>.accessMode: ReadWriteMany 3
  <new>.volumeMode: Block 4
```

- 1 The default accessMode is **ReadWriteOnce**.
- 2 The default volumeMode is **Filesystem**.
- 3 If you add an access mode for storage class, replace the **<new>** part of the parameter with the storage class name.
- 4 If you add a volume mode for a storage class, replace the **<new>** part of the parameter with the storage class name.

3. Save and exit the editor to update the config map.

7.15.8.4. Example of multiple storage class defaults

The following YAML file is an example of a **kubevirt-storage-class-defaults** ConfigMap that has storage settings configured for two storage classes, **migration** and **block**.

Ensure that all settings are supported by your underlying storage before you update the ConfigMap.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: kubevirt-storage-class-defaults
  namespace: openshift-cnv
...
data:
  accessMode: ReadWriteOnce
  volumeMode: Filesystem
  nfs-sc.accessMode: ReadWriteMany
  nfs-sc.volumeMode: Filesystem
  block-sc.accessMode: ReadWriteMany
  block-sc.volumeMode: Block
```

7.15.9. 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.

7.15.9.1. 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 `DataVolume` that is attached to a virtual machine, or attached directly to a virtual machine as an ephemeral `containerDisk` volume.

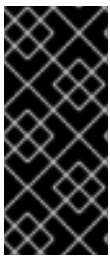
7.15.9.1.1. Importing a container disk into a PVC by using a `DataVolume`

Use the Containerized Data Importer (CDI) to import the container disk into a PVC by using a `DataVolume`. You can then attach the `DataVolume` to a virtual machine for persistent storage.

7.15.9.1.2. 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 filesystems such as CD-ROMs or for disposable virtual machines.



IMPORTANT

Using `containerDisk` volumes for read-write filesystems 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.

7.15.9.2. 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 `DataVolume` and attach it to a virtual machine, or you can attach the container disk directly to a virtual machine as an ephemeral `containerDisk` volume.

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.

7.15.9.3. Disabling TLS for a container registry to use as insecure registry

You can disable TLS (transport layer security) for a container registry by adding the registry to the **cdi-insecure-registries** ConfigMap.

Prerequisites

- Log in to the cluster as a user with the **cluster-admin** role.

Procedure

- Add the registry to the **cdi-insecure-registries** ConfigMap in the **cdi** namespace.

```
$ oc patch configmap cdi-insecure-registries -n cdi \
  --type merge -p '{"data":{"mykey": "<insecure-registry-host>:5000"}}' 1
```

- 1** Replace `<insecure-registry-host>` with the registry hostname.

7.15.9.4. 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.

7.15.10. Preparing CDI scratch space

7.15.10.1. About DataVolumes

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

7.15.10.2. Understanding scratch space

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

The CDIConfig object allows you to define which StorageClass to use to bind the scratch space PVC by setting the **scratchSpaceStorageClass** in the **spec:** section of the CDIConfig object.

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



NOTE

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

Manual provisioning

If there are no storage classes, the CDI will use 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 will remain in a **Pending** state until an appropriate PVC is made available or until a timeout function kills the pod.

7.15.10.3. CDI operations that require scratch space

Type	Reason
Registry imports	The 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.

Type	Reason
HTTP imports of custom certificates	QEMU-IMG inadequately handles custom certificates of HTTPS endpoints. Instead, the CDI downloads the image to scratch space before passing the file to QEMU-IMG.

7.15.10.4. Defining a StorageClass in the CDI configuration

Define a StorageClass in the CDI configuration to dynamically provision scratch space for CDI operations.

Procedure

- Use the **oc** client to edit the **cdiconfig/config** and add or edit the **spec: scratchSpaceStorageClass** to match a StorageClass in the cluster.

```
$ oc edit cdiconfig/config
```

```
API Version: cdi.kubevirt.io/v1alpha1
kind: CDIConfig
metadata:
  name: config
...
spec:
  scratchSpaceStorageClass: "<storage_class>"
...
```

7.15.10.5. CDI supported operations matrix

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

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

✓ Supported operation

Unsupported operation

* Requires scratch space

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

Additional resources

- See the [Dynamic provisioning](#) section for more information on StorageClasses and how these are defined in the cluster.

7.15.11. Re-using persistent volumes

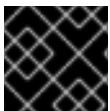
In order to re-use a statically provisioned persistent volume (PV), you must first reclaim the volume. This involves deleting the PV so that the storage configuration can be re-used.

7.15.11.1. 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.

7.15.11.2. Reclaiming statically provisioned persistent volumes

Reclaim a statically provisioned persistent volume (PV) by unbinding the persistent volume claim (PVC) and deleting the PV. You might also need to manually delete the shared storage.

Reclaiming a statically provisioned PV is dependent on the underlying storage. This procedure provides a general approach that might need to be customized depending on your storage.

Procedure

1. Ensure that the reclaim policy of the PV is set to **Retain**:
 - a. Check the reclaim policy of the PV:

```
$ oc get pv <pv_name> -o yaml | grep 'persistentVolumeReclaimPolicy'
```

- b. If the **persistentVolumeReclaimPolicy** is not set to **Retain**, edit the reclaim policy with the following command:

```
$ oc patch pv <pv_name> -p '{"spec":{"persistentVolumeReclaimPolicy":"Retain"}}'
```

2. Ensure that no resources are using the PV:

```
$ oc describe pvc <pvc_name> | grep 'Mounted By:'
```

Remove any resources that use the PVC before continuing.

3. Delete the PVC to release the PV:

```
$ oc delete pvc <pvc_name>
```

4. Optional: Export the PV configuration to a YAML file. If you manually remove the shared storage later in this procedure, you can refer to this configuration. You can also use **spec** parameters in this file as the basis to create a new PV with the same storage configuration after you reclaim the PV:

```
$ oc get pv <pv_name> -o yaml > <file_name>.yaml
```

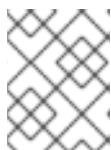
5. Delete the PV:

```
$ oc delete pv <pv_name>
```

6. Optional: Depending on the storage type, you might need to remove the contents of the shared storage folder:

```
$ rm -rf <path_to_share_storage>
```

7. Optional: Create a PV that uses the same storage configuration as the deleted PV. If you exported the reclaimed PV configuration earlier, you can use the **spec** parameters of that file as the basis for a new PV manifest:



NOTE

To avoid possible conflict, it is good practice to give the new PV object a different name than the one that you deleted.

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

Additional Resources

- [Configuring local storage for virtual machines](#)
- The OpenShift Container Platform Storage documentation has more information on [Persistent Storage](#).

7.15.12. Deleting DataVolumes

You can manually delete a DataVolume by using the **oc** command-line interface.



NOTE

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

7.15.12.1. About DataVolumes

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

7.15.12.2. Listing all DataVolumes

You can list the DataVolumes in your cluster by using the **oc** command-line interface.

Procedure

- List all DataVolumes by running the following command:

```
$ oc get dvs
```

7.15.12.3. Deleting a DataVolume

You can delete a DataVolume by using the **oc** command-line interface (CLI).

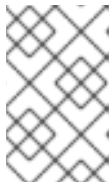
Prerequisites

- Identify the name of the DataVolume that you want to delete.

Procedure

- Delete the DataVolume 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 8. VIRTUAL MACHINE TEMPLATES


8.1. CREATING VIRTUAL MACHINE TEMPLATES

You can use virtual machine templates to create multiple virtual machines that have similar configurations. After a template is created, reference the template when creating virtual machines.

8.1.1. Creating a virtual machine template with the interactive wizard in the web console

The web console features an interactive wizard that guides you through the **General**, **Networking**, **Storage**, **Advanced**, and **Review** steps to simplify the process of creating virtual machine templates. All required fields are marked with a *. The wizard prevents you from moving to the next step until you provide values in the required fields.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machine Templates** tab.
3. Click **Create Template** and select **New with Wizard**.
4. Fill in all required fields in the **General** step.
5. Click **Next** to progress to the **Networking** screen. A NIC that is named **nic0** is attached by default.
 - a. Optional: Click **Add Network Interface** to create additional NICs.
 - b. Optional: You can remove any or all NICs by clicking the Options menu  and selecting **Delete**. Virtual machines created from a template do not need a NIC attached. NICs can be created after a virtual machine has been created.
6. Click **Next** to progress to the **Storage** screen.
 - a. Optional: Click **Add Disk** to create additional disks.
 - b. Optional: Click a disk to modify available fields. Click the ✓ button to save the changes.
 - c. Optional: Click **Disk** to choose an available disk from the **Select Storage** list.



NOTE

If either **URL** or **Container** are selected as the **Source** in the **General** step, a **rootdisk** disk is created and attached to virtual machines as the **Bootable Disk**. You can modify the **rootdisk** but you cannot remove it.

A **Bootable Disk** is not required for virtual machines provisioned from a **PXE** source if there are no disks attached to the virtual machine. If one or more disks are attached to the virtual machine, you must select one as the **Bootable Disk**.

- Click **Create Virtual Machine Template** > The **Results** screen displays the JSON configuration file for the virtual machine template.

The template is listed in the **Virtual Machine Templates** tab.

8.1.2. Virtual machine template interactive wizard fields

The following tables describe the fields for the **Basic Settings**, **Networking**, and **Storage** panes in the **Create Virtual Machine Template** interactive wizard.

8.1.2.1. Virtual machine template wizard fields

Name	Parameter	Description
Source	PXE	Provision virtual machine from PXE menu. Requires a PXE-capable NIC in the cluster.
	URL	Provision virtual machine from an image available from an HTTP or S3 endpoint.
	Container	Provision virtual machine from a bootable operating system container located in a registry accessible from the cluster. Example: kubevirt/cirros-registry-disk-demo .
	Disk	Provision virtual machine from a disk.
Operating System		The primary operating system that is selected for the virtual machine.
Flavor	small, medium, large, tiny, Custom	Presets that determine the amount of CPU and memory allocated to the virtual machine. The presets displayed for Flavor are determined by the operating system.
Memory		Size in GiB of the memory allocated to the virtual machine.
CPUs		The amount of CPU allocated to the virtual machine.
Workload Profile	High Performance	A virtual machine configuration that is optimized for high-performance workloads.

Name	Parameter	Description
	Server	A profile optimized to run server workloads.
	Desktop	A virtual machine configuration for use on a desktop.
Name		The name can contain lowercase letters (a-z), numbers (0-9), and hyphens (-), up to a maximum of 253 characters. The first and last characters must be alphanumeric. The name must not contain uppercase letters, spaces, periods (.), or special characters.
Description		Optional description field.

8.1.2.2. Cloud-init fields

Name	Description
Hostname	Sets a specific host name for the virtual machine.
Authenticated 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.

8.1.2.3. Networking fields

Name	Description
Name	Name for the Network Interface Card.
Model	Indicates the model of the Network Interface Card. Supported values are e1000 , e1000e , ne2k_pci , pcnet , rtl8139 , and virtIO .
Network	List of available NetworkAttachmentDefinition objects.


Name	Description
Type	List of available binding methods. For the default Pod network, masquerade is the only recommended binding method. For secondary networks, use the bridge binding method. The masquerade method is not supported for non-default networks.
MAC Address	MAC address for the Network Interface Card. If a MAC address is not specified, an ephemeral address is generated for the session.

8.1.2.4. Storage fields

Name	Description
Source	Select a blank disk for the virtual machine or choose from the options available: URL , Container , Attach Cloned Disk , or Attach Disk . To select an existing disk and attach it to the virtual machine, choose Attach Cloned Disk or Attach Disk from a list of available PersistentVolumeClaims (PVCs).
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 (GiB)	Size, in GiB, of the disk.
Interface	Type of disk device. Supported interfaces are virtIO , SATA , and SCSI .
Storage Class	The StorageClass that is used to create the disk.
Advanced → Volume Mode	
Defines whether the persistent volume uses a formatted file system or raw block state. Default is Filesystem .	Advanced → Access Mode
	Access mode of the persistent volume. Supported access modes are Single User (RWO) , Shared Access (RWX) , and Read Only (ROX) .

Advanced storage settings

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

Name	Parameter	Description
Volume Mode	Filesystem	Stores the virtual disk on a filesystem-based volume.
	Block	Stores the virtual disk directly on the block volume. Only use Block if the underlying storage supports it.
Access Mode	Single User (RWO)	The disk can be mounted as read/write by a single node.
	Shared Access (RWX)	The disk can be mounted as read/write by many nodes. <div style="display: flex; align-items: center;">  <div> <p>NOTE</p> <p>This is required for some features, such as live migration of virtual machines between nodes.</p> </div> </div>
	Read Only (ROX)	The disk can be mounted as read-only by many nodes.

8.2. EDITING VIRTUAL MACHINE TEMPLATES

You can update a virtual machine template in the web console, either by editing the full configuration in the YAML editor or by editing a subset of the parameters in the **Virtual Machine Template Overview** screen.

8.2.1. Editing a virtual machine template in the web console

Edit select values of a virtual machine template in the **Virtual Machine Template Overview** screen of the web console by clicking on the pencil icon next to the relevant field. Other values can be edited using the CLI.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machine Templates** tab.
3. Select a virtual machine template to open the **Virtual Machine Template Overview** screen.
4. Click the **Details** tab.
5. Click the pencil icon to make a field editable.

6. Make the relevant changes and click **Save**.

Editing a virtual machine template will not affect virtual machines already created from that template.

8.2.2. Editing virtual machine template YAML configuration in the web console

You can edit the YAML configuration of a virtual machine template from the web console.

Not all parameters can be modified. If you click **Save** with an invalid configuration, an error message indicates the parameter that cannot be modified.



NOTE

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

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machine Templates** tab.
3. Select a template.
4. Click the **YAML** tab to display the editable configuration.
5. Edit the file and click **Save**.

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

8.2.3. Adding a virtual disk to a virtual machine template

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

Procedure

1. From the **Virtual Machine Templates** tab, select your virtual machine template.
2. Select the **Disks** tab.
3. Click **Add Disks** to open the **Add Disk** window.
4. In the **Add Disk** window, specify **Source**, **Name**, **Size**, **Interface**, and **Storage Class**.
 - a. Optional: In the **Advanced** list, specify the **Volume Mode** and **Access Mode** for the virtual disk. If you do not specify these parameters, the system uses the default values from the **kubevirt-storage-class-defaults** ConfigMap.
5. Use the drop-down lists and check boxes to edit the disk configuration.
6. Click **OK**.

8.2.4. Adding a network interface to a virtual machine template

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

Procedure

1. From the **Virtual Machine Templates** tab, select the virtual machine template.
2. Select the **Network Interfaces** tab.
3. Click **Add Network Interface**.
4. In the **Add Network Interface** window, specify the **Name**, **Model**, **Network**, **Type**, and **MAC Address** of the network interface.
5. Click **Add** to add the network interface.
6. Restart the virtual machine to enable access.
7. Edit the drop-down lists and check boxes to configure the network interface.
8. Click **Save Changes**.
9. Click **OK**.

The new network interface displays at the top of the **Create Network Interface** list until the user restarts it.

The new network interface has a **Pending VM restart** Link State until you restart the virtual machine. Hover over the Link State to display more detailed information.

The **Link State** is set to **Up** by default when the network interface card is defined on the virtual machine and connected to the network.

8.2.5. Editing CD-ROMs for Virtual Machine Templates

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

Procedure

1. From the **Virtual Machine Templates** tab, select your virtual machine template.
2. Select the **Overview** tab.
3. To add or edit a CD-ROM configuration, click the pencil icon to the right of the **CD-ROMs** label. The **Edit CD-ROM** window opens.
 - If CD-ROMs are unavailable for editing, the following message displays: **The virtual machine doesn't have any CD-ROMs attached.**
 - If there are CD-ROMs available, you can remove a CD-ROM by clicking **-**.
4. In the **Edit CD-ROM** window, do the following:
 - a. Select the type of CD-ROM configuration from the drop-down list for **Media Type**. CD-ROM configuration types are **Container**, **URL**, and **Persistent Volume Claim**.
 - b. Complete the required information for each **Type**.

- c. When all CD-ROMs are added, click **Save**.

8.3. ENABLING DEDICATED RESOURCES FOR VIRTUAL MACHINE TEMPLATES

Virtual machines can have resources of a node, such as CPU, dedicated to them in order to improve performance.

8.3.1. 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.

8.3.2. Prerequisites

- The [CPU Manager](#) must be configured on the node. Verify that the node has the `cpumanager = true` label before scheduling virtual machine workloads.

8.3.3. Enabling dedicated resources for a virtual machine template

You can enable dedicated resources for a virtual machine template in the **Virtual Machine Template Overview** page of the web console.

Procedure

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

8.4. DELETING A VIRTUAL MACHINE TEMPLATE

You can delete a virtual machine template in the web console.


8.4.1. Deleting a virtual machine template in the web console

Deleting a virtual machine template permanently removes it from the cluster.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machine Templates** tab.

3. You can delete the virtual machine template from this pane, which makes it easier to perform actions on multiple templates in the one pane, or from the **Virtual Machine Template Details** pane where you can view comprehensive details of the selected template:

- Click the Options menu  of the template to delete and select **Delete Template**.
- Click the template name to open the **Virtual Machine Template Details** pane and click **Actions → Delete Template**.

4. In the confirmation pop-up window, click **Delete** to permanently delete the template.

CHAPTER 9. LIVE MIGRATION

9.1. VIRTUAL MACHINE LIVE MIGRATION

9.1.1. Prerequisites

- Before using LiveMigration, ensure that the storage class used by the virtual machine has a PersistentVolumeClaim (PVC) with a shared ReadWriteMany (RWX) access mode. See [Storage defaults for DataVolumes](#) to ensure your storage settings are correct.

9.1.2. Understanding live migration

Live migration is the process of moving a running virtual machine instance to another node in the cluster without interrupting the virtual workload or access. This can be a manual process, if you select virtual machine instance to migrate to another node, or an automatic process, if the virtual machine instance has a **LiveMigrate** eviction strategy and the node on which it is running is placed into maintenance.



IMPORTANT

Virtual machines must have a PersistentVolumeClaim (PVC) with a shared ReadWriteMany (RWX) access mode to be live migrated.

9.1.3. Updating access mode for LiveMigration

For LiveMigration 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](#)
- [Node maintenance mode](#)
- [Live migration limiting](#)

9.2. LIVE MIGRATION LIMITS AND TIMEOUTS

Live migration limits and timeouts are applied so that migration processes do not overwhelm the cluster. Configure these settings by editing the **kubevirt-config** configuration file.

9.2.1. Configuring live migration limits and timeouts

Configure live migration limits and timeouts for the cluster by adding updated key:value fields to the **kubevirt-config** configuration file, which is located in the **openshift-cnv** namespace.

Procedure

- Edit the **kubevirt-config** configuration file and add the necessary live migration parameters. The following example shows the default values:

```
$ oc edit configmap kubevirt-config -n openshift-cnv
```

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kubevirt-config
  namespace: kubevirt
  labels:
    kubevirt.io: ""
data:
  feature-gates: "LiveMigration"
  migrations: |-
    parallelMigrationsPerCluster: 5
    parallelOutboundMigrationsPerNode: 2
    bandwidthPerMigration: 64Mi
    completionTimeoutPerGiB: 800
    progressTimeout: 150
```

9.2.2. Cluster-wide live migration limits and timeouts

Table 9.1. 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 will be canceled if it has not completed in this time, in seconds per GiB of memory. For example, a virtual machine instance with 6GiB memory will timeout 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

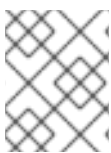
Parameter	Description	Default
progressTimeout	The migration will be canceled if memory copy fails to make progress in this time, in seconds.	150

9.3. 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.

9.3.1. Initiating live migration of a virtual machine instance in the web console

Migrate a running virtual machine instance to a different node in the cluster.




NOTE

The **Migrate Virtual Machine** action is visible to all users but only admin users can initiate a virtual machine migration.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. You can initiate the migration from this screen, which makes it easier to perform actions on multiple virtual machines in the one screen, or from the **Virtual Machine Overview** screen where you can view comprehensive details of the selected virtual machine:

- Click the Options menu  at the end of virtual machine and select **Migrate Virtual Machine**.
- Click the virtual machine name to open the **Virtual Machine Overview** screen and click **Actions** → **Migrate Virtual Machine**

4. Click **Migrate** to migrate the virtual machine to another node.

9.3.2. 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/v1alpha3
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](#)

9.4. MONITORING LIVE MIGRATION OF A VIRTUAL MACHINE INSTANCE

You can monitor the progress of a live migration of a virtual machine instance from either the web console or the CLI.

9.4.1. Monitoring live migration of a virtual machine instance in the web console

For the duration of the migration, the virtual machine has a status of **Migrating**. This status is displayed in the **Virtual Machines** tab or in the **Virtual Machine Overview** screen for the migrating virtual machine.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.

9.4.2. Monitoring live migration of a virtual machine instance in the CLI

The status of the virtual machine migration is stored in the **Status** component of the **VirtualMachineInstance** configuration.

Procedure

- Use the **oc describe** command on the migrating virtual machine instance:

```
$ oc describe vmi vmi-fedora
```

Example output

```
...
```

```

Status:
Conditions:
  Last Probe Time: <nil>
  Last Transition Time: <nil>
  Status: True
  Type: LiveMigratable
Migration Method: LiveMigration
Migration State:
  Completed: true
  End Timestamp: 2018-12-24T06:19:42Z
  Migration UID: d78c8962-0743-11e9-a540-fa163e0c69f1
  Source Node: node2.example.com
  Start Timestamp: 2018-12-24T06:19:35Z
  Target Node: node1.example.com
  Target Node Address: 10.9.0.18:43891
  Target Node Domain Detected: true


```

9.5. CANCELLING THE LIVE MIGRATION OF A VIRTUAL MACHINE INSTANCE


Cancel the live migration so that the virtual machine instance remains on the original node.

You can cancel a live migration from either the web console or the CLI.

9.5.1. Cancelling live migration of a virtual machine instance in the web console

You can cancel a live migration of the virtual machine instance using the Options menu  found on each virtual machine in the **Virtualization** → **Virtual Machines** tab, or from the **Actions** menu available on all tabs in the **Virtual Machine Overview** screen.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. You can cancel the migration from this screen, which makes it easier to perform actions on multiple virtual machines, or from the **Virtual Machine Overview** screen where you can view comprehensive details of the selected virtual machine:
 - Click the Options menu  at the end of virtual machine and select **Cancel Virtual Machine Migration**.
 - Select a virtual machine name to open the **Virtual Machine Overview** screen and click **Actions** → **Cancel Virtual Machine Migration**
4. Click **Cancel Migration** to cancel the virtual machine live migration.

9.5.2. Cancelling live migration of a virtual machine instance in the CLI

Cancel the live migration of a virtual machine instance by deleting the **VirtualMachineInstanceMigration** object associated with the migration.

Procedure

- Delete the **VirtualMachineInstanceMigration** object that triggered the live migration, **migration-job** in this example:

```
$ oc delete vmim migration-job
```

9.6. CONFIGURING VIRTUAL MACHINE EVICTION STRATEGY

The **LiveMigrate** eviction strategy ensures that a virtual machine instance is not interrupted if the node is placed into maintenance or drained. Virtual machines instances with this eviction strategy will be live migrated to another node.

9.6.1. Configuring custom virtual machines with the LiveMigration eviction strategy

You only need to configure the **LiveMigration** eviction strategy on custom virtual machines. Common templates have this eviction strategy configured by default.

Procedure

1. Add the **evictionStrategy: LiveMigrate** option to the **spec** 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/v1alpha3
kind: VirtualMachine
metadata:
  name: custom-vm
spec:
  terminationGracePeriodSeconds: 30
  evictionStrategy: LiveMigrate
  domain:
    resources:
      requests:
  ...
```

2. Restart the virtual machine for the update to take effect:

```
$ virtctl restart <custom-vm> -n <my-namespace>
```

CHAPTER 10. NODE MAINTENANCE

10.1. 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.

10.1.1. Automatic renewal of TLS certificates

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

10.2. NODE MAINTENANCE MODE

10.2.1. Understanding node maintenance mode

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 will be deleted on the node and recreated on another node.



IMPORTANT

Virtual machines must have a PersistentVolumeClaim (PVC) with a shared ReadWriteMany (RWX) access mode to be live migrated.

Additional resources:

- [Virtual machine live migration](#)
- [Configuring virtual machine eviction strategy](#)

10.3. SETTING A NODE TO MAINTENANCE MODE

10.3.1. Understanding node maintenance mode

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 will be deleted on the node and recreated on another node.




IMPORTANT

Virtual machines must have a PersistentVolumeClaim (PVC) with a shared ReadWriteMany (RWX) access mode to be live migrated.


Place a node into maintenance from either the web console or the CLI.

10.3.2. Setting a node to maintenance mode in the web console

Set a node to maintenance mode using the Options menu  found on each node in the **Compute → Nodes** list, or using the **Actions** control of the **Node Details** screen.

Procedure

1. In the OpenShift Virtualization console, click **Compute → Nodes**.
2. You can set the node to maintenance from this screen, which makes it easier to perform actions on multiple nodes in the one screen or from the **Node Details** screen where you can view comprehensive details of the selected node:

- Click the Options menu  at the end of the node and select **Start Maintenance**.
- Click the node name to open the **Node Details** screen and click **Actions → Start Maintenance**.

3. Click **Start Maintenance** in the confirmation window.

The node will live migrate virtual machine instances that have the **LiveMigration** eviction strategy, and the node is no longer schedulable. All other pods and virtual machines on the node are deleted and recreated on another node.

10.3.3. Setting a node to maintenance mode in the CLI

Set a node to maintenance mode by creating a **NodeMaintenance** Custom Resource (CR) object that references the node name and the reason for setting it to maintenance mode.

Procedure

1. Create the node maintenance CR configuration. This example uses a CR that is called **node02-maintenance.yaml**:

```
apiVersion: nodemaintenance.kubevirt.io/v1beta1
kind: NodeMaintenance
```



```

metadata:
  name: node02-maintenance
spec:
  nodeName: node02
  reason: "Replacing node02"

```

2. Create the **NodeMaintenance** object in the cluster:

```
$ oc apply -f <node02-maintenance.yaml>
```

The node live migrates virtual machine instances that have the **LiveMigration** eviction strategy, and taint the node so that it is no longer schedulable. All other pods and virtual machines on the node are deleted and recreated on another node.

Additional resources:

- [Resuming a node from maintenance mode](#)

10.4. RESUMING A NODE FROM MAINTENANCE MODE

Resuming a node brings it out of maintenance mode and schedulable again.


Resume a node from maintenance from either the web console or the CLI.

10.4.1. Resuming a node from maintenance mode in the web console

Resume a node from maintenance mode using the Options menu  found on each node in the **Compute** → **Nodes** list, or using the **Actions** control of the **Node Details** screen.

Procedure

1. In the OpenShift Virtualization console, click **Compute** → **Nodes**.
2. You can resume the node from this screen, which makes it easier to perform actions on multiple nodes in the one screen, or from the **Node Details** screen where you can view comprehensive details of the selected node:

- Click the Options menu  at the end of the node and select **Stop Maintenance**.
- Click the node name to open the **Node Details** screen and click **Actions** → **Stop Maintenance**.

3. Click **Stop Maintenance** in the confirmation window.

The node becomes schedulable, but virtual machine instances that were running on the node prior to maintenance will not automatically migrate back to this node.

10.4.2. Resuming a node from maintenance mode in the CLI

Resume a node from maintenance mode and make it schedulable again by deleting the **NodeMaintenance** object for the node.

Procedure

1. Find the **NodeMaintenance** object:

```
$ oc get nodemaintenance
```

2. Optional: Inspect the **NodeMaintenance** object to ensure it is associated with the correct node:

```
$ oc describe nodemaintenance <node02-maintenance>
```

Example output

```
Name:      node02-maintenance
Namespace:
Labels:
Annotations:
API Version: nodemaintenance.kubevirt.io/v1beta1
Kind:      NodeMaintenance
...
Spec:
  Node Name: node02
  Reason:   Replacing node02
```

3. Delete the **NodeMaintenance** object:

```
$ oc delete nodemaintenance <node02-maintenance>
```

CHAPTER 11. NODE NETWORKING

11.1. OBSERVING NODE NETWORK STATE

Node network state is the network configuration for all nodes in the cluster.

11.1.1. About nmstate

OpenShift Virtualization uses **nmstate** to report on and configure the state of the node network. This makes it possible to modify network policy configuration, such as by creating a Linux bridge on all nodes, by applying a single configuration manifest to the cluster.

Node networking is monitored and updated by the following objects:

NodeNetworkState

Reports the state of the network on that node.

NodeNetworkConfigurationPolicy

Describes the requested network configuration on nodes. You update the node network configuration, including adding and removing interfaces, by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

NodeNetworkConfigurationEnactment

Reports the network policies enacted upon each node.

OpenShift Virtualization supports the use of the following nmstate interface types:

- Linux Bridge
- VLAN
- Bond
- Ethernet

11.1.2. Viewing the network state of a node

A **NodeNetworkState** object exists on every node in the cluster. This object is periodically updated and captures the state of the network for that node.

Procedure

1. List all the **NodeNetworkState** objects in the cluster:

```
$ oc get nns
```

2. Inspect a **NodeNetworkState** to view the network on that node. The output in this example has been redacted for clarity:

```
$ oc get nns node01 -o yaml
```

Example output

```
apiVersion: nmstate.io/v1alpha1
```

```

kind: NodeNetworkState
metadata:
  name: node01 1
status:
  currentState: 2
  dns-resolver:
  ...
  interfaces:
  ...
  route-rules:
  ...
  routes:
  ...
lastSuccessfulUpdateTime: "2020-01-31T12:14:00Z" 3

```

- 1** The name of the **NodeNetworkState** is taken from the node.
- 2** The **currentState** contains the complete network configuration for the node, including DNS, interfaces, and routes.
- 3** Timestamp of the last successful update. This is updated periodically as long as the node is reachable and can be used to evaluate the freshness of the report.

11.2. UPDATING NODE NETWORK CONFIGURATION

You can update the node network configuration, such as adding or removing interfaces from nodes, by applying **NodeNetworkConfigurationPolicy** manifests to the cluster.

11.2.1. About nmstate

OpenShift Virtualization uses **nmstate** to report on and configure the state of the node network. This makes it possible to modify network policy configuration, such as by creating a Linux bridge on all nodes, by applying a single configuration manifest to the cluster.

Node networking is monitored and updated by the following objects:

NodeNetworkState

Reports the state of the network on that node.

NodeNetworkConfigurationPolicy

Describes the requested network configuration on nodes. You update the node network configuration, including adding and removing interfaces, by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

NodeNetworkConfigurationEnactment

Reports the network policies enacted upon each node.

OpenShift Virtualization supports the use of the following nmstate interface types:

- Linux Bridge
- VLAN
- Bond

- Ethernet

11.2.2. Creating an interface on nodes

Create an interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster. The manifest details the requested configuration for the interface.

By default, the manifest applies to all nodes in the cluster. To add the interface to specific nodes, add the **spec: nodeSelector** parameter and the appropriate **<key>:<value>** for your node selector.

Procedure

1. Create the **NodeNetworkConfigurationPolicy** manifest. The following example configures a Linux bridge on all worker nodes:

```
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: <br1-eth1-policy> 1
spec:
  nodeSelector: 2
  node-role.kubernetes.io/worker: "" 3
desiredState:
  interfaces:
    - name: br1
      description: Linux bridge with eth1 as a port 4
      type: linux-bridge
      state: up
      ipv4:
        dhcp: true
        enabled: true
      bridge:
        options:
          stp:
            enabled: false
      port:
        - name: eth1
```

- 1 Name of the Policy.
- 2 Optional: If you do not include the **nodeSelector**, the Policy applies to all nodes in the cluster.
- 3 This example uses the **node-role.kubernetes.io/worker: ""** node selector to select all worker nodes in the cluster.
- 4 Optional: Human-readable description for the interface.

2. Create the Policy:

```
$ oc apply -f <br1-eth1-policy.yaml> 1
```

- 1 File name of the Policy manifest.

Additional resources

- [Example Policy configurations for different interfaces](#)
- [Example for creating multiple interfaces in the same Policy](#)
- [Examples of different IP management methods in Policies](#)

11.2.3. Confirming Policy updates on nodes

A **NodeNetworkConfigurationPolicy** manifest describes your requested network configuration for nodes in the cluster. The Policy object includes your requested network configuration and the status of execution of the Policy on the cluster as a whole.

When you apply a Policy, a **NodeNetworkConfigurationEnactment** is created for every node in the cluster. The Enactment is a read-only object that represents the status of execution of the Policy on that node. If the Policy fails to be applied on the node, the Enactment for that node includes a traceback for troubleshooting.

Procedure

1. To confirm that a Policy has been applied to the cluster, list the Policies and their status:

```
$ oc get nncp
```

2. Optional: If a Policy is taking longer than expected to successfully configure, you can inspect the requested state and status conditions of a particular Policy:

```
$ oc get nncp <policy> -o yaml
```

3. Optional: If a policy is taking longer than expected to successfully configure on all nodes, you can list the status of the Enactments on the cluster:

```
$ oc get nnce
```

4. Optional: To view the configuration of a particular Enactment, including any error reporting for a failed configuration:

```
$ oc get nnce <node>.<policy> -o yaml
```

11.2.4. Removing an interface from nodes

Remove an interface from nodes by editing the **NodeNetworkConfigurationPolicy** object and set the **state** of the interface to **absent**.



NOTE

Deleting the Policy that added an interface does not change the configuration of the network policy on the node. Although a **NodeNetworkConfigurationPolicy** is an object in the cluster, it only represents the requested configuration. Similarly, removing an interface does not delete the Policy.

Procedure

1. Update the **NodeNetworkConfigurationPolicy** manifest used to create the interface. The following example removes a Linux bridge:

```

apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: <br1-eth1-policy> ❶
spec:
  nodeSelector: ❷
    node-role.kubernetes.io/worker: "" ❸
desiredState:
  interfaces:
    - name: br1
      type: linux-bridge
      state: absent ❹

```

- ❶ Name of the Policy.
- ❷ Optional: If you do not include the **nodeSelector**, the Policy applies to all nodes in the cluster.
- ❸ This example uses the **node-role.kubernetes.io/worker: ""** node selector to select all worker nodes in the cluster.
- ❹ Changing the state to **absent** removes the interface.

2. Update the Policy on the node and remove the interface:

```
$ oc apply -f <br1-eth1-policy.yaml> ❶
```

- ❶ File name of the Policy manifest.

11.2.5. Restoring node network configuration after removing an interface

Removing an interface from a node does not automatically restore the node network configuration to a previous state. After you remove an interface, any of the node NICs throughout the cluster that were previously attached or subordinate to the interface are placed in a **down** state. Restore the NICs by applying a new **NodeNetworkConfigurationPolicy** manifest to the cluster.

Procedure

1. Create a **NodeNetworkConfigurationPolicy** manifest that specifies the NIC and the desired state of **up**:

```

apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: eth1
spec:
  desiredState:
    interfaces:
      - name: eth1
        type: ethernet

```

```
state: up
ipv4:
  dhcp: true
  enabled: true
```

2. Apply the manifest to the cluster:

```
$ oc apply -f <eth1.yaml> 1
```

- 1 File name of the Policy manifest.

11.2.6. Example Policy configurations for different interfaces

11.2.6.1. Example: Linux bridge interface NodeNetworkConfigurationPolicy

Create a Linux bridge interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

The following YAML file is an example of a manifest for a Linux bridge interface. It includes sample values that you must replace with your own information.

```
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
      - name: br1 4
        description: Linux bridge with eth1 as a port 5
        type: linux-bridge 6
        state: up 7
        ipv4:
          dhcp: true 8
          enabled: true 9
        bridge:
          options:
            stp:
              enabled: false 10
        port:
          - name: eth1 11
```

- 1 Name of the Policy.
- 2 Optional: If you do not include the **nodeSelector**, the Policy applies to all nodes in the cluster.
- 3 This example uses a **hostname** node selector.
- 4 Name of the interface.

- 5 Optional: Human-readable description of the interface.
- 6 The type of interface. This example creates a bridge.
- 7 The requested state for the interface after creation.
- 8 Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.
- 9 Enables **ipv4** in this example.
- 10 Disables **stp** in this example.
- 11 The node NIC to which the bridge attaches.

11.2.6.2. Example: VLAN interface NodeNetworkConfigurationPolicy

Create a VLAN interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

The following YAML file is an example of a manifest for a VLAN interface. It includes samples values that you must replace with your own information.

```
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: vlan-eth1-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
      - name: eth1.102 4
        description: VLAN using eth1 5
        type: vlan 6
        state: up 7
        vlan:
          base-iface: eth1 8
          id: 102 9
```

- 1 Name of the Policy.
- 2 Optional: If you do not include the **nodeSelector**, the Policy applies to all nodes in the cluster.
- 3 This example uses a **hostname** node selector.
- 4 Name of the interface.
- 5 Optional: Human-readable description of the interface.
- 6 The type of interface. This example creates a VLAN.
- 7 The requested state for the interface after creation.

- 8 The node NIC to which the VLAN is attached.
- 9 The VLAN tag.

11.2.6.3. Example: Bond interface NodeNetworkConfigurationPolicy

Create a bond interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.



NOTE

OpenShift Virtualization only supports the following bond modes:

- mode=1 active-backup
- mode=5 balance-tlb
- mode=6 balance-alb

The following YAML file is an example of a manifest for a bond interface. It includes samples values that you must replace with your own information.

```
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond0-eth1-eth2-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
      - name: bond0 4
        description: Bond enslaving eth1 and eth2 5
        type: bond 6
        state: up 7
        ipv4:
          dhcp: true 8
          enabled: true 9
        link-aggregation:
          mode: active-backup 10
          options:
            miimon: '140' 11
          slaves: 12
            - eth1
            - eth2
        mtu: 1450 13
```

- 1 Name of the Policy.
- 2 Optional: If you do not include the **nodeSelector**, the Policy applies to all nodes in the cluster.
- 3 This example uses a **hostname** node selector.

- 4 Name of the interface.
- 5 Optional: Human-readable description of the interface.
- 6 The type of interface. This example creates a bond.
- 7 The requested state for the interface after creation.
- 8 Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.
- 9 Enables **ipv4** in this example.
- 10 The driver mode for the bond. This example uses an active backup mode.
- 11 Optional: This example uses **miimon** to inspect the bond link every 140ms.
- 12 The subordinate node NICs in the bond.
- 13 Optional: The maximum transmission unit (MTU) for the bond. If not specified, this value is set to **1500** by default.

11.2.6.4. Example: Ethernet interface NodeNetworkConfigurationPolicy

Configure an Ethernet interface on nodes in the cluster by applying a **NodeNetworkConfigurationPolicy** manifest to the cluster.

The following YAML file is an example of a manifest for an Ethernet interface. It includes sample values that you must replace with your own information.

```
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: eth1-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
      - name: eth1 4
        description: Configuring eth1 on node01 5
        type: ethernet 6
        state: up 7
        ipv4:
          dhcp: true 8
          enabled: true 9
```

- 1 Name of the Policy.
- 2 Optional: If you do not include the **nodeSelector**, the Policy applies to all nodes in the cluster.
- 3 This example uses a **hostname** node selector.
- 4 Name of the interface.

- 5 Optional: Human-readable description of the interface.
- 6 The type of interface. This example creates an Ethernet networking interface.
- 7 The requested state for the interface after creation.
- 8 Optional: If you do not use **dhcp**, you can either set a static IP or leave the interface without an IP address.
- 9 Enables **ipv4** in this example.

11.2.6.5. Example: Multiple interfaces in the same Policy

You can create multiple interfaces in the same Policy. These interfaces can reference each other, allowing you to build and deploy a network configuration by using a single Policy manifest.

The following example snippet creates a bond that is named **bond10** across two NICs and a Linux bridge that is named **br1** that connects to the bond.

```
...
  interfaces:
  - name: bond10
    description: Bonding eth2 and eth3 for Linux bridge
    type: bond
    state: up
    link-aggregation:
      slaves:
      - eth2
      - eth3
  - name: br1
    description: Linux bridge on bond
    type: linux-bridge
    state: up
    bridge:
      port:
      - name: bond10
...

```

11.2.7. Examples: IP management

The following example configuration snippets demonstrate different methods of IP management.

These examples use the **ethernet** interface type to simplify the example while showing the related context in the Policy configuration. These IP management examples can be used with the other interface types.

11.2.7.1. Static

The following snippet statically configures an IP address on the Ethernet interface:

```
...
  interfaces:
  - name: eth1
    description: static IP on eth1
...

```

```

type: ethernet
state: up
ipv4:
  address:
    - ip: 192.168.122.250 1
      prefix-length: 24
  enabled: true
...

```

- 1** Replace this value with the static IP address for the interface.

11.2.7.2. No IP address

The following snippet ensures that the interface has no IP address:

```

...
interfaces:
  - name: eth1
    description: No IP on eth1
    type: ethernet
    state: up
    ipv4:
      enabled: false
...

```

11.2.7.3. Dynamic host configuration

The following snippet configures an Ethernet interface that uses a dynamic IP address, gateway address, and DNS:

```

...
interfaces:
  - name: eth1
    description: DHCP on eth1
    type: ethernet
    state: up
    ipv4:
      dhcp: true
      enabled: true
...

```

The following snippet configures an Ethernet interface that uses a dynamic IP address but does not use a dynamic gateway address or DNS:

```

...
interfaces:
  - name: eth1
    description: DHCP without gateway or DNS on eth1
    type: ethernet
    state: up
    ipv4:
      dhcp: true
      auto-gateway: false
...

```

```

auto-dns: false
enabled: true

```

```
...
```

11.2.7.4. DNS

The following snippet sets DNS configuration on the host.

```

...
interfaces:
  ...
dns-resolver:
  config:
    search:
      - example.com
      - example.org
    server:
      - 8.8.8.8
...

```

11.2.7.5. Static routing

The following snippet configures a static route and a static IP on interface **eth1**.

```

...
interfaces:
  - name: eth1
    description: Static routing on eth1
    type: ethernet
    state: up
    ipv4:
      address:
        - ip: 192.0.2.251 1
          prefix-length: 24
          enabled: true
    routes:
      config:
        - destination: 198.51.100.0/24
          metric: 150
          next-hop-address: 192.0.2.1 2
          next-hop-interface: eth1
          table-id: 254
...

```

1 The static IP address for the Ethernet interface.

2 Next hop address for the node traffic. This must be in the same subnet as the IP address set for the Ethernet interface.

11.3. TROUBLESHOOTING NODE NETWORK CONFIGURATION

If the node network configuration encounters an issue, the Policy is automatically rolled back and the Enactments report failure. This includes issues such as:

- The configuration fails to be applied on the host.
- The host loses connection to the default gateway.
- The host loses connection to the API server.

11.3.1. Troubleshooting an incorrect NodeNetworkConfigurationPolicy configuration

You can apply changes to the node network configuration across your entire cluster by applying a NodeNetworkConfigurationPolicy. If you apply an incorrect configuration, you can use the following example to troubleshoot and correct the failed network Policy.

In this example, a Linux bridge Policy is applied to an example cluster that has 3 master nodes and 3 worker nodes. The Policy fails to be applied because it references an incorrect interface. To find the error, investigate the available nmstate resources. You can then update the Policy with the correct configuration.

Procedure

1. Create a Policy and apply it to your cluster. The following example creates a simple bridge on the **ens01** interface:

```
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ens01-bridge-testfail
spec:
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with the wrong port
        type: linux-bridge
        state: up
        ipv4:
          dhcp: true
          enabled: true
        bridge:
          options:
            stp:
              enabled: false
        port:
          - name: ens01
```

```
$ oc apply -f ens01-bridge-testfail.yaml
```

Example output

```
nodenetworkconfigurationpolicy.nmstate.io/ens01-bridge-testfail created
```

2. Verify the status of the Policy by running the following command:

```
$ oc get nncp
```

The output shows that the Policy failed:

Example output

```
NAME                STATUS
ens01-bridge-testfail FailedToConfigure
```

However the Policy status alone does not indicate if it failed on all nodes or a subset of nodes.

- List the Enactments to see if the Policy was successful on any of the nodes. If the Policy failed for only a subset it suggests the problem is with specific node configuration; if the Policy failed on all nodes it suggest the problem is with the Policy.

```
$ oc get nnce
```

The output shows that the Policy failed on all nodes:

Example output

```
NAME                STATUS
master-1.ens01-bridge-testfail FailedToConfigure
master-2.ens01-bridge-testfail FailedToConfigure
master-3.ens01-bridge-testfail FailedToConfigure
worker-1.ens01-bridge-testfail FailedToConfigure
worker-2.ens01-bridge-testfail FailedToConfigure
worker-3.ens01-bridge-testfail FailedToConfigure
```

- View one of the failed Enactments and look at the traceback. The following command uses the output tool **jsonpath** to filter the output:

```
$ oc get nnce worker-1.ens01-bridge-testfail -o jsonpath='{.status.conditions[?(@.type=="Failing")].message}'
```

This command returns a large traceback that has been edited for brevity:

Example output

```
error reconciling NodeNetworkConfigurationPolicy at desired state apply: , failed to execute
nmstatectl set --no-commit --timeout 480: 'exit status 1' "
...
libnmstate.error.NmstateVerificationError:
desired
=====
---
name: br1
type: linux-bridge
state: up
bridge:
  options:
    group-forward-mask: 0
    mac-ageing-time: 300
    multicast-snooping: true
```



```

stp:
  enabled: false
  forward-delay: 15
  hello-time: 2
  max-age: 20
  priority: 32768
port:
- name: ens01
description: Linux bridge with the wrong port
ipv4:
  address: []
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  dhcp: true
  enabled: true
ipv6:
  enabled: false
mac-address: 01-23-45-67-89-AB
mtu: 1500

current
=====
---
name: br1
type: linux-bridge
state: up
bridge:
  options:
    group-forward-mask: 0
    mac-ageing-time: 300
    multicast-snooping: true
  stp:
    enabled: false
    forward-delay: 15
    hello-time: 2
    max-age: 20
    priority: 32768
  port: []
description: Linux bridge with the wrong port
ipv4:
  address: []
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  dhcp: true
  enabled: true
ipv6:
  enabled: false
mac-address: 01-23-45-67-89-AB
mtu: 1500

difference
=====
--- desired
+++ current

```

```

@@ -13,8 +13,7 @@
    hello-time: 2
    max-age: 20
    priority: 32768
  - port:
  - - name: ens01
+ port: []
  description: Linux bridge with the wrong port
  ipv4:
    address: []
  line 651, in _assert_interfaces_equal\n
  current_state.interfaces[ifname],\nlibnmstate.error.NmstateVerificationError:

```

The **NmstateVerificationError** lists the **desired** Policy configuration, the **current** configuration of the Policy on the node, and the **difference** highlighting the parameters that do not match. In this example, the **port** is included in the **difference**, which suggests that the problem is the port configuration in the Policy.

- To ensure that the Policy is configured properly, view the network configuration for one or all of the nodes by requesting the **NodeNetworkState**. The following command returns the network configuration for the **master-1** node:

```
$ oc get nns master-1 -o yaml
```

The output shows that the interface name on the nodes is **ens1** but the failed Policy incorrectly uses **ens01**:

Example output

```

- ipv4:
...
  name: ens1
  state: up
  type: ethernet

```

- Correct the error by editing the existing Policy:

```
$ oc edit nncp ens01-bridge-testfail
```

```

...
  port:
    - name: ens1

```

Save the Policy to apply the correction.

- Check the status of the Policy to ensure it updated successfully:

```
$ oc get nncp
```

Example output

```

NAME                STATUS
ens01-bridge-testfail SuccessfullyConfigured

```

The updated Policy is successfully configured on all nodes in the cluster.

CHAPTER 12. LOGGING, EVENTS, AND MONITORING

12.1. VIEWING VIRTUAL MACHINE LOGS

12.1.1. Understanding virtual machine logs

Logs are collected for OpenShift Container Platform Builds, Deployments, and pods. In OpenShift Virtualization, virtual machine logs can be retrieved from the virtual machine launcher Pod in either the web console or the CLI.

The **-f** option follows the log output in real time, which is useful for monitoring progress and error checking.

If the launcher Pod is failing to start, use the **--previous** option to see the logs of the last attempt.



WARNING

ErrImagePull and **ImagePullBackOff** errors can be caused by an incorrect Deployment configuration or problems with the images that are referenced.

12.1.2. Viewing virtual machine logs in the CLI

Get virtual machine logs from the virtual machine launcher pod.

Procedure

- Use the following command:

```
$ oc logs <virt-launcher-name>
```

12.1.3. Viewing virtual machine logs in the web console

Get virtual machine logs from the associated virtual machine launcher pod.

Procedure

1. In the OpenShift Virtualization console, click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. In the **Details** tab, click the **virt-launcher-<name>** Pod in the **Pod** section.
5. Click **Logs**.

12.2. VIEWING EVENTS

12.2.1. Understanding virtual machine events

OpenShift Container Platform events are records of important life-cycle information in a namespace and are useful for monitoring and troubleshooting resource scheduling, creation, and deletion issues.

OpenShift Virtualization adds events for virtual machines and virtual machine instances. These can be viewed from either the web console or the CLI.

See also: [Viewing system event information in an OpenShift Container Platform cluster](#) .

12.2.2. Viewing the events for a virtual machine in the web console

You can view the stream events for a running a virtual machine from the **Virtual Machine Overview** panel of the web console.

The **⏸** button pauses the events stream.

The **▶** button continues a paused events stream.

Procedure

1. Click **Workloads** → **Virtualization** from the side menu.
2. Click the **Virtual Machines** tab.
3. Select a virtual machine to open the **Virtual Machine Overview** screen.
4. Click **Events** to view all events for the virtual machine.

12.2.3. Viewing namespace events in the CLI

Use the OpenShift Container Platform client to get the events for a namespace.

Procedure

- In the namespace, use the **oc get** command:

```
┆ $ oc get events
```

12.2.4. Viewing resource events in the CLI

Events are included in the resource description, which you can get using the OpenShift Container Platform client.

Procedure

- In the namespace, use the **oc describe** command. The following example shows how to get the events for a virtual machine, a virtual machine instance, and the virt-launcher Pod for a virtual machine:

```
┆ $ oc describe vm <vm>
```

```
┆ $ oc describe vmi <vmi>
```

```
$ oc describe pod virt-launcher-<name>
```

12.3. DIAGNOSING DATAVOLUMES USING EVENTS AND CONDITIONS

Use the **oc describe** command to analyze and help resolve issues with DataVolumes.

12.3.1. About conditions and events

Diagnose DataVolume 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 DataVolume.

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.

12.3.2. Analyzing DataVolumes using conditions and events

By inspecting the **Conditions** and **Events** sections generated by the **describe** command, you determine the state of the DataVolume in relation to PersistentVolumeClaims (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 DataVolume, 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 DataVolume.

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 DataVolume:

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 DataVolume is ready to be used, as in the following example. If the DataVolume is not ready to be used, the **Status** is **False**:

Example output

```

---
- name: vm1
  status:
    Conditions:
      - Last Heart Beat Time: 2020-07-15T04:31:39Z
        Last Transition Time: 2020-07-15T04:31:39Z
        Status: True
        Type: Ready

```

12.4. VIEWING INFORMATION ABOUT VIRTUAL MACHINE WORKLOADS

You can view high-level information about your virtual machines by using the **Virtual Machines** dashboard in the OpenShift Container Platform web console.

12.4.1. About the Virtual Machines dashboard

Access virtual machines from the OpenShift Container Platform web console by navigating to the **Workloads** → **Virtualization** page. The **Workloads** → **Virtualization** page contains two tabs: * **Virtual Machines** * **Virtual Machine Templates**

The following cards describe each virtual machine:

- **Details** provides identifying information about the virtual machine, including:
 - Name
 - Namespace
 - Date of creation
 - Node name
 - IP address
- **Inventory** lists the virtual machine's resources, including:
 - Network interface controllers (NICs)
 - Disks
- **Status** includes:
 - The current status of the virtual machine
 - A note indicating whether or not the QEMU guest agent is installed on the virtual machine
- **Utilization** includes charts that display usage data for:
 - CPU
 - Memory
 - Filesystem
 - Network transfer

**NOTE**

Use the drop-down list to choose a duration for the utilization data. The available options are **1 Hour**, **6 Hours**, and **24 Hours**.

- **Events** lists messages about virtual machine activity over the past hour. To view additional events, click **View all**.

12.5. MONITORING VIRTUAL MACHINE HEALTH

Use this procedure to create liveness and readiness probes to monitor virtual machine health.

12.5.1. About liveness and readiness probes

When a `VirtualMachineInstance` (VMI) fails, *liveness probes* stop the VMI. Controllers such as `VirtualMachine` then spawn other VMIs, restoring virtual machine responsiveness.

Readiness probes tell services and endpoints that the `VirtualMachineInstance` is ready to receive traffic from services. If readiness probes fail, the `VirtualMachineInstance` is removed from applicable endpoints until the probe recovers.

12.5.2. Define an HTTP liveness probe

This procedure provides an example configuration file for defining HTTP liveness probes.

Procedure

1. Customize a YAML configuration file to create an HTTP liveness probe, using the following code block as an example. In this example:
 - You configure a probe using **`spec.livenessProbe.httpGet`**, which queries port **1500** of the virtual machine instance, after an initial delay of **120** seconds.
 - The virtual machine instance installs and runs a minimal HTTP server on port **1500** using **`cloud-init`**.

**NOTE**

The **`timeoutSeconds`** value must be lower than the **`periodSeconds`** value. The **`timeoutSeconds`** default value is **1**. The **`periodSeconds`** default value is **10**.

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstance
metadata:
  labels:
    special: vmi-fedora
  name: vmi-fedora
spec:
  domain:
  devices:
    disks:
      - disk:
          bus: virtio
            name: containerdisk
```

```

- disk:
  bus: virtio
  name: cloudinitdisk
resources:
  requests:
    memory: 1024M
livenessProbe:
  initialDelaySeconds: 120
  periodSeconds: 20
  httpGet:
    port: 1500
  timeoutSeconds: 10
  terminationGracePeriodSeconds: 0
volumes:
- name: containerdisk
  registryDisk:
    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

```

2. Create the VirtualMachineInstance by running the following command:

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

12.5.3. Define a TCP liveness probe

This procedure provides an example configuration file for defining TCP liveness probes.

Procedure

1. Customize a YAML configuration file to create an TCP liveness probe, using this code block as an example. In this example:
 - You configure a probe using **spec.livenessProbe.tcpSocket**, which queries port **1500** of the virtual machine instance, after an initial delay of **120** seconds.
 - The virtual machine instance installs and runs a minimal HTTP server on port **1500** using **cloud-init**.



NOTE

The **timeoutSeconds** value must be lower than the **periodSeconds** value. The **timeoutSeconds** default value is **1**. The **periodSeconds** default value is **10**.

```
apiVersion: kubevirt.io/v1alpha3
```

```

kind: VirtualMachineInstance
metadata:
  labels:
    special: vmi-fedora
  name: vmi-fedora
spec:
  domain:
    devices:
      disks:
        - disk:
            bus: virtio
            name: containerdisk
        - disk:
            bus: virtio
            name: cloudinitdisk
      resources:
        requests:
          memory: 1024M
    livenessProbe:
      initialDelaySeconds: 120
      periodSeconds: 20
      tcpSocket:
        port: 1500
      timeoutSeconds: 10
    terminationGracePeriodSeconds: 0
  volumes:
    - name: containerdisk
      registryDisk:
        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

```

2. Create the VirtualMachineInstance by running the following command:

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

12.5.4. Define a readiness probe

This procedure provides an example configuration file for defining readiness probes.

Procedure

1. Customize a YAML configuration file to create a readiness probe. Readiness probes are configured in a similar manner to liveness probes. However, note the following differences in this example:

- Readiness probes are saved using a different spec name. For example, you create a readiness probe as **spec.readinessProbe** instead of as **spec.livenessProbe.<type-of-probe>**.
- When creating a readiness probe, you optionally set a **failureThreshold** and a **successThreshold** to switch between **ready** and **non-ready** states, should the probe succeed or fail multiple times.



NOTE

The **timeoutSeconds** value must be lower than the **periodSeconds** value. The **timeoutSeconds** default value is **1**. The **periodSeconds** default value is **10**.

```

apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstance
metadata:
  labels:
    special: vmi-fedora
  name: vmi-fedora
spec:
  domain:
    devices:
      disks:
        - disk:
            bus: virtio
            name: containerdisk
        - disk:
            bus: virtio
            name: cloudinitdisk
      resources:
        requests:
          memory: 1024M
  readinessProbe:
    httpGet:
      port: 1500
    initialDelaySeconds: 120
    periodSeconds: 20
    timeoutSeconds: 10
    failureThreshold: 3
    successThreshold: 3
  terminationGracePeriodSeconds: 0
  volumes:
    - name: containerdisk
      registryDisk:
        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: clouddisk
```

2. Create the VirtualMachineInstance by running the following command:

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

12.6. USING THE OPENSIFT CONTAINER PLATFORM DASHBOARD TO GET CLUSTER INFORMATION

Access the OpenShift Container Platform dashboard, which captures high-level information about the cluster, by clicking **Home > Dashboards > Overview** from the OpenShift Container Platform web console.

The OpenShift Container Platform dashboard provides various cluster information, captured in individual dashboard *cards*.

12.6.1. About the OpenShift Container Platform dashboards page

The OpenShift Container Platform dashboard consists of the following cards:

- **Details** provides a brief overview of informational cluster details. Status include **ok**, **error**, **warning**, **in progress**, and **unknown**. Resources can add custom status names.
 - Cluster ID
 - Provider
 - Version
- **Cluster Inventory** details number of resources and associated statuses. It is helpful when intervention is required to resolve problems, including information about:
 - Number of nodes
 - Number of pods
 - Persistent storage volume claims
 - Virtual machines (available if OpenShift Virtualization is installed)
 - Bare metal hosts in the cluster, listed according to their state (only available in **metal3** environment).
- **Cluster Health** summarizes the current health of the cluster as a whole, including relevant alerts and descriptions. If OpenShift Virtualization is installed, the overall health of OpenShift Virtualization is diagnosed as well. If more than one subsystem is present, click **See All** to view the status of each subsystem.
- **Cluster Capacity** charts help administrators understand when additional resources are required in the cluster. The charts contain an inner ring that displays current consumption, while an outer ring displays thresholds configured for the resource, including information about:

- CPU time
 - Memory allocation
 - Storage consumed
 - Network resources consumed
- **Cluster Utilization** shows the capacity of various resources over a specified period of time, to help administrators understand the scale and frequency of high resource consumption.
 - **Events** lists messages related to recent activity in the cluster, such as pod creation or virtual machine migration to another host.
 - **Top Consumers** helps administrators understand how cluster resources are consumed. Click on a resource to jump to a detailed page listing pods and nodes that consume the largest amount of the specified cluster resource (CPU, memory, or storage).

12.7. OPENSIFT CONTAINER PLATFORM CLUSTER MONITORING, LOGGING, AND TELEMETRY

OpenShift Container Platform provides various resources for monitoring at the cluster level.

12.7.1. About OpenShift Container Platform cluster monitoring

OpenShift Container Platform includes a pre-configured, pre-installed, and self-updating monitoring stack that is based on the [Prometheus](#) open source project and its wider eco-system. It provides monitoring of cluster components and includes a set of alerts to immediately notify the cluster administrator about any occurring problems and a set of [Grafana](#) dashboards. The cluster monitoring stack is only supported for monitoring OpenShift Container Platform clusters.



IMPORTANT

To ensure compatibility with future OpenShift Container Platform updates, configuring only the specified monitoring stack options is supported.

12.7.2. About cluster logging components

The cluster logging components include a collector deployed to each node in the OpenShift Container Platform cluster that collects all node and container logs and writes them to a log store. You can use a centralized web UI to create rich visualizations and dashboards with the aggregated data.

The major components of cluster logging are:

- **collection** - This is the component that collects logs from the cluster, formats them, and forwards them to the log store. The current implementation is Fluentd.
- **log store** - This is where the logs are stored. The default implementation is Elasticsearch. You can use the default Elasticsearch log store or forward logs to external log stores. The default log store is optimized and tested for short-term storage.
- **visualization** - This is the UI component you can use to view logs, graphs, charts, and so forth. The current implementation is Kibana.

For more information on cluster logging, see the [OpenShift Container Platform cluster logging](#) documentation.

12.7.3. About Telemetry

Telemetry sends a carefully chosen subset of the cluster monitoring metrics to Red Hat. The Telemeter Client fetches the metrics values every four minutes and thirty seconds and uploads the data to Red Hat. These metrics are described in this document.

This stream of data is used by Red Hat to monitor the clusters in real-time and to react as necessary to problems that impact our customers. It also allows Red Hat to roll out OpenShift Container Platform upgrades to customers to minimize service impact and continuously improve the upgrade experience.

This debugging information is available to Red Hat Support and Engineering teams with the same restrictions as accessing data reported through support cases. All connected cluster information is used by Red Hat to help make OpenShift Container Platform better and more intuitive to use.

12.7.3.1. Information collected by Telemetry

The following information is collected by Telemetry:

- The unique random identifier that is generated during an installation
- Version information, including the OpenShift Container Platform cluster version and installed update details that are used to determine update version availability
- Update information, including the number of updates available per cluster, the channel and image repository used for an update, update progress information, and the number of errors that occur in an update
- The name of the provider platform that OpenShift Container Platform is deployed on and the data center location
- Sizing information about clusters, machine types, and machines, including the number of CPU cores and the amount of RAM used for each
- The number of running virtual machine instances in a cluster
- The number of etcd members and the number of objects stored in the etcd cluster
- The OpenShift Container Platform framework components installed in a cluster and their condition and status
- Usage information about components, features, and extensions
- Usage details about Technology Previews and unsupported configurations
- Information about degraded software
- Information about nodes that are marked as **NotReady**
- Events for all namespaces listed as "related objects" for a degraded Operator
- Configuration details that help Red Hat Support to provide beneficial support for customers. This includes node configuration at the cloud infrastructure level, host names, IP addresses, Kubernetes pod names, namespaces, and services.

- Information about the validity of certificates

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.

12.7.4. CLI troubleshooting and debugging commands

For a list of the **oc** client troubleshooting and debugging commands, see the [OpenShift Container Platform CLI tools](#) documentation.

12.8. COLLECTING OPENSIFT VIRTUALIZATION DATA FOR RED HAT SUPPORT

When opening a support case, it is helpful to provide debugging information about your cluster to Red Hat Support.

The **must-gather** tool enables you to collect diagnostic information about your OpenShift Container Platform cluster, including virtual machines and other data related to OpenShift Virtualization.

For prompt support, supply diagnostic information for both OpenShift Container Platform and OpenShift Virtualization.

12.8.1. About the must-gather tool

The **oc adm must-gather** CLI command collects the information from your cluster that is most likely needed for debugging issues, such as:

- Resource definitions
- Audit logs
- Service logs

You can specify one or more images when you run the command by including the **--image** argument. When you specify an image, the tool collects data related to that feature or product.

When you run **oc adm must-gather**, a new pod is created on the cluster. The data is collected on that pod and saved in a new directory that starts with **must-gather.local**. This directory is created in the current working directory.

12.8.2. About collecting OpenShift Virtualization data

You can use the **oc adm must-gather** CLI command to collect information about your cluster, including features and objects associated with OpenShift Virtualization:

- The Hyperconverged Cluster Operator namespaces (and child objects)
- All namespaces (and their child objects) that belong to any OpenShift Virtualization resources
- All OpenShift Virtualization Custom Resource Definitions (CRDs)
- All namespaces that contain virtual machines

- All virtual machine definitions

To collect OpenShift Virtualization data with **must-gather**, you must specify the OpenShift Virtualization image: **--image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v2.4.9**.

12.8.3. Gathering data about specific features

You can gather debugging information about specific features by using the **oc adm must-gather** CLI command with the **--image** or **--image-stream** argument. The **must-gather** tool supports multiple images, so you can gather data about more than one feature by running a single command.



NOTE

To collect the default **must-gather** data in addition to specific feature data, add the **--image-stream=openshift/must-gather** argument.

Prerequisites

- Access to the cluster as a user with the **cluster-admin** role.
- The OpenShift Container Platform CLI (**oc**) installed.

Procedure

1. Navigate to the directory where you want to store the **must-gather** data.
2. Run the **oc adm must-gather** command with one or more **--image** or **--image-stream** arguments. For example, the following command gathers both the default cluster data and information specific to OpenShift Virtualization:

```
$ oc adm must-gather \
  --image-stream=openshift/must-gather \ 1
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v2.4.9 2
```

1 The default OpenShift Container Platform **must-gather** image

2 The **must-gather** image for OpenShift Virtualization

You can use the **must-gather** tool with additional arguments to gather data that is specifically related to cluster logging and the Cluster Logging Operator in your cluster. For cluster logging, run the following command:

```
$ oc adm must-gather --image=$(oc -n openshift-logging get deployment.apps/cluster-logging-operator \
  -o jsonpath='{.spec.template.spec.containers[?(@.name == "cluster-logging-operator")].image}')
```

Example 12.1. Example **must-gather** output for cluster logging

```
├── cluster-logging
│   ├── clo
│   └── cluster-logging-operator-74dd5994f-6ttgt
```

```

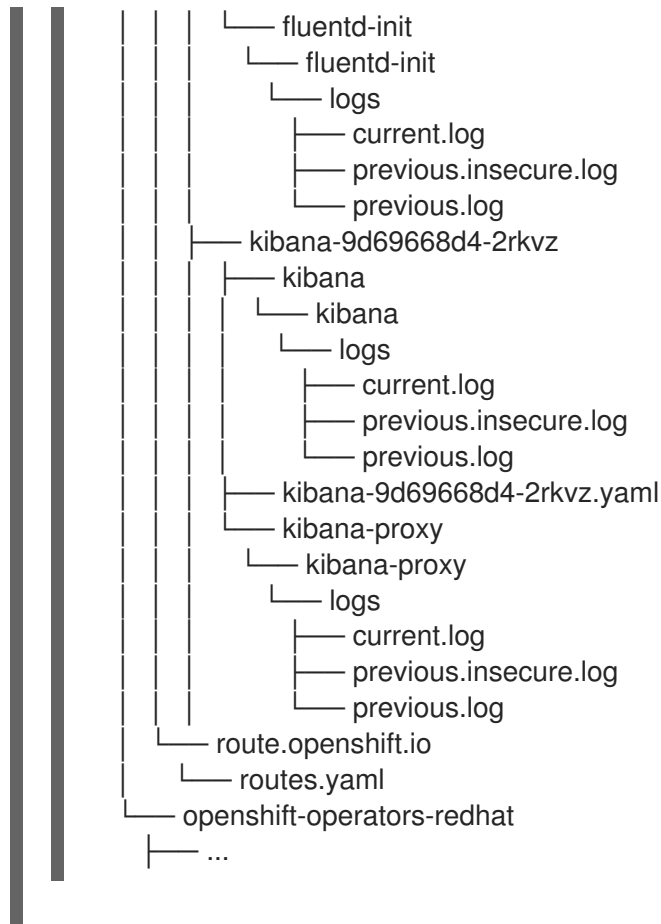
├── clusterlogforwarder_cr
├── cr
├── csv
├── deployment
├── logforwarding_cr
├── collector
├── fluentd-2tr64
├── curator
├── curator-1596028500-zkz4s
├── eo
├── csv
├── deployment
├── elasticsearch-operator-7dc7d97b9d-jb4r4
├── es
├── cluster-elasticsearch
├── aliases
├── health
├── indices
├── latest_documents.json
├── nodes
├── nodes_stats.json
├── thread_pool
├── cr
├── elasticsearch-cdm-lp8l38m0-1-794d6dd989-4jxms
├── logs
├── elasticsearch-cdm-lp8l38m0-1-794d6dd989-4jxms
├── install
├── co_logs
├── install_plan
├── olmo_logs
├── subscription
├── kibana
├── cr
├── kibana-9d69668d4-2rkvz
├── cluster-scoped-resources
├── core
├── nodes
├── ip-10-0-146-180.eu-west-1.compute.internal.yaml
├── persistentvolumes
├── pvc-0a8d65d9-54aa-4c44-9ecc-33d9381e41c1.yaml
├── event-filter.html
├── gather-debug.log
├── namespaces
├── openshift-logging
├── apps
├── daemonsets.yaml
├── deployments.yaml
├── replicasetsets.yaml
├── statefulsets.yaml
├── batch
├── cronjobs.yaml
├── jobs.yaml
├── core
├── configmaps.yaml
├── endpoints.yaml
├── events

```

```

├── curator-1596021300-wn2ks.162634ebf0055a94.yaml
├── curator.162638330681bee2.yaml
├── elasticsearch-delete-app-1596020400-gm6nl.1626341a296c16a1.yaml
├── elasticsearch-delete-audit-1596020400-9l9n4.1626341a2af81bbd.yaml
├── elasticsearch-delete-infra-1596020400-v98tk.1626341a2d821069.yaml
├── elasticsearch-rollover-app-1596020400-cc5vc.1626341a3019b238.yaml
├── elasticsearch-rollover-audit-1596020400-s8d5s.1626341a31f7b315.yaml
├── elasticsearch-rollover-infra-1596020400-7mgv8.1626341a35ea59ed.yaml
├── events.yaml
├── persistentvolumeclaims.yaml
├── pods.yaml
├── replicationcontrollers.yaml
├── secrets.yaml
├── services.yaml
├── openshift-logging.yaml
├── pods
│   ├── cluster-logging-operator-74dd5994f-6ttgt
│   │   ├── cluster-logging-operator
│   │   │   └── cluster-logging-operator
│   │   │       └── logs
│   │   │           ├── current.log
│   │   │           ├── previous.insecure.log
│   │   │           └── previous.log
│   │   └── cluster-logging-operator-74dd5994f-6ttgt.yaml
│   └── cluster-logging-operator-registry-6df49d7d4-mxxff
│       ├── cluster-logging-operator-registry
│       │   ├── cluster-logging-operator-registry
│       │   │   └── logs
│       │   │       ├── current.log
│       │   │       ├── previous.insecure.log
│       │   │       └── previous.log
│       └── cluster-logging-operator-registry-6df49d7d4-mxxff.yaml
├── mutate-csv-and-generate-sqlite-db
│   └── mutate-csv-and-generate-sqlite-db
│       └── logs
│           ├── current.log
│           ├── previous.insecure.log
│           └── previous.log
├── curator-1596028500-zkz4s
├── elasticsearch-cdm-lp8l38m0-1-794d6dd989-4jxms
├── elasticsearch-delete-app-1596030300-bpgcx
│   ├── elasticsearch-delete-app-1596030300-bpgcx.yaml
│   ├── indexmanagement
│   │   └── indexmanagement
│   │       └── logs
│   │           ├── current.log
│   │           ├── previous.insecure.log
│   │           └── previous.log
├── fluentd-2tr64
│   ├── fluentd
│   │   ├── fluentd
│   │   │   └── logs
│   │   │       ├── current.log
│   │   │       ├── previous.insecure.log
│   │   │       └── previous.log
│   └── fluentd-2tr64.yaml

```



3. Create a compressed file from the **must-gather** directory that was just created in your working directory. For example, on a computer that uses a Linux operating system, run the following command:

```
$ tar cvaf must-gather.tar.gz must-gather-local.5421342344627712289/ 1
```

- 1** Make sure to replace **must-gather-local.5421342344627712289/** with the actual directory name.

4. Attach the compressed file to your support case on the [Red Hat Customer Portal](#).