

OpenShift Dedicated 4

Virtualization

OpenShift Virtualization installation and usage.

Last Updated: 2026-11-05

OpenShift Dedicated 4 Virtualization

OpenShift Virtualization installation and usage.

Legal Notice

Copyright © Red Hat.

Except as otherwise noted below, the text of and illustrations in this documentation are licensed by Red Hat under the Creative Commons Attribution–Share Alike 3.0 Unported license . 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, the Red Hat logo, JBoss, Hibernate, and RHCE are trademarks or registered trademarks of Red Hat, LLC. or its subsidiaries in the United States and other countries.

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

XFS is a trademark or registered trademark of Hewlett Packard Enterprise Development LP or its subsidiaries in the United States and other countries.

The OpenStack ® Word Mark and OpenStack logo are trademarks or registered trademarks of the Linux Foundation, used under license.

All other trademarks are the property of their respective owners.

Abstract

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

Table of Contents

CHAPTER 1. ABOUT 13
1.1. ABOUT OPENSHIFT VIRTUALIZATION 13
1.1.1. OpenShift Virtualization capabilities 13
1.1.2. VMware vSphere to OpenShift Virtualization concept mapping 13
1.1.3. Supported cluster versions on Google Cloud 14
1.1.4. Volume and access modes for VM disks 15
1.1.5. Additional resources 15
1.2. SECURITY POLICIES 16
1.2.1. Workload security 16
1.2.2. TLS certificates 16
1.2.2.1. Automatic renewal schedules 16
1.2.3. Authorization 16
1.2.3.1. Default cluster roles for OpenShift Virtualization 17
1.2.3.2. RBAC roles for storage features in OpenShift Virtualization 17
1.2.3.2.1. Cluster-wide RBAC roles 17
1.2.3.2.2. Namespaced RBAC roles 20
1.2.3.3. Additional SCCs and permissions for the kubevirt-controller service account 21
1.2.3.3.1. Viewing the SCC and RBAC definitions for the kubevirt-controller 22
1.2.4. Additional resources 22
1.3. OPENSHIFT VIRTUALIZATION ARCHITECTURE 22
1.3.1. HyperConverged Operator (HCO) 23
1.3.2. Containerized Data Importer (CDI) Operator 24
1.3.3. Hostpath Provisioner (HPP) Operator 25
1.3.4. Scheduling, Scale, and Performance (SSP) Operator 26
1.3.5. OpenShift Virtualization Operator 26
1.3.6. Additional resources 27

CHAPTER 2. GETTING STARTED 28
2.1. GETTING STARTED WITH OPENSHIFT VIRTUALIZATION 28
2.1.1. Additional resources 28
2.2. USING THE CLI TOOLS 28
2.2.1. Install the virtctl binary 29
2.2.2. virtctl information commands 30
2.2.3. VM information commands 30
2.2.4. VM manifest creation commands 30
2.2.5. VM management commands 32
2.2.6. VM connection commands 32
2.2.7. VM export commands 33
2.2.8. Hot plug and hot unplug commands 35
2.2.9. Image upload commands 35
2.2.10. Deploy libguestfs by using virtctl 36
2.2.11. Libguestfs and virtctl guestfs commands 36
2.2.12. Additional resources 38

CHAPTER 3. INSTALLING 39
3.1. PREPARING YOUR CLUSTER FOR OPENSHIFT VIRTUALIZATION 39
3.1.1. OpenShift Virtualization on AWS bare metal 39
3.1.2. Additional resources 40
3.2. INSTALLING OPENSHIFT VIRTUALIZATION 40
3.2.1. Install the OpenShift Virtualization Operator by using the web console 40
3.2.2. Subscribe to the OpenShift Virtualization catalog by using the CLI 42

3.2.3. Deploy the OpenShift Virtualization Operator by using the CLI 44
3.2.4. Additional resources 45
3.3. UNINSTALLING OPENSHIFT VIRTUALIZATION 45
3.3.1. Prerequisites 45
3.3.2. Delete the HyperConverged custom resource 45
3.3.3. Deleting Operators from a cluster using the web console 45
3.3.4. Delete a namespace using the web console 46
3.3.5. Delete OpenShift Virtualization custom resource definitions 47
3.3.6. Uninstall OpenShift Virtualization by using the CLI 47
3.3.7. Additional resources 48

CHAPTER 4. POST-INSTALLATION CONFIGURATION 49
4.1. POSTINSTALLATION CONFIGURATION 49
4.1.1. Additional resources 49
4.2. SPECIFYING NODES FOR OPENSHIFT VIRTUALIZATION COMPONENTS 49
4.2.1. About node placement rules for OpenShift Virtualization components 49
4.2.2. Apply node placement rules 50
4.2.3. Node placement rule examples 50
4.2.3.1. Subscription object node placement rule examples 50
4.2.3.2. HyperConverged object node placement rule example 52
4.2.3.3. HostPathProvisioner object node placement rule example 54
4.2.4. Additional resources 55
4.3. POSTINSTALLATION NETWORK CONFIGURATION 55
4.3.1. Create a Linux bridge node network configuration policy 55
4.3.2. Create a Linux bridge NAD by using the web console 56
4.3.3. Configure a dedicated secondary network for live migration 58
4.3.4. Select a dedicated migration network by using the web console 59
4.3.5. Enable load balancer services by using the web console 59
4.3.6. Configure additional routes to the cdi-uploadproxy service 60
4.4. POSTINSTALLATION STORAGE CONFIGURATION 61
4.4.1. Configuring local storage by using the HPP 61
4.4.1.1. Create a storage class for the CSI driver with the storagePools stanza 61
4.4.2. Additional resources 62
4.5. VIRTUAL MACHINE RECOVERY FROM NODE FAILURES 62
4.5.1. Node health monitoring for protection of virtual machines 62
4.5.1.1. Configure VM run strategy 63
4.5.2. Configure node health checks for virtual machines 63
4.5.3. Node remediation strategies for OpenShift Virtualization 65
4.5.3.1. Self Node Remediation (SNR) configuration parameters 66
4.5.4. Capacity planning for VM failover 67
4.5.5. Additional resources 67
4.6. CONFIGURING CERTIFICATE ROTATION 68
4.6.1. Configure certificate rotation 68
4.6.2. Troubleshoot certificate rotation parameters 69
4.6.3. Additional resources 69

CHAPTER 5. UPDATING 70
5.1. UPDATING OPENSHIFT VIRTUALIZATION 70
5.1.1. OpenShift Virtualization update behavior 70
5.1.1.1. Recommended settings 70
5.1.1.2. What to expect 70
5.1.1.3. How updates work 70
5.1.1.4. Change update settings 71

5.1.1.5. Manual approval strategy 71
5.1.1.6. Approving a pending Operator update manually 71
5.1.2. Operator and resource cleanup 72
5.1.3. RHEL 9 compatibility 72
5.1.3.1. RHEL 9 machine type 72
5.1.4. Monitor update status 72
5.1.5. VM workload updates 73
5.1.5.1. Migration attempts and timeouts 74
5.1.5.2. Configure workload update methods 74
5.1.5.3. View outdated VM workloads 75
5.1.6. Early access releases 76
5.1.7. Additional resources 76

CHAPTER 6. CREATING VIRTUAL MACHINES 77
6.1. CREATE VIRTUAL MACHINES BY USING THE WEB CONSOLE 77
6.1.1. VM creation methods 77
6.1.2. Create a VM with custom configuration by using the web console 77
6.1.3. Create a VM from a template by using the web console 79
6.1.4. Clone an existing VM by using the web console 80
6.1.5. Virtual machine creation wizard reference 82
6.1.5.1. Boot source options 82
6.1.5.2. Instance type series 83
6.1.5.3. Customization settings 83
6.1.6. Additional resources 84
6.2. CREATING VIRTUAL MACHINES BY USING THE CLI 85
6.2.1. Create a VM from a manifest 85
6.2.2. Create a VM from a web-hosted image 87
6.2.3. Generalize a VM image 89
6.2.4. Create a Windows VM from an uploaded image 90
6.2.4.1. Generalize a Windows VM image 92
6.2.4.2. Specialize a Windows VM image 92
6.2.5. Upload a VM image by using the CLI 93
6.2.6. Prepare a container disk for VMs 94
6.2.7. Disable TLS for an insecure registry 95
6.2.8. Create a VM from a container disk by using the CLI 96
6.2.9. VM cloning methods 97
6.2.9.1. CSI volume cloning 97
6.2.9.2. Smart cloning 98
6.2.9.3. Host-assisted cloning 98
6.2.10. VM creation from a cloned PVC 98
6.2.10.1. Optimize clone performance at scale in OpenShift Data Foundation 99
6.2.10.2. Clone a PVC to a data volume 99
6.2.10.3. Create a VM from a cloned PVC data volume template 101
6.2.11. Supported custom video device types 102
6.2.12. Additional resources 103
6.3. INSTANCE TYPES 104
6.3.1. Instance types 104
6.3.1.1. Required attributes 104
6.3.1.2. Optional attributes 105
6.3.1.3. Controller revisions 106
6.3.2. Specify instance types and preferences with flags 106
6.3.3. Infer instance types from volumes 106
6.3.4. Set volume inference labels 107

6.3.5. Change the instance type for a VM 107
6.3.5.1. Change a VM instance type in the web console 108
6.3.5.2. Change a VM instance type from the CLI 108
6.3.6. Additional resources 109
6.4. MANAGING VM TEMPLATES 109
6.4.1. VM templates 110
6.4.2. Create a custom VM template in the web console 110
6.4.3. Enable dedicated resources for a VM template 110
6.4.4. Undeprecate a customized VM template in the web console 111
6.4.5. Additional resources 111
6.5. MANAGING BOOT SOURCES 112
6.5.1. About golden images 112
6.5.1.1. How do golden images work? 112
6.5.1.2. Red Hat implementation of golden images 112
6.5.2. About VM boot sources 112
6.5.3. Configure a boot source image namespace by using the web console 113
6.5.4. Configure a boot source image namespace by using the CLI 113
6.5.5. Additional resources 114
6.6. IMPORT A VIRTUAL MACHINE IMAGE USING THE DATAVOLUME API 114
6.6.1. Supported import sources 114
6.6.2. Import a VM image from a container registry 115
6.6.3. CPU architecture selection for DataVolume imports 116
6.6.4. Import a VM image from an HTTP or HTTPS source 117
6.6.5. Import a VM image from S3 object storage 119
6.6.6. DataVolume source fields 120
6.6.6.1. Registry source fields 121
6.6.6.2. HTTP source fields 121
6.6.6.3. S3 source fields 122
6.6.6.4. ImageIO source fields 122
6.6.6.5. VDDK source fields 122
6.6.6.6. Blank source 123
6.6.6.7. PVC source fields 123
6.6.6.8. VolumeSnapshot source fields 123
6.6.6.9. Supported content types 123
6.6.7. Data volume import troubleshooting 124
6.6.7.1. Data volume conditions 124
6.6.7.2. Diagnostic commands 124
6.6.7.3. Common failure scenarios 124
6.6.8. Additional resources 125

CHAPTER 7. MANAGING VMS 126
7.1. INSTALL THE QEMU GUEST AGENT 126
7.1.1. Install the QEMU guest agent on a Linux VM 126
7.1.2. Install the QEMU guest agent on a Windows VM 126
7.2. INSTALL VIRTIO DRIVERS ON WINDOWS VMS 127
7.2.1. Download the VirtIO drivers ISO from the web console 128
7.2.2. Attach a VirtIO container disk during Windows installation 128
7.2.3. Attach a VirtIO container disk to an existing Windows VM 129
7.2.4. Install VirtIO drivers from a container disk SATA CD 129
7.2.5. Install VirtIO drivers during Windows setup 130
7.2.6. Install VirtIO drivers from a SATA CD on an existing VM 131
7.2.7. Additional resources 131
7.3. UPDATE VIRTIO DRIVERS 132

7.3.1. Download the VirtIO drivers ISO from the web console 132
7.3.2. Update VirtIO drivers and the guest agent on a Windows VM 132
7.3.3. Enable automatic updates for Red Hat virtio-win drivers 133
7.3.4. Update VirtIO drivers on a Windows VM 133
7.3.5. Additional resources 134
7.4. CONNECT TO A VIRTUAL MACHINE CONSOLE 134
7.4.1. VNC console access requirements and limits 134
7.4.2. Connect to the VNC console by using the web console 135
7.4.3. Connect to the VNC console by using virtctl 135
7.4.4. Grant permission to generate VNC console tokens 136
7.4.5. Generate a temporary token for the VNC console 137
7.4.6. Serial console access requirements and limits 138
7.4.7. Connect to the serial console by using the web console 138
7.4.8. Connect to the serial console by using virtctl 139
7.4.9. Connect to the desktop viewer by using the web console 139
7.4.10. Additional resources 140
7.5. EDIT THE CONFIGURATION OF A VIRTUAL MACHINE 140
7.5.1. Change a VM instance type in the web console 140
7.5.2. Change VM memory without a restart 141
7.5.3. Change VM CPU sockets without a restart 142
7.5.3.1. Supported platforms and versions for CPU hot-plug 143
7.5.4. Edit a virtual machine by using the CLI 144
7.5.5. Add a disk to a virtual machine 144
7.5.5.1. Virtual machine storage fields 144
7.5.5.1.1. Advanced storage settings 145
7.5.6. Mount a Windows driver disk on a virtual machine 146
7.5.7. Add a secret, config map, or service account to a VM 146
7.5.8. Update multiple virtual machines 147
7.5.8.1. Bulk actions on virtual machines 148
7.5.9. Configure multiple I/O threads for fast storage access 149
7.5.10. Additional resources 150
7.6. EDIT THE BOOT ORDER OF A VIRTUAL MACHINE 150
7.6.1. Add items to a boot order list in the web console 150
7.6.2. Edit a boot order list in the web console 151
7.6.3. Edit a boot order list by using the CLI 151
7.6.4. Remove items from a boot order list in the web console 152
7.7. DELETE A VIRTUAL MACHINE 153
7.7.1. Delete a VM by using the web console 153
7.7.2. Delete a VM by using the CLI 153
7.8. EXPORT A VIRTUAL MACHINE 154
7.8.1. Create a VirtualMachineExport custom resource 154
7.8.2. Access exported VM manifests 157
7.8.3. Additional resources 159
7.9. MANAGE VIRTUAL MACHINE INSTANCES 159
7.9.1. Virtual machine instances 160
7.9.2. List all VMIs by using the CLI 160
7.9.3. List standalone VMIs by using the web console 160
7.9.4. Search for standalone VMIs by using the web console 161
7.9.5. Edit a standalone VMI by using the web console 161
7.9.6. Delete a standalone VMI by using the CLI 162
7.9.7. Delete a standalone VMI by using the web console 162
7.9.8. Additional resources 162
7.10. CONTROL VIRTUAL MACHINE STATES 162

7.10.1. Configure RBAC permissions for managing VM states 162
7.10.2. Enable confirmations of VM actions 165
7.10.3. Start a VM 165
7.10.4. Stop a VM 165
7.10.5. Restart a VM 166
7.10.6. Reset a VM 167
7.10.7. Pause a VM 168
7.10.8. Unpause a VM 168
7.10.9. Control the state of multiple VMs 169
7.10.10. Additional resources 169
7.11. USE A VIRTUAL TRUSTED PLATFORM MODULE (VTPM) DEVICE 169
7.11.1. vTPM devices and state persistence 170
7.11.2. Add a vTPM device to a virtual machine 171
7.11.3. Additional resources 172
7.12. MANAGE VIRTUAL MACHINES WITH OPENSHIFT PIPELINES 172
7.12.1. Prerequisites 172
7.12.2. Supported virtual machine tasks 172
7.12.3. Windows EFI installer pipeline 173
7.12.3.1. Run the example pipelines in the web console 173
7.12.3.2. Run the example pipelines by using the CLI 174
7.12.4. Remove deprecated or unused resources 175
7.12.5. Additional resources 175
7.13. ADVANCED VIRTUAL MACHINE MANAGEMENT 175
7.13.1. Working with resource quotas for virtual machines 175
7.13.1.1. Set resource quota limits for VMs by using the CLI 175
7.13.1.2. Accurate VM resource measurement with AAQ 177
7.13.1.3. Enable the Application Aware Quota Operator 177
7.13.1.4. Create VM quotas by using the web console 178
7.13.1.5. Manage VM quotas by using the list view 180
7.13.1.6. Manage VM quota details by using the web console 181
7.13.1.7. Additional resources 182
7.13.2. Specifying nodes for virtual machines 182
7.13.2.1. Node placement rules for VMs 182
7.13.2.2. Node placement examples 183
7.13.2.2.1. VM node placement with nodeSelector 183
7.13.2.2.2. VM node placement with pod affinity and pod anti-affinity 184
7.13.2.2.3. VM node placement with node affinity 185
7.13.2.2.4. VM node placement with tolerations 186
7.13.2.3. Additional resources 186
7.13.3. Using UEFI mode for virtual machines 186
7.13.3.1. About UEFI mode for virtual machines 186
7.13.3.2. Boot virtual machines in UEFI mode 186
7.13.3.3. Configure VMs with persistent EFI 187
7.13.4. Configuring PXE booting for virtual machines 188
7.13.4.1. PXE boot a VM with a specified MAC address 188
7.13.4.2. OpenShift Virtualization networking glossary 191
7.13.5. Using huge pages with virtual machines 191
7.13.5.1. What huge pages do 191
7.13.5.2. Map VM memory more efficiently with huge pages 192
7.13.6. About high availability for virtual machines 193
7.13.6.1. Additional resources 193
7.13.7. Tune the control plane 193
7.13.7.1. Configure a highBurst profile 193

7.14. VM DISKS 194
7.14.1. Hot-plug virtual disks to running VMs 194
7.14.1.1. Hot plug and unplug disks by using the web console 194
7.14.1.2. Hot plug and hot unplug a disk by using the CLI 195
7.14.1.3. Additional resources 195
7.14.2. Expand virtual machine disks 196
7.14.2.1. Expand a VM disk PVC by using the web console 196
7.14.2.2. Expand a VM disk PVC by using the CLI 196
7.14.2.3. Expand VM storage by adding blank data volumes 197
7.14.2.4. Additional resources 198
7.15. CPU MODELS 198
7.15.1. Choose a CPU model 198
7.15.1.1. Host-model CPU for homogeneous clusters 198
7.15.1.1.1. Homogeneous clusters 198
7.15.1.1.2. Live migration with host-model 199
7.15.1.2. Named CPU models for heterogeneous clusters 199
7.15.1.2.1. Heterogeneous clusters 199
7.15.2. Configure CPU models 200
7.15.2.1. Configure the default CPU model 200
7.15.2.2. Configure a CPU model for a single VM 201
7.15.3. Schedule virtual machines 201
7.15.3.1. Set a VM to inherit the node CPU model 202
7.15.3.2. Set a VM to run only on nodes with a named CPU model 202
7.15.3.3. CPU feature policy attributes 202
7.15.3.4. Set a policy attribute and CPU feature 203
7.15.3.5. Schedule VMs with a custom scheduler 203

CHAPTER 8. NETWORKING 206
8.1. NETWORKING OVERVIEW 206
8.1.1. OpenShift Virtualization networking glossary 206
8.1.2. Overlay networks 206
8.1.3. VM physical network access methods 207
8.1.4. VM network interface management options 207
8.1.5. VM SSH access 208
8.1.6. Additional resources 209
8.2. CONNECTING A VIRTUAL MACHINE TO THE DEFAULT POD NETWORK 209
8.2.1. Configure masquerade mode by using the CLI 209
8.2.2. Jumbo frames support 210
8.3. CONNECTING A VIRTUAL MACHINE TO A PRIMARY USER-DEFINED NETWORK 210
8.3.1. Primary overlay network creation in the web console 211
8.3.1.1. Create a namespace for overlay networks by using the web console 211
8.3.1.2. Create a namespace-scoped primary overlay network in the web console 212
8.3.1.3. Creating a cluster-scoped network to connect pods directly to an external network 212
8.3.1.4. Create a cluster-scoped primary overlay network in the web console 213
8.3.2. Primary overlay network creation by using the CLI 214
8.3.2.1. Create a namespace for overlay networks by using the CLI 214
8.3.2.2. Create a namespace-scoped primary overlay network by using the CLI 215
8.3.2.3. Create a cluster-scoped primary overlay network by using the CLI 216
8.3.3. VM attachment to the primary overlay network 217
8.3.3.1. Create a VM with primary overlay access in the web console 217
8.3.3.2. Attach a VM to a primary overlay network 218
8.4. EXPOSING A VIRTUAL MACHINE BY USING A SERVICE 219
8.4.1. Service types for VM network access 219

8.4.2. Expose a VM as a service by using the CLI 220
8.5. CONNECTING A VIRTUAL MACHINE TO A SERVICE MESH 221
8.5.1. Connect a virtual machine to a service mesh 221
8.6. CONFIGURING AND VIEWING IP ADDRESSES 223
8.6.1. Configure a VM IP address by using the web console 223
8.6.2. Configure a VM IP address by using the CLI 224
8.6.3. View a VM IP address by using the web console 225
8.6.4. View a VM IP address by using the CLI 225
8.6.5. Additional resources 226

CHAPTER 9. STORAGE 227
9.1. STORAGE CONFIGURATION OVERVIEW 227
9.1.1. Storage 227
9.1.2. Containerized Data Importer 227
9.1.3. Data volumes 227
9.1.4. Boot source updates 228
9.1.5. Additional resources 228
9.2. CONFIGURING STORAGE PROFILES 228
9.2.1. Customizing the storage profile 228
9.2.1.1. Specifying a volume snapshot class by using the web console 230
9.2.1.2. Specifying a volume snapshot class by using the CLI 230
9.2.1.3. Viewing automatically created storage profiles 231
9.2.1.4. Setting a default cloning strategy by using a storage profile 232
9.3. MANAGING AUTOMATIC BOOT SOURCE UPDATES 233
9.3.1. Managing automatic updates for all system-defined boot sources 233
9.3.2. Configuring the default and virt-default storage classes 234
9.3.3. Configuring a storage class for boot source images 235
9.3.4. Enabling automatic updates for custom boot sources 236
9.3.5. Enabling volume snapshot boot sources 238
9.3.6. Disabling automatic updates for a single boot source 239
9.3.7. Increasing boot source disk image retention 240
9.3.8. Verifying the status of a boot source 240
9.3.9. Additional resources 242
9.4. RESERVING PVC SPACE FOR FILE SYSTEM OVERHEAD 242
9.4.1. Overriding the default file system overhead value 242
9.5. CONFIGURING LOCAL STORAGE BY USING THE HOSTPATH PROVISIONER 243
9.5.1. Creating a hostpath provisioner with a basic storage pool 244
9.5.1.1. About creating storage classes 244
9.5.1.2. Create a storage class for the CSI driver with the storagePools stanza 245
9.5.2. About storage pools created with PVC templates 246
9.5.2.1. Creating a storage pool with a PVC template 247
9.6. ENABLING USER PERMISSIONS TO CLONE DATA VOLUMES ACROSS NAMESPACES 248
9.6.1. Creating RBAC resources for cloning data volumes 248
9.7. CONFIGURING CDI TO OVERRIDE CPU AND MEMORY QUOTAS 249
9.7.1. About CPU and memory quotas in a namespace 250
9.7.2. Overriding CPU and memory defaults 250
9.7.3. Additional resources 250
9.8. PREPARING CDI SCRATCH SPACE 250
9.8.1. About scratch space 251
9.8.1.1. Manual provisioning 251
9.8.2. CDI operations that require scratch space 251
9.8.3. Defining a storage class 252
9.8.4. CDI supported operations matrix 252

9.8.5. Additional resources 253
9.9. USING PREALLOCATION FOR DATA VOLUMES 253
9.9.1. About preallocation 253
9.9.2. Enabling preallocation for a data volume 254
9.10. MANAGING DATA VOLUME ANNOTATIONS 254
9.10.1. Example: Data volume annotations 254

CHAPTER 10. LIVE MIGRATION 256
10.1. ABOUT LIVE MIGRATION 256
10.1.1. Live migration requirements 256
10.1.2. Live migration permissions 256
10.1.3. Preserve pre-4.19 live migration permissions during update 257
10.1.4. Grant live migration permissions 258
10.1.5. VM migration tuning 259
10.1.6. VM migration support for RHCOS 10.x 259
10.1.7. Additional resources 260
10.2. CONFIGURING LIVE MIGRATION 260
10.2.1. Configure live migration limits and timeouts 260
10.2.2. Configure live migration for heavy workloads 261
10.2.3. Live migration policies 262
10.2.4. Create a live migration policy by using the CLI 263
10.3. INITIATING AND CANCELING LIVE MIGRATION 264
10.3.1. Start live migration by using the web console 264
10.3.2. Start live migration by using the CLI 265
10.3.3. Cancel live migration by using the web console 266
10.3.4. Cancel live migration by using the CLI 266
10.3.5. Additional resources 267

CHAPTER 11. NODES 268
11.1. NODE MAINTENANCE MODE 268
11.1.1. Maintaining bare-metal nodes 268
11.1.2. About the Node Maintenance Operator 268
11.1.3. Additional resources 268
11.2. CONFIGURE EVICTION AND RUN STRATEGIES 268
11.2.1. Configure VM eviction strategy 269
11.2.2. Run strategies 269
11.2.3. Configure VM run strategy 270
11.2.4. Additional resources 271
11.3. MANAGING NODE LABELING FOR OBSOLETE CPU MODELS 271
11.3.1. Node labeling for obsolete CPU models 271
11.3.2. Configure obsolete CPU models 272
11.4. PREVENTING NODE RECONCILIATION 272
11.4.1. Skip node labeling reconciliation 272
11.4.2. Additional resources 273

CHAPTER 12. MONITORING 274
12.1. MONITORING OVERVIEW 274
12.2. PROMETHEUS QUERIES FOR VIRTUAL RESOURCES 274
12.2.1. Query metrics for all projects by using the OpenShift Dedicated web console 274
12.2.2. Query metrics for user-defined projects by using the OpenShift Dedicated web console 276
12.2.3. Virtualization metrics 278
12.2.3.1. Network metrics 278
12.2.3.2. Storage metrics 279
12.2.3.3. Guest memory swapping metrics 280

12.2.3.4. Monitoring AAQ operator metrics 280
12.2.3.5. VM label metrics 281
12.2.3.6. Live migration metrics 281
12.2.4. Node Memory Overcommit dashboard 282
12.2.4.1. Dashboard filters 282
12.2.4.2. Summary section 282
12.2.4.3. Cluster section 283
12.2.4.4. Nodes section 284
12.2.4.5. System Reserved section 285
12.2.4.6. Workloads section 286
12.2.4.7. Interpreting the dashboard 286
12.2.5. Additional resources 287
12.3. EXPOSING CUSTOM METRICS FOR VIRTUAL MACHINES 287
12.3.1. Configure the node exporter service 287
12.3.2. Configure a virtual machine with the node exporter service 288
12.3.3. Create a custom monitoring label for virtual machines 289
12.3.3.1. Query the node-exporter service for metrics 290
12.3.4. Create a ServiceMonitor resource for the node exporter service 291
12.3.4.1. Access the node exporter service outside the cluster 292
12.3.5. Additional resources 293
12.4. VIRTUAL MACHINE HEALTH CHECKS 293
12.4.1. Readiness and liveness probes for VM health 293
12.4.1.1. Define an HTTP readiness probe 294
12.4.1.2. Define a TCP readiness probe 295
12.4.1.3. Define an HTTP liveness probe 296
12.4.2. Watchdog devices and recovery actions 297
12.4.2.1. Configure a watchdog device for a virtual machine 298
12.4.2.2. Install the watchdog agent on the guest 299
12.5. OPENSHIFT VIRTUALIZATION RUNBOOKS 300
12.5.1. Runbooks for deprecated alerts 303

CHAPTER 13. SUPPORT 305
13.1. SUPPORT OVERVIEW 305
13.1.1. Opening a support case 305
13.1.1.1. Collecting data for Red Hat Support 305
13.1.1.2. Submitting a support case 305
13.1.2. Report a bug in OpenShift Virtualization 305
13.1.3. Report a OpenShift Virtualization documentation issue 306
13.1.4. Web console monitoring 306
13.1.5. Additional resources 307
13.2. COLLECTING DATA FOR RED HAT SUPPORT 307
13.2.1. Collecting data about your environment 307
13.2.2. Collecting data about virtual machines 307
13.2.3. Generate a VM memory dump 308
13.2.4. Additional resources 309
13.3. TROUBLESHOOTING 309
13.3.1. View VM and resource events 309
13.3.2. OpenShift Virtualization pod logs 310
13.3.2.1. Configure OpenShift Virtualization pod log verbosity 310
13.3.2.2. View virt-launcher pod logs with the web console 311
13.3.2.3. View OpenShift Virtualization pod logs with the CLI 311
13.3.3. Guest system logs 312
13.3.3.1. Enable default access to guest system logs with the web console 312

13.3.3.2. Enable default access to guest system logs with the CLI 313
13.3.3.3. Set per-VM guest system log access with the web console 313
13.3.3.4. Set per-VM guest system log access with the CLI 313
13.3.3.5. View guest system logs with the web console 314
13.3.3.6. View guest system logs with the CLI 315
13.3.4. Log aggregation with Loki 315
13.3.4.1. View aggregated OpenShift Virtualization logs with Loki 315
13.3.4.2. OpenShift Virtualization LogQL queries 315
13.3.4.3. Common error messages 317
13.3.5. Data volume troubleshooting 317
13.3.5.1. Data volume conditions and events 318
13.3.5.2. Data volume condition and event analysis 318
13.3.6. Additional resources 320

CHAPTER 14. BACKUP AND RESTORE 321
14.1. BACKUP AND RESTORE BY USING VM SNAPSHOTS 321
14.1.1. VM snapshot concepts and components 321
14.1.1.1. VM snapshot controller and custom resources 322
14.1.2. Application-consistent snapshots and backups 322
14.1.3. Create a snapshot by using the web console 323
14.1.4. Create a snapshot by using the CLI 323
14.1.5. Verify online snapshots by using snapshot indications 326
14.1.6. Restore a VM from a snapshot by using the web console 327
14.1.7. Restore a VM from a snapshot by using the CLI 328
14.1.8. Delete a snapshot by using the web console 331
14.1.9. Delete a snapshot by using the CLI 331
14.2. BACKING UP AND RESTORING VIRTUAL MACHINES 332
14.2.1. Install and configure OADP with OpenShift Virtualization 332
14.2.2. Install the Data Protection Application 333
14.2.2.1. DataProtectionApplication CR examples by storage provider 335

CHAPTER 1. ABOUT

1.1. ABOUT OPENSHIFT VIRTUALIZATION

OpenShift Virtualization provides a comprehensive virtualization solution that allows you to run and manage virtual machine workloads alongside container workloads in your OpenShift Dedicated cluster.

1.1.1. OpenShift Virtualization capabilities

OpenShift Virtualization provides scalable, enterprise-grade virtualization functionality for your cluster. You can use it to manage virtual machines (VMs) exclusively or alongside container workloads.

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

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

OpenShift Virtualization on OpenShift Dedicated includes guest subscriptions for Red Hat Enterprise Linux (RHEL) based on the number of vCPUs on the host:

- Hosts with 96 or more vCPUs: Unlimited RHEL guest subscriptions are included.
- Hosts with fewer than 96 vCPUs: You can run RHEL guests with a guest vCPU to host vCPU ratio of up to 8:1.
  For example, a host with 64 vCPUs can run up to 512 RHEL guest vCPUs (64 host vCPUs x 8 = 512 guest vCPUs).

OVN-Kubernetes is the default network provider for OpenShift Virtualization on OpenShift Dedicated. For more information, see "OVN-Kubernetes" in the *Additional resources*.

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

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

1.1.2. VMware vSphere to OpenShift Virtualization concept mapping

If you are familiar with VMware vSphere, you can map its concepts to the OpenShift Virtualization components that accomplish similar tasks.

However, because OpenShift Virtualization is conceptually different from vSphere, and much of its functionality comes from the underlying OpenShift Dedicated, OpenShift Virtualization does not have direct alternatives for all vSphere concepts or components.

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

vSphere concept	OpenShift Virtualization	Explanation
Datastore	Persistent volume (PV) Persistent volume claim (PVC)	Stores VM disks. A PV represents existing storage and is attached to a VM through a PVC. When configured for shared access, PVCs can be mounted by multiple VMs simultaneously.
Dynamic Resource Scheduling (DRS)	Pod eviction policy Descheduler	Provides active resource balancing. A combination of pod eviction policies and a descheduler allows VMs to be live migrated to more appropriate nodes to keep node resource utilization manageable.
NSX	OVN-Kubernetes	Provides an overlay network configuration. There is no direct equivalent for NSX in OpenShift Virtualization, but you can use the OVN-Kubernetes network provider.
Storage Policy Based Management (SPBM)	Storage class	Provides policy-based storage selection. Storage classes represent various storage types and describe storage capabilities, such as quality of service, backup policy, reclaim policy, and whether volume expansion is allowed. A PVC can request a specific storage class to satisfy application requirements.
vCenter vRealize Operations	OpenShift Metrics and Monitoring	Provides host and VM metrics. You can view metrics and monitor the overall health of the cluster and VMs by using the OpenShift Dedicated web console.
vMotion	Live migration	Moves a running VM to another node without interruption. For live migration to be available, the PVC attached to the VM must use storage that supports live migration.

1.1.3. Supported cluster versions on Google Cloud

OpenShift Virtualization on Google Cloud is supported on OpenShift Dedicated using either Google Cloud Hyperdisk or Google Cloud NetApp Volumes (GCNV) for persistent storage.

Refer to the following table for the minimum version you need to install based on your chosen storage solution.

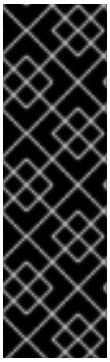
Component	Version required with Google Cloud Hyperdisk	Version required with Google Cloud NetApp Volumes (GCNV)
OpenShift Dedicated	4.21.5 or later	4.21 or later
OpenShift Virtualization Operator	4.21.1 or later	4.21.2 or later
NetApp Trident CSI Operator	N/A	26.02.0 or later

1.1.4. Volume and access modes for VM disks

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

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

- **ReadWriteMany** (RWX) access mode is required for live migration.
- The **Block** volume mode performs significantly better than the **Filesystem** volume mode. This is because the **Filesystem** volume mode uses more storage layers, including a file system layer and a disk image file. These layers are not necessary for VM disk storage.



IMPORTANT

You cannot live migrate virtual machines with the following configurations:

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

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

1.1.5. Additional resources

- OVN-Kubernetes
- Glossary of common terms for OpenShift Dedicated storage
- Using the OpenShift Assisted Installer Service to Deploy an OpenShift Cluster on Bare Metal and vSphere
- NIST-certified tool
- Red Hat Ecosystem Catalog
- Pod disruption budgets
- About live migration
- Configure eviction and run strategies

1.2. SECURITY POLICIES

OpenShift Virtualization provides built-in security features and authorization policies to protect virtual machine workloads and ensure secure cluster operations across your environment.

Key points

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

1.2.1. Workload security

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

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

1.2.2. TLS certificates

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

1.2.2.1. Automatic renewal schedules

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

- KubeVirt certificates are renewed daily.
- Containerized Data Importer controller (CDI) certificates are renewed every 15 days. Automatic TLS certificate rotation does not disrupt any operations. For example, the following operations continue to function without any disruption:
- Migrations
- Image uploads
- VNC and console connections

1.2.3. Authorization

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

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

1.2.3.1. Default cluster roles for OpenShift Virtualization

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

Table 1.2. OpenShift Virtualization cluster roles

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

1.2.3.2. RBAC roles for storage features in OpenShift Virtualization

Cluster-wide and namespaced RBAC roles enable the Containerized Data Importer (CDI) to manage storage resources, data volumes, and virtual machine disk operations.

1.2.3.2.1. Cluster-wide RBAC roles

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

CDI cluster role	Resources	Verbs
cdi.kubevirt.io:admin	**datavolumes**, **uploadtokenrequests**	* (all)
	datavolumes/source	**create**

CDI cluster role	Resources	Verbs
cdi.kubevirt.io:edit	**datavolumes**, **uploadtokenrequests**	*
	datavolumes/source	**create**
cdi.kubevirt.io:view	**cdiconfigs**, **dataimportcrons**, **datasources**, **datavolumes**, **objecttransfers**, **storageprofiles**, **volumeimportsources**, **volumeuploadsources**, **volumeclonesources**	**get**, **list**, **watch**
	datavolumes/source	**create**
cdi.kubevirt.io:config-reader	**cdiconfigs**, **storageprofiles**	**get**, **list**, **watch**

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

API group	Resources	Verbs
rbac.authorization.k8s.io	**clusterrolebindings**, **clusterroles**	**get**, **list**, **watch**, **create**, **update**, **delete**
security.openshift.io	**securitycontextconstraints**	**get**, **list**, **watch**, **update**, **create**
apiextensions.k8s.io	**customresourcedefinitions**, **customresourcedefinitions/status**	**get**, **list**, **watch**, **create**, **update**, **delete**
cdi.kubevirt.io	*	*
upload.cdi.kubevirt.io	*	*
admissionregistration.k8s.io	**validatingwebhookconfigurations**, **mutatingwebhookconfigurations**	**create**, **list**, **watch**

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

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

API group	Resources	Verbs
"" (core)	**events**	**create**, **patch**
"" (core)	**persistentvolumeclaims**	**get**, **list**, **watch**, **create**, **update**, **delete**, **deletecollection**, **patch**
"" (core)	**persistentvolumes**	**get**, **list**, **watch**, **update**
"" (core)	**persistentvolumeclaims/finalizers**, **pods/finalizers**	**update**
"" (core)	**pods**, **services**	**get**, **list**, **watch**, **create**, **delete**
"" (core)	**configmaps**	**get**, **create**
storage.k8s.io	**storageclasses**, **csidrivers**	**get**, **list**, **watch**
config.openshift.io	**proxies**	**get**, **list**, **watch**

API group	Resources	Verbs
cdi.kubevirt.io	*	*
snapshot.storage.k8s.io	**volumesnapshots**, **volumesnapshotclasses**, **volumesnapshotcontents**	**get**, **list**, **watch**, **create**, **delete**
snapshot.storage.k8s.io	**volumesnapshots**	**update**, **deletecollection**
apiextensions.k8s.io	**customresourcedefinitions**	**get**, **list**, **watch**
scheduling.k8s.io	**priorityclasses**	**get**, **list**, **watch**
image.openshift.io	**imagestreams**	**get**, **list**, **watch**
"" (core)	**secrets**	**create**
kubevirt.io	**virtualmachines/finalizers**	**update**

1.2.3.2.2. Namespaced RBAC roles

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

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

API group	Resources	Verbs
monitoring.coreos.com	**servicemonitors**, **prometheusrules**	**get**, **list**, **watch**, **create**, **delete**, **update**, **patch**
coordination.k8s.io	**leases**	**get**, **create**, **update**

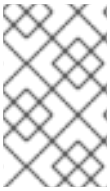
Table 1.7. Namespaced roles for the **cdi-controller** service account

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

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

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

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



NOTE

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

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

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

scc.AllowHostDirVolumePlugin = true

This allows virtual machines to use the hostpath volume plugin.

scc.AllowPrivilegedContainer = false

This ensures the **virt-launcher** pod is not run as a privileged container.

scc.AllowedCapabilities = []corev1.Capability{"SYS_NICE", "NET_BIND_SERVICE"}

- **SYS_NICE** allows setting the CPU affinity.
- **NET_BIND_SERVICE** allows DHCP and Slirp operations.

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

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

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

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

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

1.2.4. Additional resources

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

1.3. OPENSHIFT VIRTUALIZATION ARCHITECTURE

OpenShift Virtualization architecture consists of several Operators and components that manage the lifecycle, storage, networking, and scheduling of virtual machine workloads within the cluster.

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

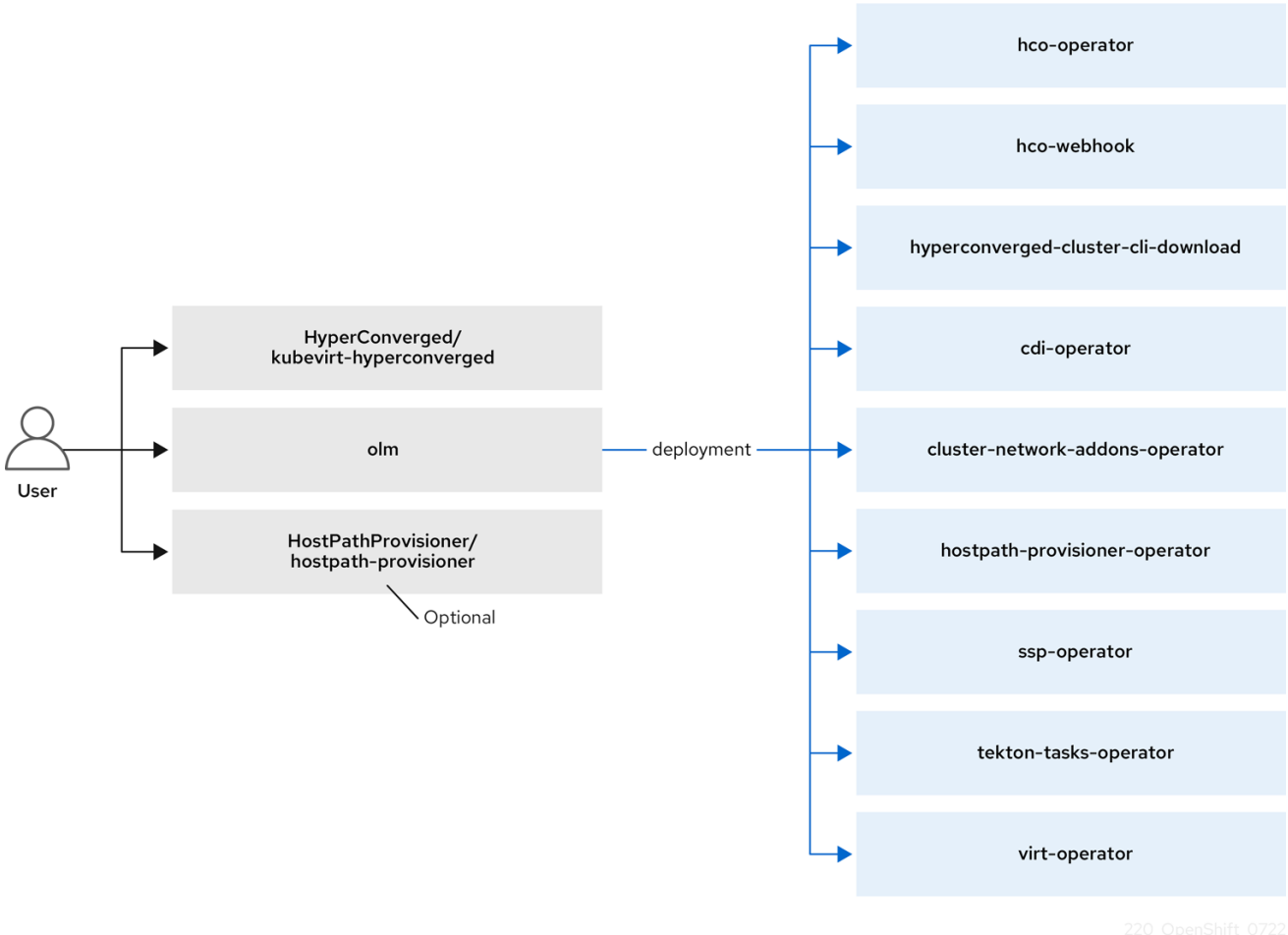
- Compute: **virt-operator**
- Storage: **cdi-operator**
- Network: **cluster-network-addons-operator**
- Scaling: **ssp-operator**

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

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

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

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



1.3.1. HyperConverged Operator (HCO)

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

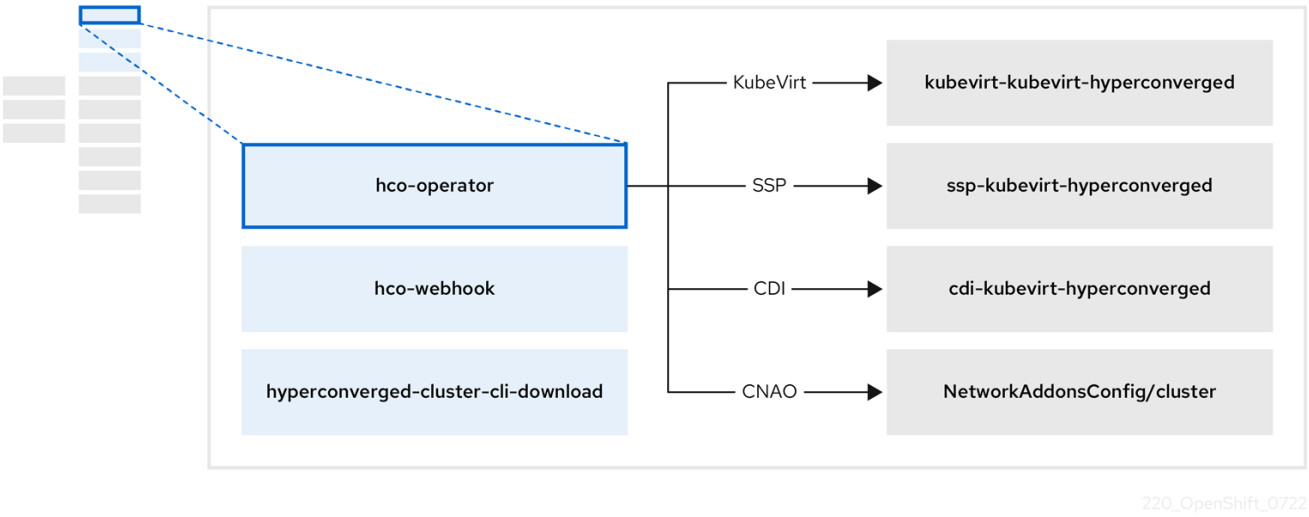


Table 1.8. HyperConverged Operator components

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

1.3.2. Containerized Data Importer (CDI) Operator

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

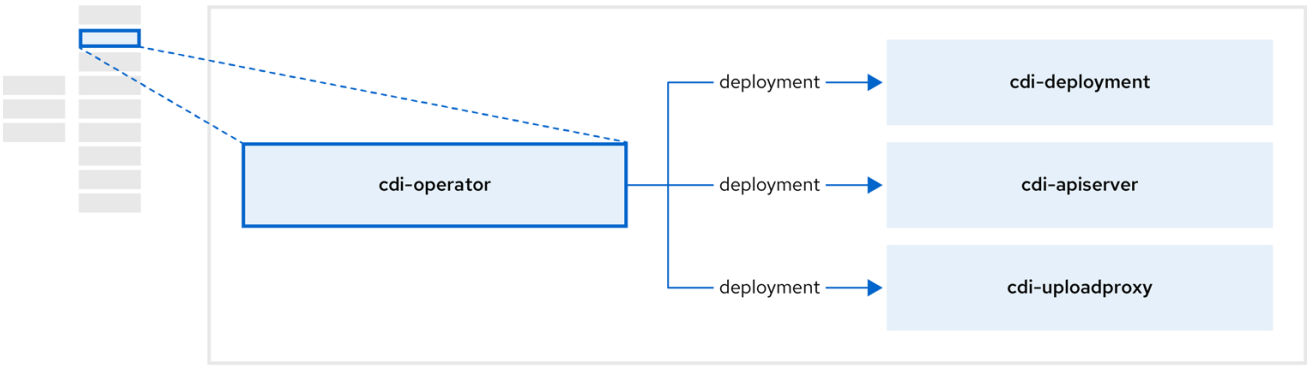


Table 1.9. CDI Operator components

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

1.3.3. Hostpath Provisioner (HPP) Operator

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

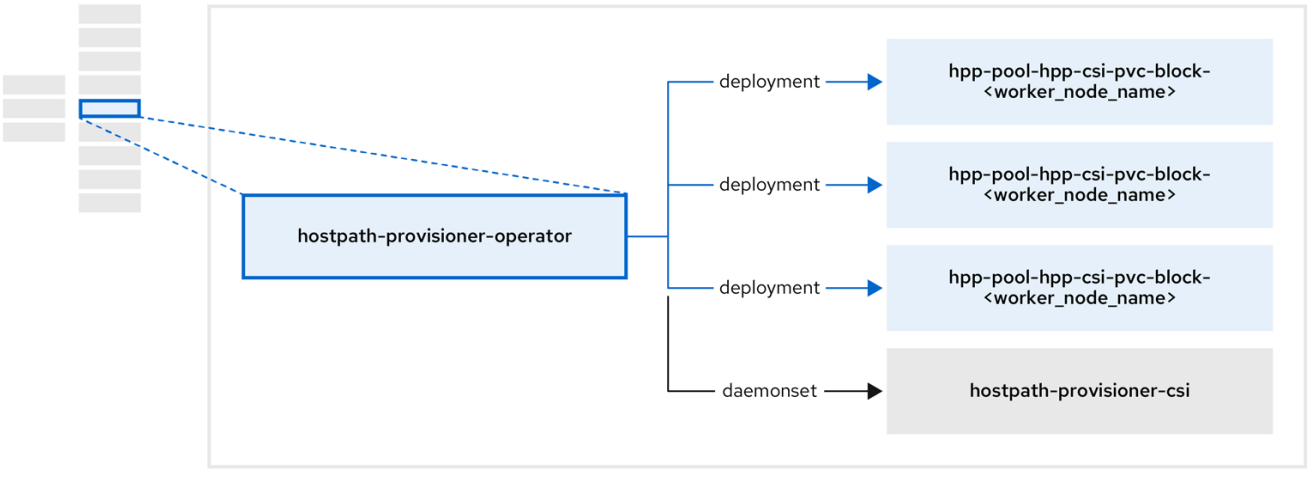


Table 1.10. HPP Operator components

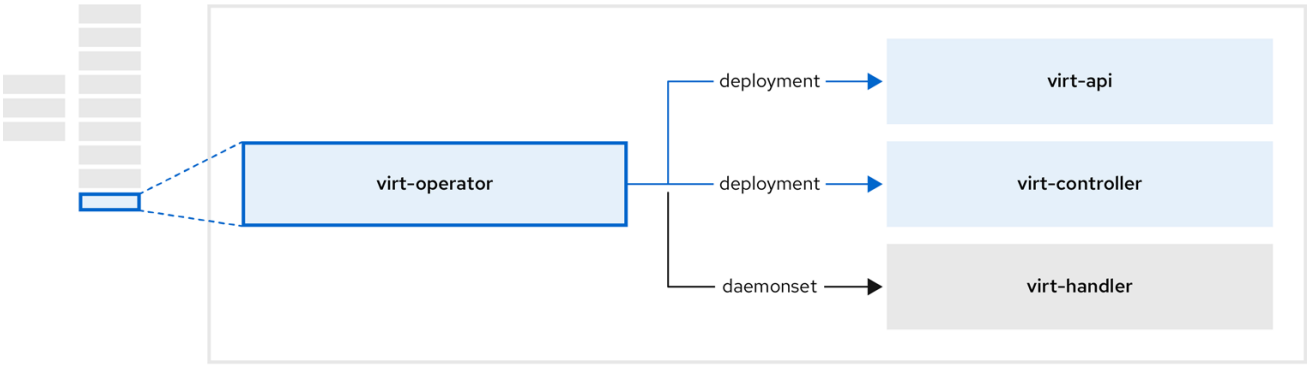
Component	Description
deployment/hpp-pool-hpp-csi-pvc-block-<worker_node_name>	Provides a worker for each node where the HPP is designated to run. The pods mount the specified backing storage on the node.
daemonset/hostpath-provisioner-csi	Implements the Container Storage Interface (CSI) driver interface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

1.3.4. Scheduling, Scale, and Performance (SSP) Operator

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

1.3.5. OpenShift Virtualization Operator

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



220_OpenShift_0622

Table 1.11. virt-operator components

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

Component	Description
pod/virt-launcher	Contains the VM that was created by the user as implemented by **libvirt** and **qemu**.

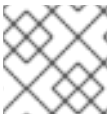
1.3.6. Additional resources

- virtctl client commands

CHAPTER 2. GETTING STARTED

2.1. GETTING STARTED WITH OPENSHIFT VIRTUALIZATION

Explore OpenShift Virtualization by taking guided tours, installing the Operator, and configuring a basic environment. Learn how to migrate from your current platform, then learn more about how to deploy and manage virtual machines (VMs) by following the additional resources links.



NOTE

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

2.1.1. Additional resources

- Prepare your cluster for OpenShift Virtualization
- Use a CSI-enabled storage provider
- Configure local storage for virtual machines
- Install the OpenShift Virtualization Operator
- Specify nodes for virtual machines
- Install and use the **virtctl** command-line interface (CLI) tool
- Create a VM by using the web console
- Import a custom image from a web page
- Upload an image from your local machine
- Clone a persistent volume claim (PVC)
- Connect to a virtual machine console
- Connect to the desktop viewer by using the web console
- Manage a VM by using the web console
- Export a VM
- Review post-installation configuration options
- Configure storage options and automatic boot source updates
- Learn about monitoring and health checks
- Learn about live migration

2.2. USING THE CLI TOOLS

You can manage OpenShift Virtualization resources by using the **virtctl** command-line tool. Virtual machine (VM) commands can also be used to manage virtual machine instances (VMIs) unless otherwise specified.



NOTE

You can access and change VM disk images by using the **libguestfs** command-line tool. You deploy **libguestfs** by using the **virtctl libguestfs** command.

2.2.1. Install the virtctl binary

Download and install the **virtctl** binary that is available from the OpenShift Dedicated web console. You can run **virtctl** on Red Hat Enterprise Linux (RHEL) 9 or later, other Linux operating systems, Windows, or macOS.

Procedure

1. Navigate to the **Virtualization** page in the web console.
2. Click the **Question Mark (?)** icon in the top right corner of screen.
3. Select **Command Line Tools** from the menu.
4. Locate the **virtctl - KubeVirt command line interface** section of the page.
5. Click the **Download virtctl** link to download the **virtctl** binary for your operating system.
6. Install **virtctl**:
   - For RHEL and other Linux operating systems:
     a. Decompress the archive file:

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

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

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

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

     ```
     $ echo $PATH
     ```

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

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

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

```
C:\> path
```

- For macOS:

  a. Decompress the archive file.

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

```
echo $PATH
```

2.2.2. virtctl information commands

You can use the following **virtctl** commands to view the client and server versions and to list the commands and options that are available.

Table 2.1. Information commands

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

2.2.3. VM information commands

You can use the following **virtctl** commands to view the file systems, operating systems, and logged-in users of a guest machine.

Table 2.2. VM information commands

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

2.2.4. VM manifest creation commands

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

Table 2.3. VM manifest creation commands

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

Command	Description
virtctl create preference --namespace <namespace_value>	Create a manifest for a namespaced VM preference.

2.2.5. VM management commands

You can use the following **virtctl** commands to start, stop, pause, restart, and migrate virtual machines (VMs).

Table 2.4. VM management commands

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

2.2.6. VM connection commands

You can use the following **virtctl** commands to connect to the console of a virtual machine (VM), open an SSH connection, copy files, and expose a port as a service.

Table 2.5. VM connection commands

Command	Description
virtctl console <vm_name>	Connect to the serial console of a VM.

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

2.2.7. VM export commands

You can use the following **virtctl vmexport** commands to export a volume from a virtual machine (VM), a VM snapshot, or a persistent volume claim (PVC). These commands create the export, download the volume, and retrieve the manifest.

Table 2.6. VM export commands

Command	Description

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

Command	Description
virtctl vmexport download export --manifest --include-secret	Retrieve the manifest for an existing export. The manifest includes the header secret.
virtctl vmexport download export --manifest --manifest-output-format=json	Retrieve the manifest for an existing export in json format. The manifest does not include the header secret.
virtctl vmexport download export --manifest --include-secret --output=manifest.yaml	Retrieve the manifest for an existing export. The manifest includes the header secret and writes it to the file specified.

2.2.8. Hot plug and hot unplug commands

You can use the following **virtctl** commands to add or remove a virtual disk while a virtual machine (VM) or virtual machine instance (VMI) is running.

Table 2.7. Hot plug and hot unplug commands

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

2.2.9. Image upload commands

You can use the following **virtctl image-upload** commands to upload a virtual machine (VM) image to a new or existing data volume.

Table 2.8. Image upload commands

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

2.2.10. Deploy libguestfs by using virtctl

You can use the **virtctl guestfs** command to deploy an interactive container that has **libguestfs-tools** installed and a persistent volume claim (PVC) attached. Use the container to inspect and modify the VM disk on the PVC.

Procedure

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

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



IMPORTANT

The **<pvc_name>** argument is required. If you do not include it, an error message appears.

2.2.11. Libguestfs and virtctl guestfs commands

The **libguestfs** tools let you access and modify virtual machine (VM) disk images. You can run them in the container that **virtctl guestfs** deploys to view and edit files in a guest, clone VMs, and resize disks.

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

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

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

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

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

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

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



NOTE

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

2.2.12. Additional resources

- Red Hat Ansible Automation Hub
- **libguestfs**

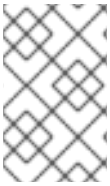
CHAPTER 3. INSTALLING

3.1. PREPARING YOUR CLUSTER FOR OPENSHIFT VIRTUALIZATION

Review platform compatibility information before you install OpenShift Virtualization. For detailed system requirements, see "Hardware, software, and operational requirements" in the Additional resources section.

3.1.1. OpenShift Virtualization on AWS bare metal

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



NOTE

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

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

Installing

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

Accessing virtual machines (VMs)

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



NOTE

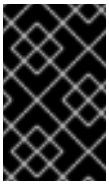
The load balancer approach is preferable because OpenShift Dedicated automatically creates the load balancer in AWS and manages its lifecycle. A security group is also created for the load balancer, and you can use annotations to attach existing security groups. When you remove the service, OpenShift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

Networking

- You cannot use Single Root I/O Virtualization (SR-IOV) or bridge Container Network Interface (CNI) networks, including virtual LAN (VLAN). If your application requires a flat layer 2 network or control over the IP pool, consider using OVN-Kubernetes secondary overlay networks.
- If your application requires a flat layer 2 network that does not need egress traffic, consider using OVN-Kubernetes secondary overlay networks with a **Layer2** topology.

Storage

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



IMPORTANT

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

Hosted control planes (HCPs)

- You can run OpenShift Virtualization on HCP clusters that use AWS bare-metal nodes. However, using OpenShift Virtualization VMs as HCP nodes is not currently supported on AWS.

3.1.2. Additional resources

- Networking overview
- Exposing a virtual machine by using a service
- Google Cloud NetApp Volumes
- GCNV storage pool limits
- GCNV service levels
- Glossary of common terms for OpenShift Dedicated storage

3.2. INSTALLING OPENSHIFT VIRTUALIZATION

Install OpenShift Virtualization to add virtualization functionality to your OpenShift Dedicated cluster.

3.2.1. Install the OpenShift Virtualization Operator by using the web console

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

Prerequisites

- Install OpenShift Dedicated 4 on your cluster.
- Log in to the OpenShift Dedicated web console as a user with **cluster-admin** permissions.
- Create a machine pool based on a bare metal compute node instance type. For more information, see "Creating a machine pool" in the Additional resources of this section.

Procedure

1. From the **Administrator** perspective, click **Ecosystem → Software Catalog**.
2. In the **Filter by keyword** field, type **Virtualization**.

3. Select the **OpenShift Virtualization Operator** tile with the **Red Hat** source label.
4. Read the information about the Operator and click **Install**.
5. On the **Install Operator** page:
    a. Select **stable** from the list of available **Update Channel** options. This ensures that you install the version of OpenShift Virtualization that is compatible with your OpenShift Dedicated version.
    b. For **Installed Namespace**, ensure that the **Operator recommended namespace** option is selected. This installs the Operator in the mandatory **openshift-cnv** namespace, which is automatically created if it does not exist.



WARNING

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

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



WARNING

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

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

Verification

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

3.2.2. Subscribe to the OpenShift Virtualization catalog by using the CLI

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

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

Prerequisites

- You have installed OpenShift Dedicated 4 on your cluster.
- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have created a machine pool based on a bare metal compute node instance type.

Procedure

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

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-cnv
  labels:
    openshift.io/cluster-monitoring: "true"
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: kubevirt-hyperconverged-group
  namespace: openshift-cnv
spec:
  targetNamespaces:
    - openshift-cnv
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.22.9
  channel: "stable"
```

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

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

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

Verification

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

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

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

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

   Example output:

   ```
   NAME                                      DISPLAY                    VERSION   REPLACES
   PHASE
   kubevirt-hyperconverged-operator.v4.22.9   OpenShift Virtualization   4.22.9    kubevirt-
   hyperconverged-operator.v4.21.0   Succeeded
   ```

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

   ```
   $ oc get hyperconvergeds.v1beta1.hco.kubevirt.io -n openshift-cnv kubevirt-hyperconverged
   -o json | jq .status.versions
   ```

   Example output:

   ```
   {
   "name": "operator",
   "version": "4.22.9"
   }
   ```

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

   ```
   $ oc get hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
   -o json | jq -r '.status.conditions[] | {type,status}'
   ```

   Example output:

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

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

3.2.3. Deploy the OpenShift Virtualization Operator by using the CLI

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

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have logged in as a user with **cluster-admin** privileges.
- You have subscribed to the OpenShift Virtualization catalog in the **openshift-cnv** namespace.
- You have created a machine pool based on a bare metal compute node instance type.

Procedure

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

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

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

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

Verification

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

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

The following output displays if deployment was successful:

```
NAME                                       DISPLAY                    VERSION   REPLACES   PHASE
kubevirt-hyperconverged-operator.v4.22.9   OpenShift Virtualization   4.22.9
Succeeded
```

3.2.4. Additional resources

- Creating a machine pool
- Configure certificate rotation
- Creating a hostpath provisioner with a basic storage pool

3.3. UNINSTALLING OPENSHIFT VIRTUALIZATION

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

To uninstall OpenShift Virtualization, perform the following tasks:

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

3.3.1. Prerequisites

- Delete all virtual machine instances. You cannot uninstall OpenShift Virtualization while its workloads remain on the cluster.

3.3.2. Delete the HyperConverged custom resource

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

Prerequisites

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

Procedure

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

3.3.3. Deleting Operators from a cluster using the web console

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



WARNING

Failed Operator installations do not automatically remove created permissions. Audit and remove remaining permissions to secure the namespace.

For more information, see Audit and secure delegated Operator installations (Red Hat Knowledgebase).

Prerequisites

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

Procedure

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



NOTE

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

3.3.4. Delete a namespace using the web console

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

Prerequisites

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

Procedure

1. Navigate to **Administration → Namespaces**.

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

3.3.5. Delete OpenShift Virtualization custom resource definitions

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

Prerequisites

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

Procedure

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

3.3.6. Uninstall OpenShift Virtualization by using the CLI

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

Prerequisites

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

Procedure

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

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

2. Delete the OpenShift Virtualization Operator subscription:

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

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

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

4. Delete the OpenShift Virtualization namespace:

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

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

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

Example output:

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

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

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

3.3.7. Additional resources

- Deleting the **HyperConverged** custom resource
- Deleting Operators from a cluster using the web console
- Deleting a namespace using the web console
- Deleting OpenShift Virtualization custom resource definitions
- Deleting a virtual machine using the web console
- Deleting a standalone virtual machine instance using the CLI

CHAPTER 4. POST-INSTALLATION CONFIGURATION

4.1. POSTINSTALLATION CONFIGURATION

After you have installed OpenShift Virtualization, there are procedures that you can complete to ensure that your environment is properly set up. You can configure the components that are relevant for your environment.

- 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.
- Node placement rules for OpenShift Virtualization Operators, workloads, and controllers.
- Network configuration:
  - Enabling the creation of load balancer services by using the OpenShift Dedicated web console
- Storage configuration:
  - Defining a default storage class for the Container Storage Interface (CSI)
  - Configuring local storage by using the Hostpath Provisioner (HPP)

4.1.1. Additional resources

- Specifying nodes for OpenShift Virtualization components
- Postinstallation network configuration
- Postinstallation storage configuration

4.2. SPECIFYING NODES FOR OPENSHIFT VIRTUALIZATION COMPONENTS

You can configure node placement rules to specify where OpenShift Virtualization Operators, workloads, and controllers are deployed. Custom placement rules allow you to isolate virtual machine (VM) traffic or dedicate specialized compute resources to critical workloads.



IMPORTANT

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

4.2.1. About node placement rules for OpenShift Virtualization components

You can use node placement rules to deploy virtual machines only on nodes intended for virtualization workloads, to deploy Operators only on infrastructure nodes, or to maintain separation between workloads.

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

nodeSelector

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

affinity

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

tolerations

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

4.2.2. Apply node placement rules

You can apply node placement rules to ensure that virtualization components run on the most suitable nodes for your workload requirements. You can apply node placement rules by editing OpenShift Virtualization objects.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You are logged in with cluster administrator permissions.

Procedure

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

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

2. Save the file to apply the changes.

4.2.3. Node placement rule examples

Examples of node placement rules for different objects that determine where OpenShift Virtualization components are deployed.

4.2.3.1. Subscription object node placement rule examples

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



NOTE

- You cannot configure node placement rules for the **Subscription** object by using the web console.
- The **Subscription** object does not support the **affinity** node placement rule.

Example Subscription object with nodeSelector rule:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.22.9
  channel: "stable"
  config:
    nodeSelector:
      example.io/example-infra-key: example-infra-value
```

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

The following pods are affected by the **Subscription** object node placement rules. These pods are created when the OpenShift Virtualization Operator is first installed, even without an HyperConverged custom resource (CR):

- **aaq-operator**
- **cdi-operator**
- **cluster-network-addons-operator**
- **hco-operator**
- **hco-webhook**
- **hostpath-provisioner-operator**
- **hyperconverged-cluster-cli-download**
- **kubevirt-migration-operator**
- **ssp-operator**
- **virt-operator**

Example Subscription object with tolerations rule:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source:  redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.22.9
  channel: "stable"
```

```
  config:
    tolerations:
    - key: "key"
      operator: "Equal"
      value: "virtualization"
      effect: "NoSchedule"
```

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

4.2.3.2. HyperConverged object node placement rule example

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

Example HyperConverged object with nodeSelector rule:

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  deployment:
    nodePlacements:
      infra:
        nodeSelector:
          example.io/example-infra-key: example-infra-value
      workload:
        nodeSelector:
          example.io/example-workloads-key: example-workloads-value
```

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

Example HyperConverged object with affinity rule:

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  deployment:
    nodePlacements:
      infra:
        affinity:
          nodeAffinity:
            requiredDuringSchedulingIgnoredDuringExecution:
              nodeSelectorTerms:
              - matchExpressions:
```

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

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

Example HyperConverged object with tolerations rule:

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  deployment:
    nodePlacements:
      workload:
        tolerations:
        - key: "key"
          operator: "Equal"
          value: "virtualization"
          effect: "NoSchedule"
```

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

The following pods are affected by the **HyperConverged** object node placement rules. These pods are created when an HyperConverged CR is deployed:

- **cdi-apiserver**
- **cdi-deployment**
- **cdi-uploadproxy**
- **kubemacpool-cert-manager**
- **kubemacpool-mac-controller-manager**
- **kubevirt-apiserver-proxy**
- **kubevirt-console-plugin**
- **kubevirt-ipam-controller-manager**
- **kubevirt-migration-controller**
- **virt-api**
- **virt-controller**
- **virt-exportproxy**
- **virt-template-validator**

4.2.3.3. HostPathProvisioner object node placement rule example

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



WARNING

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

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

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

Example HostPathProvisioner object with nodeSelector rule:

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
```

```
spec:
  imagePullPolicy: IfNotPresent
  pathConfig:
    path: "</path/to/backing/directory>"
    useNamingPrefix: false
  workload:
    nodeSelector:
      example.io/example-workloads-key: example-workloads-value
```

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

4.2.4. Additional resources

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

4.3. POSTINSTALLATION NETWORK CONFIGURATION

By default, OpenShift Virtualization uses a single internal pod network after installation.

4.3.1. Create a Linux bridge node network configuration policy

You can configure a Linux bridge network for live migration or external access to virtual machines (VMs) by creating a **NodeNetworkConfigurationPolicy** (NNCP) manifest.

Prerequisites

- You have installed the Kubernetes NMState Operator.

Procedure

1. Configure the **NNCP** manifest. This example includes sample values that you must replace with your own information.

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy
spec:
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with eth1 as a port
        type: linux-bridge
        state: up
        ipv4:
          enabled: false
        bridge:
          options:
            stp:
```

```
      enabled: false
    port:
      - name: eth1
```

- **metadata.name**: Defines the name of the node network configuration policy.
- **spec.desiredState.interfaces.name**: Defines the name of the new Linux bridge.
- **spec.desiredState.interfaces.description**: Optional field that can be used to define a human-readable description for the bridge.
- **spec.desiredState.interfaces.type**: Defines the interface type. In this example, the type is a Linux bridge.
- **spec.desiredState.interfaces.state**: Defines the requested state for the interface after creation.
- **spec.desiredState.interfaces.ipv4.enabled**: Defines whether the ipv4 protocol is active. Setting this to **false** disables IPv4 addressing on this bridge.
- **spec.desiredState.interfaces.bridge.options.stp.enabled**: Defines whether Spanning Tree Protocol (STP) is active. Setting this to **false** disables STP on this bridge.
- **spec.desiredState.interfaces.bridge.port.name**: Defines the node NIC that the bridge is attached to.



NOTE

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

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

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

2. Create the **NNCP** manifest:

```
$ oc apply -f linux-bridge-nncp.yaml
```

4.3.2. Create a Linux bridge NAD by using the web console

Use the OpenShift Dedicated web console to create a **NetworkAttachmentDefinition** (**net-attach-def**) that connects pods and virtual machines to a layer-2 network.

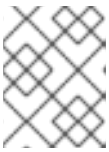


WARNING

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

Procedure

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



NOTE

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

3. Enter a unique **Name** and optional **Description**.
4. Select **CNV Linux bridge** from the **Network Type** list.
5. Enter the name of the bridge in the **Bridge Name** field.
6. Optional: If the resource has VLAN IDs configured, enter the ID numbers in the **VLAN Tag Number** field.
7. Optional: Select **MAC Spoof Check** to enable MAC spoof filtering. This feature provides security against a MAC spoofing attack by allowing only a single MAC address to exit the pod.
8. Optional: In the **YAML** tab, add the **spec.config.disableContainerInterface** field. When set to **true**, the Bridge CNI plug-in skips creating a standard container virtual ethernet (**veth**) interface inside the pod's network namespace. This enables the virtualization networking backend to attach the layer-2 interface directly to the guest VM instead of a standard container.
Example YAML:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: bridge-network
  annotations:
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/br1
spec:
  config: |
    {
# ...
      "disableContainerInterface": true,
# ...
    }
```

9. Click **Create**.

4.3.3. Configure a dedicated secondary network for live migration

Configure a dedicated secondary network for live migration to minimize the effects of network saturation on tenant workloads during live migration.

Prerequisites

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

Procedure

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

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

   - **metadata.name**: Defines the name of the **NetworkAttachmentDefinition** object.
   - **config.master**: Defines the name of the NIC to be used for live migration.
   - **config.type**: Defines the name of the CNI plugin that provides the network for the **NetworkAttachmentDefinition**.
   - **config.range**: Defines an IP address range for the secondary network. This range must not overlap the IP addresses of the main network.

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

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

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

Example **HyperConverged** manifest:

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  virtualization:
    liveMigrationConfig:
      completionTimeoutPerGiB: 800
      network: <network>
      parallelMigrationsPerCluster: 5
      parallelOutboundMigrationsPerNode: 2
      progressTimeout: 150
# ...
```

- **spec.virtualization.liveMigrationConfig.network**: Defines the name of the Multus **NetworkAttachmentDefinition** object to be used for live migrations.

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

Verification

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

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

4.3.4. Select a dedicated migration network by using the web console

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

Prerequisites

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

Procedure

1. Go to **Virtualization → Settings** in the OpenShift Dedicated web console.
2. On the **Cluster** tab, click **General settiings**.
3. Click **Live Migration**.
4. Select the network from the **Live migration network** list.

4.3.5. Enable load balancer services by using the web console

You can enable the creation of load balancer services for a virtual machine (VM) from the OpenShift Dedicated web console. This setting must be on before you can create a load balancer service for SSH access.

Prerequisites

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

Procedure

1. Go to **Virtualization → Settings**.
2. Click **Cluster**.
3. Expand **General settings** and **SSH configuration**.
4. Set **SSH over LoadBalancer service** to on.

4.3.6. Configure additional routes to the cdi-uploadproxy service

Configure additional routes to the **cdi-uploadproxy** service to enable users to upload virtual machine images from outside the cluster.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You logged in to the cluster as a user with the **cluster-admin** role.

Procedure

1. Configure the route to the external host by running the following command:

```
$ oc create route reencrypt <route_name> -n openshift-cnv \
    --insecure-policy=Redirect \
    --hostname=<host_name_or_address> \
    --service=cdi-uploadproxy
```

where:

<route_name>

Specifies the name to assign to this custom route.

<host_name_or_address>

Specifies the fully qualified domain name or IP address of the external host providing image upload access.

2. Run the following command to annotate the route. This ensures that the correct Containerized Data Importer (CDI) CA certificate is injected when certificates are rotated:

```
$ oc annotate route <route_name> -n openshift-cnv \
```

```
operator.cdi.kubevirt.io/injectUploadProxyCert="true"
```

where:

<route_name>

Specifies the name of the route you created.

4.4. POSTINSTALLATION STORAGE CONFIGURATION

If your storage provider is not recognized by the Containerized Data Importer (CDI), you must configure storage profiles after you install OpenShift Virtualization. Storage profiles provide recommended storage settings based on the associated storage class.

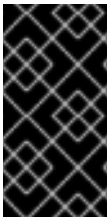
Optional: You can configure local storage by using the hostpath provisioner (HPP).

See the "Storage configuration overview" documentation for more options, including configuring the CDI, data volumes, and automatic boot source updates.

4.4.1. Configuring local storage by using the HPP

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

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

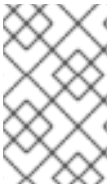


IMPORTANT

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

4.4.1.1. Create a storage class for the CSI driver with the storagePools stanza

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



NOTE

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

Prerequisites

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

Procedure

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

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

- **reclaimPolicy** defines whether the underlying storage is deleted or retained when a user deletes a PVC. The two possible **reclaimPolicy** values are **Delete** and **Retain**. If you do not specify a value, the default value is **Delete**.
- **volumeBindingMode** defines the timing of PV creation. In this example, the **WaitForFirstConsumer** configuration delays PV creation until the scheduler assigns a pod to a specific node.



NOTE

Virtual machines use data volumes based on local PVs, which reside on specific nodes. When the system prepares a disk image for the virtual machine, the scheduler might not place the virtual machine on the node where it pinned the local storage PV.

+ To solve this problem, use the Kubernetes pod scheduler to bind the persistent volume claim (PVC) to a PV on the correct node. Setting the **volumeBindingMode** parameter of the **StorageClass** to **WaitForFirstConsumer** delays PV binding and provisioning until you create a pod that uses the PVC.

- **parameters.storagePool** defines the name of the storage pool defined in the HPP custom resource (CR).

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

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

4.4.2. Additional resources

- Configuring storage profiles
- Storage configuration overview

4.5. VIRTUAL MACHINE RECOVERY FROM NODE FAILURES

To ensure that virtual machines (VMs) recover automatically when a node fails, configure node health checks, automated remediation, and capacity planning. These recommendations come from chaos testing results and help minimize VM downtime during node failure conditions.

4.5.1. Node health monitoring for protection of virtual machines

By default, virtual machines (VMs) are not automatically recovered to other available nodes when a node fails. To minimize VM downtime and ensure workload resilience, configure node health monitoring and automated remediation.

Without the Node Health Check Operator and a remediation Operator such as Self Node Remediation (SNR) or Fence Agents Remediation (FAR), VMs experience downtime until the node recovers. Apply the following configurations to ensure your workloads are resilient under node failure conditions:

- Configure VMs with **runStrategy: Always** to allow recovery to other available nodes.
- Deploy and configure the Node Health Check Operator to monitor node status changes and trigger automated remediation.
- Configure a remediation operator, such as SNR or FAR, to recover workloads from failed nodes.
- Plan node capacity to ensure that enough resources are available to host VMs that migrate from failed nodes.

4.5.1.1. Configure VM run strategy

You can configure a run strategy for a virtual machine (VM) by using the command line. The run strategy controls whether a VM automatically restarts after disruptions such as node failures or maintenance events.

Prerequisites

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

Procedure

- Edit the **VirtualMachine** resource by running the following command:

```
$ oc edit vm <vm_name> -n <namespace>
```

Example run strategy:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  runStrategy: Always
# ...
```

4.5.2. Configure node health checks for virtual machines

You can configure the Node Health Check Operator to monitor node status and trigger automated remediation when a node becomes unhealthy.

Prerequisites

- You have installed the Node Health Check Operator.
- You have installed a remediation operator.
- You have cluster administrator privileges.

Procedure

1. Create a **NodeHealthCheck** custom resource (CR) to define the health check criteria and remediation strategy:

```
apiVersion: remediation.medik8s.io/v1alpha1
kind: NodeHealthCheck
metadata:
  name: nodehealthcheck-sample
spec:
  minHealthy: <count>
  pauseRequests:
    - <pause_test_cluster>
  remediationTemplate:
    apiVersion: self-node-remediation.medik8s.io/v1alpha1
    name: self-node-remediation-automatic-strategy-template
    namespace: openshift-workload-availability
    kind: SelfNodeRemediationTemplate
  escalatingRemediations:
    - remediationTemplate:
        apiVersion: self-node-remediation.medik8s.io/v1alpha1
        name: self-node-remediation-resource-deletion-template
        namespace: openshift-workload-availability
        kind: SelfNodeRemediationTemplate
      order: 1
      timeout: 300s
  selector:
    matchExpressions:
      - key: node-role.kubernetes.io/worker
        operator: Exists
  unhealthyConditions:
    - type: Ready
      status: "False"
      duration: 30s
    - type: Ready
      status: Unknown
      duration: 30s
```

where:

- **spec.minHealthy** defines the number of worker nodes required to host VMs that migrate from failed nodes.
    - For critical environments, set this value to the minimum number of nodes that you require to maintain the cluster workload.
    - For hyperconverged storage, set this value to one less than your number of nodes to limit remediation to a single worker-node failure and maintain storage stability.
- **spec.remediationTemplate** defines the remediation template to use when the Node Health Check Operator detects an unhealthy node. This example uses the Self Node Remediation Operator.
- **spec.escalatingRemediations** defines escalating remediation strategies. If the initial remediation does not resolve the issue within the specified timeout, the next remediation strategy runs.

- **spec.selector** defines the nodes to monitor. This example monitors all worker nodes.
- **spec.unhealthyConditions** defines the parameters to identify an unhealthy node.
- **spec.unhealthyConditions.duration** defines the duration that a condition must persist before remediation starts. Set lower values for faster recovery. The following table shows recommended values:

Table 4.1. Recommended unhealthyConditions duration values

Environment	Recommended duration
Critical	30-60 seconds
Standard	60-180 seconds
Conservative	180-360 seconds

2. Apply the **NodeHealthCheck** CR by running the following command:

```
$ oc apply -f nodehealthcheck-sample.yaml
```

Verification

- Verify that the **NodeHealthCheck** CR exists by running the following command:

```
$ oc get nodehealthcheck nodehealthcheck-sample
```

TIP

If remediation does not trigger as expected, verify that the remediation Operator runs correctly, and check the **NodeHealthCheck** CR events for errors:

```
$ oc describe nodehealthcheck nodehealthcheck-sample
```

4.5.3. Node remediation strategies for OpenShift Virtualization

Use the best remediation strategy for your environment to ensure that virtual machine (VM) workloads recover automatically when nodes become unhealthy.

Self Node Remediation (SNR) and Fence Agents Remediation (FAR) are the recommended remediation Operators for OpenShift Virtualization environments. Recovery time depends on the number of VMs on the failed nodes.

Self Node Remediation (SNR)

Use for nodes that do not have a management interface, or when the management interface might be unreachable. SNR does not require a management interface to function. Do not use SNR for hyperconverged storage because API unavailability can cause remediation actions that negatively impact hyperconverged storage protection domains.

Fence Agents Remediation (FAR)

The fastest remediation Operator for workload recovery time. Use FAR when a baseboard management controller (BMC) interface is available. FAR requires a management interface that is reachable from the Kubernetes pod network.

You must use FAR for hyperconverged storage configurations. For hyperconverged storage, set the Node Health Check **minHealthy** property to your total node count minus one. With this setting, FAR does not remediate more than a single worker-node failure within the same zone, storage placement, and replication group to maintain storage stability.



NOTE

- Remediation actions start after the node conditions monitored by the Node Health Check Operator exceed the time specified in **spec.unhealthyConditions[].duration**.
- Disruption to the node running the **self-node-remediation-controller-manager** pod increases recovery times.

4.5.3.1. Self Node Remediation (SNR) configuration parameters

You can tune the Self Node Remediation (SNR) Operator configuration to optimize recovery times and ensure availability of VM workloads.

SelfNodeRemediationTemplate strategies

The **SelfNodeRemediationTemplate** custom resource (CR) defines the remediation strategy to use when the Node Health Check Operator identifies a node as unhealthy. Configure the remediation strategy in a **SelfNodeRemediationTemplate** CR:

```
apiVersion: self-node-remediation.medik8s.io/v1alpha1
kind: SelfNodeRemediationTemplate
metadata:
  name: self-node-remediation-resource-deletion-template
  namespace: openshift-operators
spec:
  template:
    spec:
      remediationStrategy: ResourceDeletion
```

where:

- **spec.remediationStrategy** defines the remediation strategy. Use **ResourceDeletion** to delete pods and associated volume attachments on the unhealthy node. Use **OutOfServiceTaint** to place an **out-of-service** taint on the unhealthy node and delete pods and associated volume attachments. This strategy is faster than **ResourceDeletion**.

SelfNodeRemediationConfig parameters

The **SelfNodeRemediationConfig** CR defines the timing and connectivity parameters for the SNR Operator. Tuning these parameters based on your environment conditions helps ensure availability of workloads and prevents unnecessary remediation actions. Configure the timing and connectivity parameters in a **SelfNodeRemediationConfig** CR:

```
apiVersion: self-node-remediation.medik8s.io/v1alpha1
kind: SelfNodeRemediationConfig
metadata:
```

```
  name: self-node-remediation-config
  namespace: openshift-operators
spec:
  safeTimeToAssumeNodeRebootedSeconds: 180
  watchdogFilePath: /dev/watchdog
  isSoftwareRebootEnabled: true
  apiServerTimeout: 15s
  apiCheckInterval: 5s
  maxApiErrorThreshold: 3
  peerApiServerTimeout: 5s
  peerDialTimeout: 5s
  peerRequestTimeout: 5s
  peerUpdateInterval: 15m
```

where:

- **spec.safeTimeToAssumeNodeRebootedSeconds** defines the time in seconds to wait before assuming the node has rebooted. Set this value to the node reboot time plus 20% for faster recovery, or plus 40% for a more conservative approach.

4.5.4. Capacity planning for VM failover

Plan node capacity to ensure your cluster has enough resources to host VMs that migrate from failed nodes.

When a node fails and you have enabled remediation, VMs automatically migrate to other available nodes in the cluster. Monitor the following resources on your cluster to ensure adequate capacity for VM failover:

- VM count per node
- Available CPU
- Available memory
- Available disk
- Available network bandwidth

Check the current resource use of your nodes:

```
$ oc adm top nodes
```



NOTE

The number of VMs that a node can host depends on the **maxPods** value set in the kubelet configuration:

```
kubeletConfig:
  maxPods: 250
```

4.5.5. Additional resources

- Configure eviction and run strategies
- Run strategies
- About high availability for virtual machines
- Control virtual machine states
- Workload Availability for Red Hat OpenShift

4.6. CONFIGURING CERTIFICATE ROTATION

Configure certificate rotation parameters to replace existing certificates.

4.6.1. Configure certificate rotation

If you did not configure certificate rotation during OpenShift Virtualization installation in the web console, you can use the **HyperConverged** custom resource (CR) to apply the configuration after installation.

Prerequisites

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

Procedure

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

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

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

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  security:
    certConfig:
      ca:
        duration: 48h0m0s
        renewBefore: 24h0m0s
      server:
        duration: 24h0m0s
        renewBefore: 12h0m0s
```

- **ca.renewBefore**: Set to a value less than or equal to the value of **ca.duration**.
- **server.duration**: Set to a value less than or equal to the value of **ca.duration**.
- **server.renewBefore**; Set to a value less than or equal to the value of **server.duration**.

3. Apply updates to the **HyperConverged** CR by running the following command:

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

For example:

```
$ oc apply -f kubevirt-hyperconverged.yaml
```

4.6.2. Troubleshoot certificate rotation parameters

Deleting one or more **certConfig** values in the **HyperConverged** custom resource (CR) causes the **certConfig** values to revert to the default values.

If the default values conflict with one of the following conditions, you receive an error message instead:

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

For example, if you remove the **server.duration** value, the default value of **24h0m0s** is greater than the value of **ca.duration**, which conflicts with the specified conditions:

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  # ...
  security:
    certConfig:
      ca:
        duration: 4h0m0s
        renewBefore: 1h0m0s
      server:
        duration: 4h0m0s
        renewBefore: 4h0m0s
# ...
```

This results in the following error message:

```
error: hyperconvergeds.hco.kubevirt.io "kubevirt-hyperconverged" could not be patched: admission webhook "validate-hco.kubevirt.io" denied the request: spec.certConfig: ca.duration is smaller than server.duration
```

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

4.6.3. Additional resources

- golang **ParseDuration** format

CHAPTER 5. UPDATING

5.1. UPDATING OPENSHIFT VIRTUALIZATION

Learn how to keep OpenShift Virtualization updated and compatible with OpenShift Dedicated.

5.1.1. OpenShift Virtualization update behavior

When you install OpenShift Virtualization, you select an update channel and an approval strategy. Both settings affect supportability.

The update channel determines the version of OpenShift Virtualization that you use. The approval strategy determines whether updates occur automatically or require manual approval.

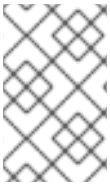
5.1.1.1. Recommended settings

To keep a supportable environment, use the following settings:

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

Most OpenShift Virtualization installations use the **stable** release channel and the **Automatic** approval strategy. Use other settings only if you understand the risks.

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



NOTE

Each minor version of OpenShift Virtualization is supported only with the corresponding OpenShift Dedicated version. For example, you must run OpenShift Virtualization 4.22 on OpenShift Dedicated 4.22.

5.1.1.2. What to expect

You can expect consistent update behavior in OpenShift Virtualization, including duration, automation, and data preservation.

- The time required to complete an update depends on your network connection. Most automatic updates complete within fifteen minutes.
- Updating OpenShift Virtualization does not interrupt network connections.
- An update preserves data volumes and their associated persistent volume claims.

5.1.1.3. How updates work

Learn how Operator Lifecycle Manager (OLM) updates the OpenShift Virtualization Operator and how update channels and approval strategies affect upgrade behavior.

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

5.1.1.4. Change update settings

You can control how and when updates are installed by changing the update channel and approval strategy for the OpenShift Virtualization Operator subscription.

Prerequisites

- You have installed the OpenShift Virtualization Operator.
- You have logged in to the OpenShift Dedicated web console as a cluster administrator.

Procedure

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

5.1.1.5. Manual approval strategy

If you use the **Manual** approval strategy, you must approve every pending update. If OpenShift Dedicated and OpenShift Virtualization updates are out of sync, your cluster becomes unsupported.

To avoid risk to cluster supportability and functionality, use the **Automatic** approval strategy. If you must use the **Manual** approval strategy, approve pending Operator updates as soon as they become available.

5.1.1.6. Approving a pending Operator update manually

If an installed Operator has the approval strategy in its subscription set to **Manual**, you must manually approve the update before installation can begin. Manual approval reviews the changes and control when updates are applied to prevent unexpected downtime.

Prerequisites

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

Procedure

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

5.1.2. Operator and resource cleanup

When updating OpenShift Virtualization to version 4.22, you can remove certain Operators and resources that are no longer required. This helps reclaim space and resources on your cluster.

5.1.3. RHEL 9 compatibility

OpenShift Virtualization 4.22 is based on Red Hat Enterprise Linux (RHEL) 9.

5.1.3.1. RHEL 9 machine type

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

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



IMPORTANT

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

5.1.4. Monitor update status

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



NOTE

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

Prerequisites

- You have logged in to the OpenShift Dedicated cluster as a cluster administrator.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Run the following command:

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

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

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

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

```
$ oc get hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv \
  -o=jsonpath='{range .status.conditions[*]}{.type}{"\t"}{.status}{"\t"}{.message}{"\n"}{end}'
```

   A successful upgrade results in the following output:

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

5.1.5. VM workload updates

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



NOTE

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

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

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

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

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

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

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

5.1.5.1. Migration attempts and timeouts

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

5 minutes

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

15 minutes

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

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



NOTE

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

5.1.5.2. Configure workload update methods

You can configure how virtual machine workloads are updated during cluster upgrades by editing the **HyperConverged** custom resource (CR).

Prerequisites

- You have enabled live migration in the cluster.



NOTE

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

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

Procedure

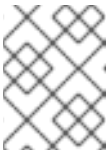
1. To open the **HyperConverged** CR in your default editor, run the following command:

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

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

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

- **spec.workloadUpdateStrategy.workloadUpdateMethods** defines the methods that can be used to perform automated workload updates. The available values are **LiveMigrate** and **Evict**. If you enable both options as shown in this example, updates use **LiveMigrate** for VMIs that support live migration and **Evict** for any VMIs that do not support live migration. To disable automatic workload updates, you can either remove the **workloadUpdateStrategy** stanza or set **workloadUpdateMethods: []** to leave the array empty.
    - **LiveMigrate** is the least disruptive update method. VMIs that support live migration are updated by migrating the virtual machine (VM) guest into a new pod with the updated components enabled. If **LiveMigrate** is the only workload update method listed, VMIs that do not support live migration are not disrupted or updated.
    - **Evict** is a disruptive method that shuts down VMI pods during upgrade. **Evict** is the only update method available if live migration is not enabled in the cluster. If a VMI is controlled by a **VirtualMachine** object that has **runStrategy: Always** configured, a new VMI is created in a new pod with updated components.
- **spec.workloadUpdateStrategy.batchEvictionSize** defines the number of VMIs that can be forced to be updated at a time by using the **Evict** method. This does not apply to the **LiveMigrate** method.
- **spec.workloadUpdateStrategy.batchEvictionInterval** defines the interval to wait before evicting the next batch of workloads. This does not apply to the **LiveMigrate** method.



NOTE

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

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

5.1.5.3. View outdated VM workloads

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



NOTE

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

Prerequisites

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

Procedure

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

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

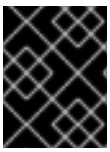
5.1.6. Early access releases

You can access development builds by subscribing to the **candidate** update channel for your version of OpenShift Virtualization.

These releases are not fully tested by Red Hat and are not supported. Use them only on non-production clusters to test new capabilities and bug fixes.

The **stable** channel matches the underlying OpenShift Dedicated version and is fully tested. It is suitable for production systems. You can switch between the **stable** and **candidate** channels in OperatorHub. Updating from a **candidate** release to a **stable** release is not tested by Red Hat.

Red Hat promotes some candidate releases to the **stable** channel. Other candidate releases might not include all GA features. Red Hat might remove some features from candidate builds before GA. Candidate releases might not offer update paths to later GA releases.



IMPORTANT

Use the candidate channel only for testing where you can delete and re-create the cluster.

5.1.7. Additional resources

- What are Operators?
- Operator Lifecycle Manager concepts and resources
- Cluster service versions (CSVs)
- About live migration
- Configure eviction and run strategies
- Configuring live migration limits and timeouts

CHAPTER 6. CREATING VIRTUAL MACHINES

6.1. CREATE VIRTUAL MACHINES BY USING THE WEB CONSOLE

You can create virtual machines (VMs) by using the OpenShift Dedicated web console. The web console creation wizard provides workflows for configuring a custom VM, creating a VM from a template, and cloning an existing VM.



IMPORTANT

You must install VirtIO drivers on Windows VMs. Download VirtIO drivers only from official Red Hat sources. For more information, see "Additional resources".

6.1.1. VM creation methods

The web console creation wizard provides three workflows for creating a virtual machine (VM). Choose the workflow that matches your situation.

Table 6.1. VM creation methods

If you want to	Use this workflow	Requirements
Build a VM with a specific combination of compute, storage, and network settings, when no existing template matches your requirements	**Custom configuration**	A boot source, such as a custom disk image, container image, or URL
Deploy a standardized VM quickly from a preconfigured image and settings	**Create from Template**	A template that has an available boot source in the cluster
Create one or more VMs that duplicate an existing VM and its configuration	**Clone existing VirtualMachine**	Access to the source VM that you want to clone

For any method, the creation wizard reference explains how boot source, instance type, and customization options affect the VM.

Additional resources

- Virtual machine creation wizard reference

6.1.2. Create a VM with custom configuration by using the web console

You can use the **Custom configuration** workflow in the web console to create a virtual machine (VM) by selecting a guest operating system, boot source, compute resources, and advanced settings.

Use this workflow when:

- No existing template matches your requirements.
- You need a specific combination of compute, storage, and network settings.
- You want to boot from a custom disk image.

Prerequisites

- You have access to the OpenShift Dedicated web console.
- You have **edit** or **admin** permissions in the target project.

Procedure

1. In the OpenShift Dedicated web console, navigate to **Virtualization → VirtualMachines**.
2. Click **Create**.
3. On the **Deployment details** page, configure the following settings:
    a. Select **Custom configuration**.
    b. In the **Name** field, enter a name for the VM. You can also click the refresh icon to generate a name automatically.
    c. Optional: Enter a **Description** for the VM.
    d. Review the **Location** to verify the target cluster and project. If the folder preview feature is enabled, you can also verify the folder. To change the location, click the edit icon.
    e. Click **Next**.
4. On the **Guest operating system** page, specify the operating system:
    a. Select the guest operating system, such as **RHEL**, **Microsoft Windows**, or **Other Linux**.
    b. Select the operating system version from the **Guest operating system type** list.
    c. Click **Next**.
5. On the **Boot source** page, specify the boot volume:
    a. Select **Boot volume** to use an existing boot volume or add a new one, or select **No boot source** to assign an empty disk that you can configure later.
    b. If you selected **Boot volume**, select an existing volume from the list, or click **Add volume** to configure a new boot volume.
    c. Click **Next**.
6. On the **Compute resources** page, define the instance type:
    a. Select an instance type series, such as **General purpose** or **Compute Exclusive**.
    b. Select the vCPU and memory size from the drop-down list.
    c. Click **Next**.
7. Optional: On the **Customization** page, configure advanced settings by navigating the following tabs:
    - **Details**: Configure the **Hostname**, **Headless mode**, **Guest system log access**, and **Deletion protection**.

- **Storage**: Add or change storage disks.
- **Network**: Add network interfaces.
- **Scheduling**: Configure scheduling requirements and the run strategy.
- **SSH**: Configure SSH key settings.
- **Initial run**: Define **Cloud-init** or **Sysprep** initialization scripts.
- **Labels and annotations**: Add labels to help organize and categorize VMs, and use annotations to store extra metadata.

a. Click **Next**.

8. On the **Review and create** page, review the VM configuration:

a. Optional: Select the **Start this VirtualMachine after creation** checkbox.

b. Click **Create VirtualMachine**.

Verification

- Navigate to **Virtualization → VirtualMachines** and verify that the VM is displayed in the list.

6.1.3. Create a VM from a template by using the web console

You can use the **Create from Template** workflow in the web console to create a virtual machine (VM) by selecting a preconfigured template that provides standardized images and settings.

Use this workflow when:

- A template matches your workload requirements and operating system.
- You want to use a preconfigured VM with standardized settings.
- You want to quickly deploy a VM with minimal customization.

Prerequisites

- You have access to the OpenShift Dedicated web console.
- You have **edit** or **admin** permissions in the target project.
- A template with an available boot source exists in the cluster.

Procedure

1. In the OpenShift Dedicated web console, navigate to **Virtualization → VirtualMachines**.
2. Click **Create**.
3. On the **Deployment details** page, configure the following settings:

a. Select **Create from Template**.

b. In the **Name** field, enter a name for the VM. You can also click the refresh icon to generate a name automatically.

c. Optional: Enter a **Description** for the VM.

d. Review the **Location** to verify the target cluster and project. If the folder preview feature is enabled, you can also verify the folder. To change the location, click the edit icon.

e. Click **Next**.

4. On the **Template** page, select a template:

a. Optional: Filter the template list. You can select a project from the **All projects** menu, apply a **Filter** for categories such as operating system or workload type, or enter a keyword in the **Filter by keyword** field.

b. Click a template tile to select it and view its details.

c. Click **Next**.

5. Optional: On the **Customization** page, configure advanced settings by navigating the following tabs:

- **Details**: Configure the **Hostname**, **Headless mode**, **Guest system log access**, and **Deletion protection**.
- **Storage**: Add or change storage disks.
- **Network**: Add network interfaces.
- **Scheduling**: Configure scheduling requirements and the run strategy.
- **SSH**: Configure SSH key settings.
- **Initial run**: Define **Cloud-init** or **Sysprep** initialization scripts.
- **Labels and annotations**: Add labels to help organize and categorize VMs, and use annotations to store extra metadata.

a. Click **Next**.

6. On the **Review and create** page, review the VM configuration:

a. Optional: Select the **Start this VirtualMachine after creation** checkbox.

b. Click **Create VirtualMachine**.

Verification

- Navigate to **Virtualization → VirtualMachines** and verify that the VM is displayed in the list.

6.1.4. Clone an existing VM by using the web console

You can use the **Clone existing VirtualMachine** workflow in the web console to create a new virtual machine (VM) by cloning an existing VM and its configuration.

Use this workflow when:

- You want to duplicate an existing VM with the same configuration.
- You need multiple VMs with a consistent setup.

Prerequisites

- You have access to the OpenShift Dedicated web console.
- You have **edit** or **admin** permissions in the target project.
- The source VM that you want to clone exists in a project that you can access.

Procedure

1. In the OpenShift Dedicated web console, navigate to **Virtualization → VirtualMachines**.
2. Click **Create**.
3. On the **Deployment details** page, configure the following settings:
   a. Select **Clone existing VirtualMachine**.
   b. Review the **Project** and **Group** under **Location**. To change the target project or group, click the edit icon.
   c. Click **Next**.
4. On the **Source** page, select the VM to clone:
   a. Browse the list of available VMs, or use the search and filter options to locate a specific VM. You can filter by **Status**, **Operating system**, **Storage class**, **Hardware devices**, **Scheduling**, **Node**, **Guest agent**, and **Architecture type**.
   b. Select the VM that you want to clone.
   c. Click **Next**.
5. On the **Review and create** page, review the VM configuration:
   a. Optional: In the **Name** field, enter a name for the VM. You can also click the refresh icon to generate a name automatically.
   b. Optional: Enter a **Description** for the VM.
   c. Review the **Location** to verify the target cluster and project. If the folder preview feature is enabled, you can also verify the folder. To change the location, click the edit icon.
   d. Optional: The **Start this VirtualMachine after creation** checkbox is selected by default. Clear the checkbox if you do not want the cloned VM to start immediately.
   e. Click **Clone VirtualMachine**.

Verification

- Navigate to **Virtualization → VirtualMachines** and verify that the VM is displayed in the list.

6.1.5. Virtual machine creation wizard reference

The virtual machine (VM) creation wizard in the web console presents several options that affect how a VM runs. The following tables describe the options that have the greatest impact.

6.1.5.1. Boot source options

On the **Boot source** page, you can select an existing boot volume or click **Add volume** to create one. When you add a volume, the **Source type** list determines where the boot disk image comes from.

Table 6.2. Boot source types

Source type	Description	When to use
Volume (upload new)	Uploads a persistent volume claim (PVC) image or ISO file from your local machine.	You have a disk image or ISO on your workstation. Select **This is an ISO file** for installation media.
Volume (use existing)	Reuses a volume that is already available on the cluster.	A boot volume is already prepared in the cluster.
Volume snapshot	Creates a boot disk from an existing volume snapshot.	You want fast provisioning from a point-in-time image.
Registry	Imports a container disk from a container registry.	The image is published to a registry such as Quay. Requires a container image URL and, for private registries, credentials.
URL	Imports an image from an HTTP or HTTPS endpoint.	The image is hosted at a web address. Select **TLS certificate required** for endpoints that need a certificate.

When you add a volume, the following settings affect how the disk is stored and identified:

Preference

The preferred VM attribute values required to run a given workload, such as the guest operating system. The preference influences default values elsewhere in the wizard.

Default InstanceType

The default instance type associated with the boot volume.

StorageClass

The storage class used to provision the volume. The default storage class is used unless you select another.

Volume Mode

Determines whether the volume is presented as a raw block device (**Block**) or a formatted file system (**Filesystem**).

Access Mode

Controls how many nodes can mount the volume: shared access (**RWX**), single-user (**RWO**), or read-only (**ROX**). Live migration requires shared access (**RWX**).

6.1.5.2. Instance type series

On the **Compute resources** page, you select an instance type series and then a size. The series determines the CPU and memory characteristics of the VM. Some series require specially configured nodes.

Table 6.3. Instance type series

Series	Name	Characteristics	Node requirement
U	General Purpose	Neutral, general-purpose resources. VMs share physical CPU cores with other VMs on a time-slice basis.	None
O	Overcommitted	Based on the U series, but memory is overcommitted.	None
CX	Compute Exclusive	Exclusive compute resources for compute-intensive workloads, with dedicated CPUs, isolated emulator threads, and virtual non-uniform memory access (NUMA) topology.	CPU manager enabled and huge pages available on the nodes.
M	Memory Intensive	Resources for memory-intensive workloads, with burstable CPU performance.	None
D	Dedicated vCPU	Consistent, predictable performance. VMs are assigned exclusive physical CPU cores, which avoids CPU sharing and time-slice contention.	None
N	Network	Resources for network-intensive Data Plane Development Kit (DPDK) workloads, such as virtual network functions (VNFs).	Nodes capable of running DPDK workloads and labeled with **node-role.kubevirt.io/worker-dpdk**.
RT	Realtime	Resources for realtime workloads, such as **oslat**.	Nodes capable of running realtime workloads.

Instance types are named in the format **series.size**, where the size ranges from **nano** to **8xlarge**. For example, **u1.medium** provides 1 vCPU and 4 GiB of memory. You can select a **Red Hat provided** instance type or a **User provided** instance type that an administrator has defined.

6.1.5.3. Customization settings

The optional **Customization** step groups advanced settings into tabs. The following settings are the ones that most commonly affect VM behavior.

Table 6.4. Key customization settings

Setting	Tab	Description
Disk interface	Storage	Determines disk performance and compatibility. **VirtIO** offers the best performance but requires additional drivers on Windows. **SATA** has lower performance but is supported by most operating systems, including Windows. **SCSI** supports large numbers of devices.
Ephemeral disk	Storage	Adds a disk from a container image whose changes are lost when the VM reboots.
Use this disk as a boot source	Storage	Marks a disk as bootable. Only one disk can be bootable at a time.
Headless mode	Details	Removes the default graphics device. VNC console access is not available when this option is enabled.
Deletion protection	Details	Prevents the VM from being deleted through the web console.
Boot mode	Details	Sets the firmware interface, such as BIOS or UEFI, used to boot the VM.
Run strategy	Scheduling	Controls how the VM behaves after a failure, shutdown, or restart, such as **Rerun on failure**.
Eviction strategy	Scheduling	Determines what happens to the VM when its node is drained. **LiveMigrate** moves the VM to another node instead of stopping it.
Network binding	Network	Sets how the network interface connects. **Masquerade** is the default for pod networking; **Bridge** connects to an L2 network; **SR-IOV** attaches a virtual function for high performance.
Initialization method	Initial run	Configures first-boot settings by using **cloud-init** for Linux guests or **Sysprep** for Windows guests.
Dynamic SSH key injection	SSH	Injects a public SSH key that is applied without restarting the VM. When it is not enabled, keys are applied through **virtctl**.
Labels and annotations	Metadata	**Labels** are key-value pairs that you can query to organize and select VMs. **Annotations** store arbitrary metadata that cannot be queried.

6.1.6. Additional resources

- Red Hat VirtIO drivers download page
- How to check virtio-win drivers version on Windows guest

- Installing and updating VirtIO drivers for Windows virtual machines

6.2. CREATING VIRTUAL MACHINES BY USING THE CLI

You can create virtual machines (VMs) from the command line by editing or creating a **VirtualMachine** manifest. You can also create a VM from an operating system image that you import, upload, build into a container disk, or clone from an existing persistent volume claim (PVC).



NOTE

You can also create VMs from instance types by using the OpenShift Dedicated web console.



IMPORTANT

You must install the QEMU guest agent on VMs created from operating system images that are not provided by Red Hat.

6.2.1. Create a VM from a manifest

You can create a virtual machine (VM) from a **VirtualMachine** manifest. To simplify the creation of these manifests, you can use the **virtctl** command-line tool.

Prerequisites

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

Procedure

1. Create a **VirtualMachine** manifest for your VM and save it as a YAML file. For example, to create a minimal Red Hat Enterprise Linux (RHEL) VM, run the following command:

```
$ virtctl create vm --name rhel-9-minimal --volume-import type:ds,src:openshift-virtualization-os-images/rhel9
```

2. Review the **VirtualMachine** manifest for your VM:



NOTE

This example manifest does not configure VM authentication.

Example manifest for a RHEL VM

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: rhel-9-minimal
spec:
  dataVolumeTemplates:
  - metadata:
```

```
    name: imported-volume-mk4lj
  spec:
    sourceRef:
      kind: DataSource
      name: rhel9
      namespace: openshift-virtualization-os-images
    storage:
      resources: {}
instancetype:
  inferFromVolume: imported-volume-mk4lj
  inferFromVolumeFailurePolicy: Ignore
preference:
  inferFromVolume: imported-volume-mk4lj
  inferFromVolumeFailurePolicy: Ignore
runStrategy: Always
template:
  spec:
    domain:
      devices:
        video:
          type: virtio
      memory:
        guest: 512Mi
      resources: {}
    terminationGracePeriodSeconds: 180
    volumes:
    - dataVolume:
        name: imported-volume-mk4lj
      name: imported-volume-mk4lj
```

- **name: rhel-9-minimal** specifies the name of the VM.
- **name: rhel9** specifies the boot source for the guest operating system in the **sourceRef** section.
- **namespace: openshift-virtualization-os-images** specifies the namespace for the boot source. Golden images are stored in the **openshift-virtualization-os-images** namespace.
- **instancetype: inferFromVolume: imported-volume-mk4lj** specifies the instance type inferred from the selected **DataSource** object.
- **preference: inferFromVolume: imported-volume-mk4lj** specifies that the preference is inferred from the selected **DataSource** object.
- **type: virtio** specifies the use of a custom video device (a VirtIO device in this example) to enable hardware graphics acceleration. Enabling a custom video device is in Technology Preview for OpenShift Virtualization 4.22.

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

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

4. Optional: Start the virtual machine:

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

6.2.2. Create a VM from a web-hosted image

You can create a virtual machine (VM) from an image on a web page by using the command line.

When the VM is created, the data volume with the image is imported into persistent storage.

Prerequisites

- You must have access credentials for the web page that contains the image.
- You have installed the **virtctl** CLI.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **VirtualMachine** manifest for your VM and save it as a YAML file. For example, to create a minimal Red Hat Enterprise Linux (RHEL) VM from an image on a web page, run the following command:

```
$ virtctl create vm --name vm-rhel-9 --instancetype u1.small --preference rhel.9 --volume-import type:http,url:https://example.com/rhel9.qcow2,size:10Gi
```

2. Review the **VirtualMachine** manifest for your VM:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-rhel-9
spec:
  dataVolumeTemplates:
  - metadata:
      name: imported-volume-6dcpf
    spec:
      source:
        http:
          url: https://example.com/rhel9.qcow2
      storage:
        resources:
          requests:
            storage: 10Gi
  instancetype:
    name: u1.small
  preference:
    name: rhel.9
  runStrategy: Always
  template:
    spec:
      domain:
        devices: {}
        resources: {}
      terminationGracePeriodSeconds: 180
      volumes:
```

```
    - dataVolume:
        name: imported-volume-6dcpf
      name: imported-volume-6dcpf
```

- **metadata.name** defines the VM name.
- **spec.dataVolumeTemplates.metadata.name** defines the data volume name.
- **spec.dataVolumeTemplates.spec.source.http.url** defines the URL of the image.
- **spec.dataVolumeTemplates.spec.storage.resources.requests.storage** defines the size of the storage requested for the data volume.
- **spec.instancetype.name** defines the instance type to use to control resource sizing of the VM.
- **spec.preference.name** defines the preference to use.

3. Create the VM by running the following command:

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

The **oc create** command creates the data volume and the VM. The CDI controller creates an underlying PVC with the correct annotation and the import process begins. When the import is complete, the data volume status changes to **Succeeded**. You can start the VM.

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

Verification

1. The importer pod downloads the image from the specified URL and stores it on the provisioned persistent volume. View the status of the importer pod:

```
$ oc get pods
```

2. Monitor the status of the data volume:

```
$ oc get dv <data_volume_name>
```

If the provisioning is successful, the data volume phase is **Succeeded**.

Example output:

```
NAME                    PHASE       PROGRESS   RESTARTS   AGE
imported-volume-6dcpf   Succeeded   100.0%                18s
```

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

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

If the VM is running and the serial console is accessible, the output looks as follows:

```
Successfully connected to vm-rhel-9 console. The escape sequence is ^]
```

-

6.2.3. Generalize a VM image

You can generalize a Red Hat Enterprise Linux (RHEL) image to remove all system-specific configuration data before you use the image to create a boot source image, a preconfigured snapshot of a virtual machine (VM). You can use a boot source image to deploy new VMs.

You can generalize a RHEL VM by using the **virtctl**, **guestfs**, and **virt-sysprep** tools.

Prerequisites

- You have a RHEL virtual machine (VM) to use as a base VM.
- You have installed the OpenShift CLI (**oc**).
- You have installed the **virtctl** tool.

Procedure

1. Stop the RHEL VM if it is running, by entering the following command:

```
$ virtctl stop <my_vm_name>
```

2. Optional: Clone the virtual machine to avoid losing the data from your original VM. You can then generalize the cloned VM.
3. Retrieve the **dataVolume** that stores the root filesystem for the VM by running the following command:

```
$ oc get vm <my_vm_name> -o jsonpath="{.spec.template.spec.volumes}{'\n'}"
```

   Example output:

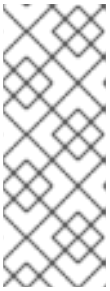
```
[{"dataVolume":{"name":"<my_vm_volume>"},"name":"rootdisk"},{"cloudInitNoCloud":{...}]
```

4. Retrieve the persistent volume claim (PVC) that matches the listed **dataVolume** by running the followimg command:

```
$ oc get pvc
```

   Example output:

```
NAME          STATUS   VOLUME  CAPACITY   ACCESS MODES  STORAGECLASS
AGE
<my_vm_volume> Bound  …
```



NOTE

If your cluster configuration does not enable you to clone a VM, to avoid losing the data from your original VM, you can clone the VM PVC to a data volume instead. You can then use the cloned PVC to create a boot source image.

If you are creating a boot source image by cloning a PVC, continue with the next steps, using the cloned PVC.

5. Deploy a new interactive container with **libguestfs-tools** and attach the PVC to it by running the following command:

```
$ virtctl guestfs <my-vm-volume> --uid 107
```

   This command opens a shell for you to run the next command.

6. Remove all configurations specific to your system by running the following command:

```
$ virt-sysprep -a disk.img
```

7. In the OpenShift Dedicated console, click **Virtualization → Catalog**.
8. Click **Add volume**.
9. In the **Add volume** window:
   a. From the **Source type** list, select **Use existing Volume**.
   b. From the **Volume project** list, select your project.
   c. From the **Volume name** list, select the correct PVC.
   d. In the **Volume name** field, enter a name for the new boot source image.
   e. From the **Preference** list, select the RHEL version you are using.
   f. From the **Default Instance Type** list, select the instance type with the correct CPU and memory requirements for the version of RHEL you selected previously.
   g. Heterogeneous clusters only: From the **Architecture** list, select the architecture that corresponds with the selected volume.
   h. Click **Save**.

Result

The new volume appears in the **Select volume to boot from** list. This is your new boot source image. You can use this volume to create new VMs.

6.2.4. Create a Windows VM from an uploaded image

You can create a Windows virtual machine (VM) by uploading a Windows image to a persistent volume claim (PVC) and then cloning the PVC when you create a VM by using the OpenShift Dedicated web console.



IMPORTANT

You must install VirtIO drivers on Windows VMs. Download VirtIO drivers only from official Red Hat sources.

Prerequisites

- You created a Windows installation DVD or USB with the Windows Media Creation Tool. See Create Windows 10 installation media in the Microsoft documentation.
- You created an **autounattend.xml** answer file. See Answer files (unattend.xml) in the Microsoft documentation.

Procedure

1. Upload the Windows image as a new PVC:
   a. Navigate to **Storage → PersistentVolumeClaims** in the web console.
   b. Click **Create PersistentVolumeClaim → With Data upload form**.
   c. Browse to the Windows image and select it.
   d. Enter the PVC name, select the storage class and size and then click **Upload**.
   The Windows image is uploaded to a PVC.
2. Configure a new VM by cloning the uploaded PVC:
   a. Navigate to **Virtualization → Catalog**.
   b. Select a Windows template tile and click **Customize VirtualMachine**.
   c. Select **Clone (clone PVC)** from the **Disk source** list.
   d. Select the PVC project, the Windows image PVC, and the disk size.
3. Apply the answer file to the VM:
   a. Click **Customize VirtualMachine parameters**.
   b. On the **Sysprep** section of the **Scripts** tab, click **Edit**.
   c. Browse to the **autounattend.xml** answer file and click **Save**.
4. Set the run strategy of the VM:
   a. Clear **Start this VirtualMachine after creation** so that the VM does not start immediately.
   b. Click **Create VirtualMachine**.
   c. On the **YAML** tab, replace **running:false** with **runStrategy: RerunOnFailure** and click **Save**.
5. Click the Options menu ⋮ and select **Control → Start**.
   The VM boots from the **sysprep** disk containing the **autounattend.xml** answer file.

6.2.4.1. Generalize a Windows VM image

You can generalize a Windows operating system image to remove all system-specific configuration data before you use the image to create a new virtual machine (VM).

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

Prerequisites

- A running Windows VM with the QEMU guest agent installed.

Procedure

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

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

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

Result

You can now specialize the VM.

6.2.4.2. Specialize a Windows VM image

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

Prerequisites

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

Procedure

1. In the OpenShift Dedicated console, click **Virtualization → Catalog**.
2. Select a Windows template and click **Customize VirtualMachine**.

3. Select **PVC (clone PVC)** from the **Disk source** list.
4. Select the PVC project and PVC name of the generalized Windows image.
5. Click **Customize VirtualMachine parameters**.
6. Click the **Scripts** tab.
7. In the **Sysprep** section, click **Edit**, browse to the **unattend.xml** answer file, and click **Save**.
8. Click **Create VirtualMachine**.

Result

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

6.2.5. Upload a VM image by using the CLI

You can upload an operating system image by using the **virtctl** command-line tool. You can use an existing data volume or create a new data volume for the image.

Prerequisites

- You must have an **ISO**, **IMG**, or **QCOW2** operating system image file.
- For best performance, compress the image file by using the virt-sparsify tool or the **xz** or **gzip** utilities.
- The client machine must be configured to trust the OpenShift Dedicated router's certificate.
- You have installed the **virtctl** CLI.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Upload the image by running the **virtctl image-upload** command:

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

<datavolume_name>

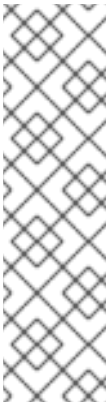
The name of the data volume.

<datavolume_size>

The size of the data volume. For example: **--size=500Mi**, **--size=1G**

</path/to/image>

The file path of the image.



NOTE

- If you do not want to create a new data volume, omit the **--size** parameter and include the **--no-create** flag.
- When uploading a disk image to a PVC, the PVC size must be larger than the size of the uncompressed virtual disk.
- To allow insecure server connections when using HTTPS, use the **--insecure** parameter. When you use the **--insecure** flag, the authenticity of the upload endpoint is **not** verified.

2. Optional. To verify that a data volume was created, view all data volumes by running the following command:

```
$ oc get dvs
```

6.2.6. Prepare a container disk for VMs

You can build a virtual machine (VM) image into a container disk and upload it to a registry.

The size of a container disk is limited by the maximum layer size of the registry where the container disk is hosted.

You create a VM from a container disk by performing the following steps:

1. Build an operating system image into a container disk and upload it to your container registry.
2. If your container registry does not have TLS, configure your environment to disable TLS for your registry.
3. Create a VM with the container disk as the disk source by using the OpenShift Dedicated web console or the command line.



IMPORTANT

If the container disks are large, the I/O traffic might increase and cause worker nodes to be unavailable. You can prune **DeploymentConfig** objects to resolve this issue.



NOTE

For Red Hat Quay, you can change the maximum layer size by editing the YAML configuration file that is created when Red Hat Quay is first deployed.

Prerequisites

- You must have **podman** installed.
- You must have a QCOW2 or RAW image file.

Procedure

1. Create a Dockerfile to build the VM image into a container image. The VM image must be owned by QEMU, which has a UID of **107**, and placed in the **/disk/** directory inside the container. Permissions for the **/disk/** directory must then be set to **0440**.
   The following example uses the Red Hat Universal Base Image (UBI) to handle these configuration changes in the first stage, and uses the minimal **scratch** image in the second stage to store the result:

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

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

   where:

   <vm_image>

   Specifies the image in either QCOW2 or RAW format. If you use a remote image, replace **<vm_image>.qcow2** with the complete URL.

2. Build and tag the container:

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

3. Push the container image to the registry:

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

6.2.7. Disable TLS for an insecure registry

You can disable TLS (transport layer security) for one or more container registries by editing the **insecureRegistries** field of the **HyperConverged** custom resource.

Prerequisites

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

Procedure

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

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

2. Add a list of insecure registries to the **spec.storageImport.insecureRegistries** field.
   Example **HyperConverged** custom resource:

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

```
spec:
  storageImport:
    insecureRegistries:
      - "private-registry-example-1:5000"
      - "private-registry-example-2:5000"
```

Replace the examples in the **insecureRegistries** list with valid registry hostnames.

6.2.8. Create a VM from a container disk by using the CLI

You can create a virtual machine (VM) from a container disk by using the command line.

Prerequisites

- You must have access credentials for the container registry that contains the container disk.
- You have installed the **virtctl** CLI.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **VirtualMachine** manifest for your VM and save it as a YAML file. For example, to create a minimal Red Hat Enterprise Linux (RHEL) VM from a container disk, run the following command:

```
$ virtctl create vm --name vm-rhel-9 --instancetype u1.small --preference rhel.9 --volume-containerdisk src:registry.redhat.io/rhel9/rhel-guest-image:9.5
```

2. Review the **VirtualMachine** manifest for your VM:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-rhel-9
spec:
  instancetype:
    name: u1.small
  preference:
    name: rhel.9
  runStrategy: Always
  template:
    metadata:
      creationTimestamp: null
    spec:
      domain:
        devices: {}
        resources: {}
      terminationGracePeriodSeconds: 180
      volumes:
      - containerDisk:
          image: registry.redhat.io/rhel9/rhel-guest-image:9.5
        name: vm-rhel-9-containerdisk-0
```

- **metadata.name** defines the VM name.
- **spec.instancetype.name** defines the instance type to use to control resource sizing of the VM.
- **spec.preference.name** defines the preference to use.
- **spec.template.spec.volumes.containerDisk.image** defines the URL of the container disk.

3. Create the VM by running the following command:

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

Verification

1. Monitor the status of the VM:

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

If the provisioning is successful, the VM status is **Running**. Example output:

```
NAME       AGE   STATUS    READY
vm-rhel-9   18s   Running   True
```

2. Verify that provisioning is complete and that the VM has started by accessing its serial console:

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

If the VM is running and the serial console is accessible, the output looks as follows:

```
Successfully connected to vm-rhel-9 console. The escape sequence is ^]
```

6.2.9. VM cloning methods

When cloning a data volume, the Containerized Data Importer (CDI) chooses one of the Container Storage Interface (CSI) clone methods: CSI volume cloning or smart cloning. Both methods are efficient but have certain requirements. If the requirements are not met, the CDI uses host-assisted cloning.

Host-assisted cloning is the slowest and least efficient method of cloning, but it has fewer requirements than either of the other two cloning methods.

6.2.9.1. CSI volume cloning

Container Storage Interface (CSI) cloning uses CSI driver features to more efficiently clone a source data volume.

CSI volume cloning has the following requirements:

- The CSI driver that backs the storage class of the persistent volume claim (PVC) must support volume cloning.
- For provisioners not recognized by the CDI, the corresponding storage profile must have the **cloneStrategy** set to CSI Volume Cloning.

- The source and target PVCs must have the same storage class and volume mode.
- If you create the data volume, you must have permission to create the **datavolumes/source** resource in the source namespace.
- The source volume must not be in use.

6.2.9.2. Smart cloning

When a Container Storage Interface (CSI) plugin with snapshot capabilities is available, the Containerized Data Importer (CDI) creates a persistent volume claim (PVC) from a snapshot, which then allows efficient cloning of additional PVCs.

Smart cloning has the following requirements:

- A snapshot class associated with the storage class must exist.
- The source and target PVCs must have the same storage class and volume mode.
- If you create the data volume, you must have permission to create the **datavolumes/source** resource in the source namespace.
- The source volume must not be in use.

6.2.9.3. Host-assisted cloning

When the requirements for neither Container Storage Interface (CSI) volume cloning nor smart cloning have been met, host-assisted cloning is used as a fallback method. Host-assisted cloning is less efficient than either of the two other cloning methods.

Host-assisted cloning uses a source pod and a target pod to copy data from the source volume to the target volume. The target persistent volume claim (PVC) is annotated with the fallback reason that explains why host-assisted cloning has been used, and an event is created.

Example PVC target annotation:

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

Example event:

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

6.2.10. VM creation from a cloned PVC

You can create a virtual machine (VM) by cloning the persistent volume claim (PVC) of an existing VM by using the command line.

You can clone a PVC by using one of the following options:

- Cloning a PVC to a new data volume.
  This method creates a data volume whose lifecycle is independent of the original VM. Deleting the original VM does not affect the new data volume or its associated PVC.
- Cloning a PVC by creating a **VirtualMachine** manifest with a **dataVolumeTemplates** stanza.
  This method creates a data volume whose lifecycle is dependent on the original VM. Deleting the original VM deletes the cloned data volume and its associated PVC.

6.2.10.1. Optimize clone performance at scale in OpenShift Data Foundation

When you use OpenShift Data Foundation, the storage profile configures the default cloning strategy as **csi-clone**. However, this method has limitations, as shown in the following link.

After a certain number of clones are created from a persistent volume claim (PVC), a background flattening process begins, which can significantly reduce clone creation performance at scale.

To improve performance when creating hundreds of clones from a single source PVC, use the **VolumeSnapshot** cloning method instead of the default **csi-clone** strategy.

Procedure

1. Create a **VolumeSnapshot** custom resource (CR) of the source image by using the following content:

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshot
metadata:
  name: golden-volumesnapshot
  namespace: golden-ns
spec:
  volumeSnapshotClassName: ocs-storagecluster-rbdplugin-snapclass
  source:
    persistentVolumeClaimName: golden-snap-source
```

2. Add the **spec.source.snapshot** stanza to reference the **VolumeSnapshot** as the source for the **DataVolume clone**:

```
spec:
  source:
    snapshot:
      namespace: golden-ns
      name: golden-volumesnapshot
```

6.2.10.2. Clone a PVC to a data volume

You can clone the persistent volume claim (PVC) of an existing virtual machine (VM) disk to a data volume by using the command line.

You create a data volume that references the original source PVC. The lifecycle of the new data volume is independent of the original VM. Deleting the original VM does not affect the new data volume or its associated PVC.

Cloning between different volume modes is supported for host-assisted cloning, such as cloning from a block persistent volume (PV) to a file system PV, as long as the source and target PVs belong to the **kubevirt** content type.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- The VM with the source PVC must be powered down.
- If you clone a PVC to a different namespace, you must have permissions to create resources in the target namespace.
- Additional prerequisites for smart-cloning:
  - Your storage provider must support snapshots.
  - The source and target PVCs must have the same storage provider and volume mode.
  - The value of the **driver** key of the **VolumeSnapshotClass** object must match the value of the **provisioner** key of the **StorageClass** object as shown in the following example:
    Example **VolumeSnapshotClass** object:

```
kind: VolumeSnapshotClass
apiVersion: snapshot.storage.k8s.io/v1
driver: openshift-storage.rbd.csi.ceph.com
# ...
```

    Example **StorageClass** object:

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

Procedure

1. Create a **DataVolume** manifest as shown in the following example:

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

   where:

   <datavolume>

   Specifies the name of the new data volume.

<source_namespace>

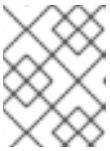
Specifies the namespace of the source PVC.

<my_vm_disk>

Specifies the name of the source PVC.

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

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



NOTE

Data volumes prevent a VM from starting before the PVC is prepared. You can create a VM that references the new data volume while the PVC is being cloned.

6.2.10.3. Create a VM from a cloned PVC data volume template

You can create a virtual machine (VM) that clones the persistent volume claim (PVC) of an existing VM by using a data volume template. This method creates a data volume whose lifecycle is independent on the original VM.

Prerequisites

- The VM with the source PVC must be powered down.
- You have installed the **virtctl** CLI.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **VirtualMachine** manifest for your VM and save it as a YAML file, for example:

```
$ virtctl create vm --name rhel-9-clone --volume-import type:pvc,src:my-project/imported-volume-q5pr9
```

2. Review the **VirtualMachine** manifest for your VM:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: rhel-9-clone
spec:
  dataVolumeTemplates:
  - metadata:
      name: imported-volume-h4qn8
    spec:
      source:
        pvc:
          name: imported-volume-q5pr9
          namespace: my-project
      storage:
        resources: {}
  instancetype:
```

```
    inferFromVolume: imported-volume-h4qn8
    inferFromVolumeFailurePolicy: Ignore
  preference:
    inferFromVolume: imported-volume-h4qn8
    inferFromVolumeFailurePolicy: Ignore
  runStrategy: Always
  template:
    spec:
      domain:
        devices: {}
        memory:
          guest: 512Mi
        resources: {}
      terminationGracePeriodSeconds: 180
      volumes:
      - dataVolume:
          name: imported-volume-h4qn8
        name: imported-volume-h4qn8
```

- **metadata.name** defines the VM name.
- **spec.dataVolumeTemplates.spec.source.pvc.name** defines the name of the source PVC.
- **spec.dataVolumeTemplates.spec.source.pvc.namespace** defines the namespace of the source PVC.
- **spec.instancetype.inferFromVolume** defines that if the PVC source has appropriate labels, the instance type is inferred from the selected **DataSource** object.
- **spec.preference.inferFromVolume** defines that if the PVC source has appropriate labels, the preference is inferred from the selected **DataSource** object.

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

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

6.2.11. Supported custom video device types

When creating a virtual machine (VM), you can configure a custom video device type to override the default video configuration.

Configuring a custom video device allows you to specify different video devices, based on your guest operating system requirements and performance needs.



IMPORTANT

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

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

Using a custom video device provides several advantages:

Performance

Certain video devices provide better performance than the default configuration. For example, VirtIO is a more efficient video device on AMD/x86_64 architecture than legacy VGA.

Resolution flexibility

With some video device types, you can set custom display resolutions.

Memory efficiency

Some video types are more memory efficient for headless or console-only operations.

You can configure the following video device types:

- VirtIO: provides improved performance, and hardware-accelerated video decoding and encoding by offloading tasks to the host. Recommended for modern guest operating systems with available **VirtIO** drivers.
- VGA: the standard for analog video display (default on AMD/x86_64 with BIOS).
- Bochs: an emulated graphics adapter that provides a simple interface for guest operating systems to manage display settings (default on AMD/x86_64 with EFI).
- Cirrus: a legacy video device that provides stable video output.
- ramfb: a simple, unaccelerated virtual display device primarily used in the QEMU emulator, and useful for ARM architecture.

Table 6.5. Video device support by architecture

Architecture	Boot mode	Default type	Supported types
AMD/x86_64	BIOS	**vga**	**virtio**, **vga**, **bochs**, **cirrus**, **ramfb**
AMD/x86_64	EFI	**bochs**	**virtio**, **vga**, **bochs**, **cirrus**, **ramfb**
ARM64	BIOS/EFI	**virtio**	**virtio**, **ramfb**
s390x	BIOS/EFI	**virtio**	**virtio**

6.2.12. Additional resources

- Instance types
- Installing the QEMU guest agent
- Installing VirtIO drivers on Windows VMs
- Red Hat VirtIO drivers download page
- How to check virtio-win drivers version on Windows guest

- Installing and updating VirtIO drivers for Windows virtual machines
- Sysprep (Generalize) a Windows installation
- Configuration pass of Windows Setup (generalize)
- Configuration pass of Windows Setup (specialize)
- Cloning a VM by using the web console
- Managing automatic boot source updates
- Setting a default cloning strategy using a storage profile
- Volume cloning

6.3. INSTANCE TYPES

You can simplify virtual machine (VM) creation by using instance types, which define a reusable set of resources, such as CPU and memory. You can create instance types or change the VMs that use them.

6.3.1. Instance types

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

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

OpenShift Virtualization provides two CRDs for configuring instance types:

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

These objects use the same **VirtualMachineInstancetypeSpec**.

6.3.1.1. Required attributes

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



NOTE

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

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

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

```
apiVersion: instancetype.kubevirt.io/v1beta1
```

```
kind: VirtualMachineInstancetype
metadata:
  name: example-instancetype
spec:
  cpu:
    guest: 1
  memory:
    guest: 128Mi
```

- **spec.cpu.guest** is a required field that specifies the number of vCPUs to allocate to the guest.
- **spec.memory.guest** is a required field that specifies an amount of memory to allocate to the guest.

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

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

where:

--cpu <value>

Specifies the number of vCPUs to allocate to the guest. Required.

--memory <value>

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

TIP

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

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

6.3.1.2. Optional attributes

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

annotations

List annotations to apply to the VM.

gpus

List vGPUs for passthrough.

hostDevices

List host devices for passthrough.

ioThreadsPolicy

Define an IO threads policy for managing dedicated disk access.

launchSecurity

Configure Secure Encrypted Virtualization (SEV).

nodeSelector

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

schedulerName

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

6.3.1.3. Controller revisions

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

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

6.3.2. Specify instance types and preferences with flags

You can specify instance types and preferences by using flags.

Prerequisites

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

Procedure

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

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

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

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

6.3.3. Infer instance types from volumes

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

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

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

Prerequisites

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

Procedure

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

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

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

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

6.3.4. Set volume inference labels

Use the following labels on your PVC, data source, or data volume to instruct the inference mechanism which instance type, preference, or both to use when trying to boot from a volume.

- A cluster-wide instance type: **instancetype.kubevirt.io/default-instancetype** label.
- A namespaced instance type: **instancetype.kubevirt.io/default-instancetype-kind** label. Defaults to the **VirtualMachineClusterInstancetype** label if left empty.
- A cluster-wide preference: **instancetype.kubevirt.io/default-preference** label.
- A namespaced preference: **instancetype.kubevirt.io/default-preference-kind** label. Defaults to **VirtualMachineClusterPreference** label, if left empty.

Prerequisites

- You must have an instance type, preference, or both on the cluster.
- You have installed the OpenShift CLI (**oc**).

Procedure

- To apply a label to a data source, use **oc label**. The following command applies a label that points to a cluster-wide instance type:

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

6.3.5. Change the instance type for a VM

Cluster administrators and VM owners can change the instance type for existing virtual machines to adjust resources or optimize performance for specific workloads.

Changing the instance type for a VM allows you to adapt to evolving workload requirements without recreating the virtual machine. When a VM's workload increases over time, you can switch to an instance type with more CPU, additional memory, or specific hardware resources to prevent performance bottlenecks and ensure the VM continues to meet demand.

Different instance types are optimized for specific use cases, so switching to a specialized instance type can improve performance for particular workloads. For example, you might transition to a compute-optimized instance type for CPU-intensive applications or to a memory-optimized type for workloads that require larger memory allocations.

You can change the instance type for an existing VM using either the OpenShift Dedicated web console or the OpenShift CLI (**oc**).

6.3.5.1. Change a VM instance type in the web console

You can change the instance type associated with a running virtual machine (VM) by using the web console. The change takes effect immediately.

Prerequisites

- You created the VM by using an instance type.

Procedure

1. In the OpenShift Dedicated web console, click **Virtualization → VirtualMachines**.
2. Select a VM to open the **VirtualMachine details** page.
3. Click the **Configuration** tab.
4. On the **Details** tab, click the instance type text to open the **Edit Instancetype** dialog. For example, click **1 CPU | 2 GiB Memory**
5. Edit the instance type by using the **Series** and **Size** lists.
    a. Select an item from the **Series** list to show the relevant sizes for that series. For example, select **General Purpose**.
    b. Select the new instance type for the VM from the **Size** list. For example, select **medium: 1 CPUs, 4Gi Memory**, which is available in the **General Purpose** series.
6. Click **Save**.

Verification

1. Click the **YAML** tab.
2. Click **Reload**.
3. Review the VM YAML to confirm that the instance type changed.

6.3.5.2. Change a VM instance type from the CLI

To change the instance type of a VM, change the **name** field in the VM spec. This triggers the update logic, which ensures that a new, immutable controller revision snapshot is taken of the new resource configuration.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You created the VM by using an instance type, or have administrator privileges for the VM that you want to modify.

Procedure

1. Stop the VM.
2. Run the following command, and replace **<vm_name>** with the name of your VM, and **<new_instancetype>** with the name of the instance type you want to change to:

```
$ oc patch vm/<vm_name> --type merge -p '{"spec":{"instancetype":{"name": "
<new_instancetype>"}}}'
```

Verification

- Check the controller revision reference in the updated VM **status** field. Run the following command and verify that the revision name is updated in the output:

```
$ oc get vms/<vm_name> -o json | jq .status.instancetypeRef
```

Example output:

```
{
  "controllerRevisionRef": {
    "name": "vm-cirros-csmall-csmall-3e86e367-9cd7-4426-9507-b14c27a08671-2"
  },
  "kind": "VirtualMachineInstancetype",
  "name": "csmall"
}
```

- Optional: Check that the VM instance is running the new configuration defined in the latest controller revision. For example, if you updated the instance type to use 2 vCPUs instead of 1, run the following command and check the output:

```
$ oc get vmi/<vm_name> -o json | jq .spec.domain.cpu
```

Example output that verifies that the revision uses 2 vCPUs:

```
{
  "cores": 1,
  "model": "host-model",
  "sockets": 2,
  "threads": 1
}
```

6.3.6. Additional resources

6.4. MANAGING VM TEMPLATES

You can create, customize, and manage the virtual machine (VM) templates that you use to create VMs. To create a VM from a template, use the web console creation wizard.

6.4.1. VM templates

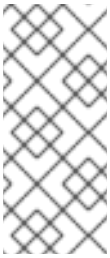
VM templates define a reusable VM configuration. You can create a VM from a template to deploy a standardized VM quickly.

Speed up creation with boot sources

You can speed up VM creation by using templates that have an available boot source. A template with a boot source displays the **Available boot source** label if it does not have a custom label. A template without a boot source displays the **Boot source required** label.

Customize before starting the VM

You can customize the disk source and VM parameters before you start the VM.



NOTE

If you copy a VM template with all its labels and annotations, your version of the template is marked as deprecated when a new version of the Scheduling, Scale, and Performance (SSP) Operator is deployed. You can remove this designation. See "Removing a deprecated designation from a customized VM template by using the web console".

Single-node OpenShift

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

6.4.2. Create a custom VM template in the web console

You can create a virtual machine template by editing a YAML file example in the OpenShift Dedicated web console.

Procedure

1. In the web console, click **Virtualization → Templates** in the side menu.
2. Optional: Use the **Project** drop-down menu to change the project associated with the new template. All templates are saved to the **openshift** project by default.
3. Click **Create Template**.
4. Specify the template parameters by editing the YAML file.
5. Click **Create**.
   The template is displayed on the **Templates** page.
6. Optional: Click **Download** to download and save the YAML file.

6.4.3. Enable dedicated resources for a VM template

You can enable dedicated resources for a virtual machine (VM) template in the OpenShift Dedicated web console. VMs that are created from this template will be scheduled with dedicated resources.

Procedure

1. In the OpenShift Dedicated web console, click **Virtualization → Templates** in the side menu.
2. Select the template that you want to edit to open the **Template details** page.
3. On the **Scheduling** tab, click the edit icon beside **Dedicated Resources**.
4. Select **Schedule this workload with dedicated resources (guaranteed policy)**
5. Click **Save**.

6.4.4. Undeprecate a customized VM template in the web console

You can customize an existing virtual machine (VM) template before you start the VM, by modifying the VM or template parameters, such as data sources, cloud-init, or SSH keys.



NOTE

If you customize a template by copying it and including all of its labels and annotations, the customized template is marked as deprecated when a new version of the Scheduling, Scale, and Performance (SSP) Operator is deployed. You can remove the deprecated designation from the customized template.

Procedure

1. Navigate to **Virtualization → Templates** in the web console.
2. From the list of VM templates, click the template marked as deprecated.
3. Click **Edit** next to the pencil icon beside **Labels**.
4. Remove the following two labels:
   - **template.kubevirt.io/type: "base"**
   - **template.kubevirt.io/version: "version"**
5. Click **Save**.
6. Click the pencil icon beside the number of existing **Annotations**.
7. Remove the following annotation:
   - **template.kubevirt.io/deprecated**
8. Click **Save**.

6.4.5. Additional resources

- Creating a VM from a template by using the web console
- Managing automatic boot source updates

6.5. MANAGING BOOT SOURCES

You can manage the RHEL boot sources that OpenShift Virtualization uses to create virtual machines (VMs), including where they are stored and how they are automatically updated.

RHEL golden images are published as container disks in a secure registry. The Containerized Data Importer (CDI) polls and imports golden images into your cluster and stores them in the **openshift-virtualization-os-images** project as snapshots or persistent volume claims (PVCs).

RHEL images are automatically updated. You can disable and re-enable automatic updates for these images. For more information, see "Additional resources".

Cluster administrators can enable automatic subscription for RHEL virtual machines in the OpenShift Dedicated web console.

For information about creating VMs from these boot sources, see "Additional resources".



IMPORTANT

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

6.5.1. About golden images

A golden image is a preconfigured snapshot of a virtual machine (VM) that you can use as a resource to deploy new VMs. For example, you can use golden images to provision the same system environment consistently and deploy systems more quickly and efficiently.

6.5.1.1. How do golden images work?

Golden images are created by installing and configuring an operating system and software applications on a reference machine or virtual machine. This includes setting up the system, installing required drivers, applying patches and updates, and configuring specific options and preferences.

After the golden image is created, it is saved as a template or image file that can be replicated and deployed across multiple clusters. The golden image can be updated by its maintainer periodically to incorporate necessary software updates and patches, ensuring that the image remains up to date and secure, and newly created VMs are based on this updated image.

6.5.1.2. Red Hat implementation of golden images

Red Hat publishes golden images as container disks in the registry for versions of Red Hat Enterprise Linux (RHEL). Container disks are virtual machine images that are stored as a container image in a container image registry. Any published image will automatically be made available in connected clusters after the installation of OpenShift Virtualization. After the images are available in a cluster, they are ready to use to create VMs.

6.5.2. About VM boot sources

Virtual machines (VMs) consist of a VM definition and one or more disks that are backed by data volumes. VM templates enable you to create VMs using predefined specifications.

Every template requires a boot source, which is a fully configured disk image including configured drivers. Each template contains a VM definition with a pointer to the boot source. Each boot source has

a predefined name and namespace. For some operating systems, a boot source is automatically provided. If it is not provided, then an administrator must prepare a custom boot source.

Provided boot sources are updated automatically to the latest version of the operating system. For auto-updated boot sources, persistent volume claims (PVCs) and volume snapshots are created with the cluster's default storage class. If you select a different default storage class after configuration, you must delete the existing boot sources in the cluster namespace that are configured with the previous default storage class.

6.5.3. Configure a boot source image namespace by using the web console

You can configure a custom namespace for boot source images in your cluster by using the OpenShift Dedicated web console.

Procedure

1. In the web console, select **Virtualization → Settings**.
2. On the **Cluster** tab, select **General settings → Templates and images management**.
3. Click **Bootable volumes project**.
4. Select a namespace to use for boot source images.
   a. If you already created a namespace, select it from the **Project** list.
   b. If you did not create a namespace, scroll to the bottom of the list and click **Create project**.
      i. Enter a name for your new namespace in the **Name** field of the **Create project** dialog.
      ii. Click **Create**.

6.5.4. Configure a boot source image namespace by using the CLI

You can configure a custom namespace for boot source images in your cluster by setting the **spec.commonBootImageNamespace** field in the **HyperConverged** custom resource (CR).

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You created a namespace to use for boot source images.

Procedure

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

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

2. Configure the custom namespace by updating the value of the **spec.commonBootImageNamespace** field.
   Example configuration file:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
```

```
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  commonBootImageNamespace: <custom_namespace>
# ...
```

where:

spec.commonBootImageNamespace

Specifies the namespace to use for boot source images.

3. Save your changes and exit the editor.

6.5.5. Additional resources

- Managing Red Hat boot source updates
- Creating a VM from a template by using the web console
- Creating a VM with custom configuration by using the web console
- Creating a VM from a **VirtualMachine** manifest by using the command line

6.6. IMPORT A VIRTUAL MACHINE IMAGE USING THE DATAVOLUME API

You can import a virtual machine (VM) image into your OpenShift Virtualization cluster by using the **DataVolume** API. A **DataVolume** automates PVC creation and image data transfer from an external source by using the Containerized Data Importer (CDI).

6.6.1. Supported import sources

The **DataVolume** API supports importing VM images from several external sources. The Containerized Data Importer (CDI) detects the image format and converts it as needed.

The following import sources are available:

HTTP or HTTPS

You specify the URL of the image file by using the **source.http** field. CDI downloads the image directly from the web server. You can optionally supply TLS certificates for custom certificate authorities and credentials for Basic authentication.

Container registry

You specify the image location by using the **source.registry** field with a **docker://** URL prefix. CDI pulls the image from the registry by using standard container image pull mechanisms. You can supply registry credentials in a **Secret** object if the registry requires authentication.

S3 object storage

You specify the object URL by using the **source.s3** field. CDI downloads the image from an S3-compatible storage service. You must supply access credentials in a **Secret** object.

oVirt (ImageIO)

You specify the connection details by using the **source.imageio** field. CDI connects to the oVirt ImageIO API to transfer disk images from a Red Hat Virtualization environment.

VMware (VDDK)

You specify the connection details by using the **source.vddk** field. CDI uses the VMware Virtual Disk Development Kit (VDDK) to transfer disk images from a VMware vSphere environment.

Supported image formats include QCOW2, raw, ISO, and compressed archive files.

6.6.2. Import a VM image from a container registry

You can import a virtual machine (VM) image from a container registry into a persistent volume claim (PVC) by using the **DataVolume** API. The Containerized Data Importer (CDI) creates a data volume and an associated PVC to store the imported image.

Prerequisites

- You have access to the container registry.
- If the registry is private, registry credentials are stored in a **Secret** object.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **DataVolume** object that specifies the container registry as the source:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: registry-image-datavolume
spec:
  source:
    registry:
      url: "docker://registry.example.com/my-vm-image:latest"
      pullMethod: node
      secretRef: <registry_secret>
      certConfigMap: <ca_certs_configmap>
  storage:
    resources:
      requests:
        storage: 20Gi
```

where:

url

Specifies the container registry URL with the **docker://** prefix.

pullMethod

Optional: Set to **node** to use the node container runtime cache and pull secrets. The default value is **pod**.

secretRef

Optional: Specifies the name of a **Secret** object that stores the registry credentials.

certConfigMap

Optional: Specifies the name of a **ConfigMap** object that stores custom CA certificates for the registry.

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

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

3. Monitor the import progress by running the following command:

```
$ oc get dv <datavolume_name>
```

Verification

- Confirm that the data volume reaches the **Succeeded** phase by running the following command:

```
$ oc get dv <datavolume_name>
```

Example output:

```
NAME                        PHASE       PROGRESS
registry-image-datavolume   Succeeded   100.0%
```

- Confirm that the PVC is created and bound by running the following command:

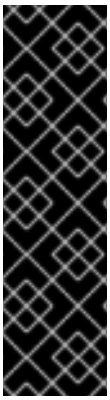
```
$ oc get pvc
```

Example output:

```
NAME                        STATUS   VOLUME     CAPACITY   ACCESS MODES
registry-image-datavolume   Bound    pv-name    20Gi       RWO
```

6.6.3. CPU architecture selection for DataVolume imports

You can target a specific CPU architecture when you import a virtual machine (VM) image from an Open Container Initiative (OCI) image index by setting the **platform.architecture** field in the **DataVolume** custom resource (CR).



IMPORTANT

Specifying CPU architecture for data volume imports is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

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

In heterogeneous clusters where nodes run different CPU architectures, importing the correct image variant is essential. Without specifying the target architecture, the imported image might not match the

architecture of the node where the VM is scheduled to run. The **platform.architecture** field resolves this by selecting the appropriate variant from the image index.

The following CPU architectures are supported:

- **amd64**
- **arm64**
- **s390x**

When you set the **platform.architecture** field in combination with **pullMethod: node**, OpenShift Virtualization adds a node selector to the importer pod. This node selector ensures that the importer pod runs on a node with the matching CPU architecture. The **node** pull method uses the container runtime cache of the node where the importer pod runs, so the pod must run on a node with the correct architecture. Without the node selector, the importer pod might run on a node with a different architecture, resulting in a mismatch between the pulled image and the target platform.

Example **DataVolume** CR:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: my-vm-image
spec:
  source:
    registry:
      url: "docker://registry.example.com/my-multi-arch-image:latest"
      pullMethod: node
      platform:
        architecture: arm64
  storage:
    resources:
      requests:
        storage: 30Gi
```

6.6.4. Import a VM image from an HTTP or HTTPS source

You can import a virtual machine (VM) image from an HTTP or HTTPS source by creating a **DataVolume** object. The Containerized Data Importer (CDI) controller creates a data volume and an underlying persistent volume claim (PVC), then downloads the image from the specified URL into the PVC.

Prerequisites

- A VM image is accessible from an HTTP or HTTPS URL.
- Optional: Custom CA certificates for HTTPS connections are stored in a **ConfigMap** object.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **DataVolume** object that specifies the HTTP or HTTPS source of the VM image:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: http-image-datavolume
spec:
  source:
    http:
      url: "https://example.com/images/my-vm-image.qcow2"
      secretRef: <http_secret>
      certConfigMap: <ca_certs_configmap>
  storage:
    resources:
      requests:
        storage: 20Gi
```

where:

url

Specifies the HTTP or HTTPS URL of the VM image.

secretRef

Optional: Specifies the name of a **Secret** object that stores Basic authentication credentials. The **Secret** must contain **username** and **password** keys.

certConfigMap

Optional: Specifies the name of a **ConfigMap** object that stores custom CA certificates for HTTPS connections.

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

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

3. Monitor the import progress by running the following command:

```
$ oc get dv <datavolume_name>
```

Verification

- Confirm that the data volume reaches the **Succeeded** phase by running the following command:

```
$ oc get dv <datavolume_name>
```

Example output:

```
NAME                    PHASE       PROGRESS
http-image-datavolume   Succeeded   100.0%
```

- Confirm that the PVC is created and bound by running the following command:

```
$ oc get pvc
```

Example output:

```
-
```

```
NAME                    STATUS   VOLUME     CAPACITY   ACCESS MODES
http-image-datavolume   Bound    pv-name    20Gi       RWO
```

6.6.5. Import a VM image from S3 object storage

You can import a virtual machine (VM) image from an S3-compatible object storage bucket by using a **DataVolume** object. The Containerized Data Importer (CDI) pulls the image from the S3 endpoint and populates a persistent volume claim (PVC) that you can use to create a VM.

Prerequisites

- A VM image is stored in an S3-compatible bucket.
- You have S3 access credentials (access key ID and secret key) for your S3-compatible storage account.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **Secret** object for your S3 access credentials:

```
apiVersion: v1
kind: Secret
metadata:
  name: s3-secret
type: Opaque
stringData:
  accessKeyId: "<access_key>"
  secretKey: "<secret_key>"
```

where:

<access_key>

Specifies the access key ID for your S3-compatible storage account.

<secret_key>

Specifies the secret key for your S3-compatible storage account.

2. Apply the **Secret** object:

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

3. Create a **DataVolume** object that specifies the S3 source URL and secret reference:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: s3-image-datavolume
spec:
  source:
    s3:
      url: "https://s3.example.com/my-bucket/my-vm-image.qcow2"
      secretRef: s3-secret
```

```
    certConfigMap: <ca_certs_configmap>
  storage:
    resources:
      requests:
        storage: 20Gi
```

where:

url

Specifies the S3 URL of the VM image. For non-AWS S3 services, specify the full endpoint URL.

secretRef

Specifies the name of the **Secret** with the S3 access credentials.

certConfigMap

Optional: Specifies the name of a **ConfigMap** object that stores custom CA certificates for the S3 endpoint. Required only if your endpoint uses a custom certificate authority.

4. Apply the **DataVolume** object by running the following command:

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

5. Monitor the import progress by running the following command:

```
$ oc get dv <datavolume_name>
```

Verification

- Confirm that the data volume reaches the **Succeeded** phase by running the following command:

```
$ oc get dv <datavolume_name>
```

Example output:

```
NAME                 PHASE       PROGRESS
s3-image-datavolume    Succeeded   100.0%
```

- Confirm that the PVC is created and bound by running the following command:

```
$ oc get pvc
```

Example output:

```
NAME                 STATUS   VOLUME     CAPACITY   ACCESS MODES
s3-image-datavolume    Bound    pv-name    20Gi       RWO
```

6.6.6. DataVolume source fields

You can configure a **DataVolume** object to import a virtual machine (VM) image from several source types. Each source type has its own set of fields that control where and how the Containerized Data Importer (CDI) retrieves the image.

6.6.6.1. Registry source fields

The **source.registry** source type imports a VM image from a container registry.

Table 6.6. **source.registry** fields

Field	Type	Description
url	String	Container registry URL of the image, specified with the **docker://** prefix. For example, **docker://registry.example.com/my-image:latest**.
pullMethod	String	Method for pulling the image. Specify **pod** to use the CDI importer pod or **node** to use the node's container image cache. The default value is **pod**.
secretRef	String	Name of a **Secret** object that stores the credentials for the container registry.
certConfigMap	String	Name of a **ConfigMap** object that stores custom certificate authority (CA) certificates for the container registry.
platform.architecture	String	Target CPU architecture when importing from a multi-architecture image index. Supported values are **amd64**, **arm64**, and **s390x**. This field is a Technology Preview feature.

6.6.6.2. HTTP source fields

The **source.http** source type imports a VM image from an HTTP or HTTPS URL.

Table 6.7. **source.http** fields

Field	Type	Description
url	String	HTTP or HTTPS URL of the VM image.
secretRef	String	Name of a **Secret** object that stores Basic authentication credentials for the HTTP server.
certConfigMap	String	Name of a **ConfigMap** object that stores custom certificate authority (CA) certificates for the HTTP server.
extraHeaders	List of strings	Additional HTTP headers to include in the image download request.

Field	Type	Description
secretExtraHeaders	List of strings	Names of **Secret** objects that store additional HTTP headers to include in the image download request.

6.6.6.3. S3 source fields

The **source.s3** source type imports a VM image from an S3-compatible object storage bucket.

Table 6.8. **source.s3** fields

Field	Type	Description
url	String	S3 URL of the VM image.
secretRef	String	Name of a **Secret** object that stores the S3 access key ID and secret access key.
certConfigMap	String	Name of a **ConfigMap** object that stores custom certificate authority (CA) certificates for the S3 endpoint.

6.6.6.4. ImageIO source fields

The **source.imageio** source type imports a disk image from a Red Hat Virtualization (oVirt) environment by using the ImageIO API.

Table 6.9. **source.imageio** fields

Field	Type	Description
url	String	URL of the oVirt Engine API endpoint. For example, **https://ovirt-engine.example.com/ovirt-engine/api**.
diskId	String	ID of the oVirt disk to import.
secretRef	String	Name of a **Secret** object that stores the credentials to access the oVirt Engine.
certConfigMap	String	Name of a **ConfigMap** object that stores CA certificates for the oVirt Engine.

6.6.6.5. VDDK source fields

The **source.vddk** source type imports a disk image from a VMware vSphere environment by using the Virtual Disk Development Kit (VDDK).

Table 6.10. **source.vddk** fields

Field	Type	Description
url	String	URL of the vCenter or ESXi host.
uuid	String	UUID of the virtual machine in vCenter or ESXi.
backingFile	String	Path to the virtual hard drive to import from vCenter or ESXi.
thumbprint	String	TLS certificate thumbprint of the vCenter or ESXi host.
secretRef	String	Name of a **Secret** object that stores the username and password for the vCenter or ESXi host.
initImageURL	String	URL of a container image with an extracted VDDK library. Overrides the **v2v-vmware** config map.

6.6.6.6. Blank source

The **source.blank** source type creates an empty disk with no data. This source type has no configurable fields.

6.6.6.7. PVC source fields

The **source.pvc** source type clones the contents of an existing persistent volume claim (PVC).

Table 6.11. **source.pvc** fields

Field	Type	Description
namespace	String	Namespace of the source PVC.
name	String	Name of the source PVC.

6.6.6.8. VolumeSnapshot source fields

The **source.snapshot** source type creates a volume from an existing **VolumeSnapshot** object.

Table 6.12. **source.snapshot** fields

Field	Type	Description
namespace	String	Namespace of the source **VolumeSnapshot** object.
name	String	Name of the source **VolumeSnapshot** object.

6.6.6.9. Supported content types

The Containerized Data Importer (CDI) supports the following content types for VM image imports:

- QCOW2
- Raw
- ISO
- Archive (**tar.gz**)

6.6.7. Data volume import troubleshooting

You can diagnose and resolve common issues with data volume imports by inspecting data volume conditions and events. Understanding these conditions helps you identify the root cause of failed or stalled imports.

6.6.7.1. Data volume conditions

Each data volume reports three conditions that indicate its current state. You can view these conditions by running the **oc describe dv** command.

Bound

Indicates whether the persistent volume claim (PVC) has been created and bound to a persistent volume. **Status: True** means the PVC is bound.

Running

Indicates whether an import operation is actively running. **Status: True** means data is being imported.

Ready

Indicates whether the data volume is ready to use. **Status: True** means the import completed successfully.

6.6.7.2. Diagnostic commands

You can use the following commands to gather information about a data volume import:

oc describe dv <datavolume_name>

Shows the data volume conditions, events, and status. Replace **<datavolume_name>** with the name of your data volume.

oc get events --field-selector involvedObject.name=<datavolume_name>

Shows events related to the data volume. Replace **<datavolume_name>** with the name of your data volume.

6.6.7.3. Common failure scenarios

Table 6.13. Common data volume import failures

Symptom	Possible cause	Resolution

Symptom	Possible cause	Resolution
Import pod in **ErrImagePull** or **ImagePullBackOff**	Wrong registry URL or missing credentials	Verify the registry URL and ensure the **secretRef** points to a valid **Secret**.
Import pod in **CrashLoopBackOff**	Invalid image format or corrupted image	Verify the image is in a supported format (QCOW2, raw, ISO, archive).
Data volume stuck in **ImportScheduled**	No available persistent volume matching the storage request	Verify available storage and storage class configuration.
Data volume shows **Scratch space required** error	CDI requires scratch space for image conversion but none is available	Configure a scratch space storage class in the **CDI** object or ensure enough storage is available.
TLS certificate errors	Custom CA not configured	Create a **ConfigMap** object with the CA certificate and reference it in the **certConfigMap** field.
HTTP 401 or 403 errors	Missing or invalid authentication credentials	Verify the **Secret** referenced by **secretRef** has valid credentials.

6.6.8. Additional resources

- Preparing CDI scratch space

CHAPTER 7. MANAGING VMS

7.1. INSTALL THE QEMU GUEST AGENT

The QEMU guest agent is a daemon that runs on the VM and passes information to the host about the VM, users, file systems, and secondary networks. You must install the QEMU guest agent on VMs created from operating system images that are not provided by Red Hat.

7.1.1. Install the QEMU guest agent on a Linux VM

The **qemu-guest-agent** is available by default in Red Hat Enterprise Linux (RHEL) virtual machines (VMs). To create snapshots of a VM in the **Running** state with the highest integrity, install the QEMU guest agent.

The QEMU guest agent takes a consistent snapshot by attempting to quiesce the VM file system. This ensures that in-flight I/O is written to the disk before the snapshot is taken. If the guest agent is not present, quiescing is not possible and a best-effort snapshot is taken.

The conditions under which a snapshot is taken are reflected in the snapshot indications that are displayed in the web console or CLI. If these conditions do not meet your requirements, try creating the snapshot again, or use an offline snapshot

Prerequisites

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

Procedure

1. Log in to the VM by using a console or SSH.
2. Install the QEMU guest agent by running the following command:

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

3. Ensure the service is persistent and start it:

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

Verification

- Run the following command to verify that **AgentConnected** is listed in the VM spec:

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

7.1.2. Install the QEMU guest agent on a Windows VM

For Windows virtual machines (VMs), the QEMU guest agent is included in the VirtIO drivers. You can install the drivers during a Windows installation or on an existing Windows VM.

To create snapshots of a VM in the **Running** state with the highest integrity, install the QEMU guest agent.

The QEMU guest agent takes a consistent snapshot by attempting to quiesce the VM file system. This ensures that in-flight I/O is written to the disk before the snapshot is taken. If the guest agent is not present, quiescing is not possible and a best-effort snapshot is taken.

Note that in a Windows guest operating system, quiescing also requires the Volume Shadow Copy Service (VSS). Therefore, before you create a snapshot, ensure that VSS is enabled on the VM as well.

The conditions under which a snapshot is taken are reflected in the snapshot indications that are displayed in the web console or CLI. If these conditions do not meet your requirements, try creating the snapshot again or use an offline snapshot.

Procedure

1. In the Windows guest operating system, use the **File Explorer** to navigate to the **guest-agent** directory in the **virtio-win** CD drive.
2. Run the **qemu-ga-x86_64.msi** installer.

Verification

1. Obtain a list of network services by running the following command:

```
$ net start
```

2. Verify that the output contains the **QEMU Guest Agent**.

7.2. INSTALL VIRTIO DRIVERS ON WINDOWS VMS

VirtIO drivers are paravirtualized device drivers required for Microsoft Windows virtual machines (VMs) to run in OpenShift Virtualization. The drivers are shipped with the rest of the images. You can also download the **virtio-win** ISO file from the web console to install or update drivers on Windows VMs.

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

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

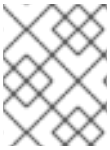
Table 7.1. Supported drivers

Driver name	Hardware ID	Description
viostor	VEN_1AF4&DEV_1001, VEN_1AF4&DEV_1042	The block driver. Sometimes labeled as an **SCSI Controller** in the **Other devices** group. Required for virtual machines using **bus: virtio**.

Driver name	Hardware ID	Description
vioscsi	VEN_1AF4&DEV_1004, VEN_1AF4&DEV_1048	The SCSI pass-through driver. Sometimes labeled as a **SCSI Controller** in the **Other devices** group. Required for virtual machines using **bus: scsi**.
viorng	VEN_1AF4&DEV_1005, VEN_1AF4&DEV_1044	The entropy source driver. Sometimes labeled as a **PCI Device** in the **Other devices** group.
NetKVM	VEN_1AF4&DEV_1000, VEN_1AF4&DEV_1041	The network driver. Sometimes labeled as an **Ethernet Controller** in the **Other devices** group. Available only if a VirtIO NIC is configured.

7.2.1. Download the VirtIO drivers ISO from the web console

You can download the **virtio-win** ISO file from the OpenShift Dedicated web console. The ISO file includes the VirtIO drivers and the QEMU guest agent installer for Microsoft Windows virtual machines (VMs).



NOTE

If your cluster has Windows VMs, a notification on the **Virtualization → VirtualMachines** page provides a link to the **Downloads** tab where you can download the ISO file.

Procedure

1. In the OpenShift Dedicated web console, navigate to **Virtualization → Settings**.
2. Click the **Downloads** tab.
3. In the **Windows drivers** section, click **Download ISO**.
   The **virtio-win** ISO file downloads to your local machine.

7.2.2. Attach a VirtIO container disk during Windows installation

You must attach the VirtIO container disk to the Windows VM to install the necessary Windows drivers. This can be done during creation of the VM.

Procedure

1. When creating a Windows VM from a template, click **Customize VirtualMachine**.
2. Select **Mount Windows drivers disk**.
3. Click the **Customize VirtualMachine parameters**.

4. Click **Create VirtualMachine**.

Result

After the VM is created, the **virtio-win** SATA CD disk will be attached to the VM.

7.2.3. Attach a VirtIO container disk to an existing Windows VM

You must attach the VirtIO container disk to the Windows VM to install the necessary Windows drivers. This can be done to an existing VM.

Procedure

1. Navigate to the existing Windows VM, and click **Actions → Stop**.
2. Go to **VM Details → Configuration → Storage**.
3. Select the **Mount Windows drivers disk** checkbox.
4. Click **Save**.
5. Start the VM, and connect to a graphical console.

7.2.4. Install VirtIO drivers from a container disk SATA CD

You can install VirtIO drivers from a container disk that you add to a Windows virtual machine (VM) as a SATA CD drive.

TIP

You do not need to download the **container-native-virtualization/virtio-win-rhel9** container disk. If the container disk is not present, the cluster downloads it from the Red Hat registry. However, to reduce the installation time, you can download the container disk in advance from the Red Hat Ecosystem Catalog .

You can also download the **virtio-win** ISO file from the OpenShift Dedicated web console by navigating to **Virtualization → Settings → Downloads**.

Prerequisites

- You must have access to the Red Hat registry or to the downloaded **container-native-virtualization/virtio-win** container disk in a restricted environment.
- You have installed the **virtctl** CLI.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Add the **container-native-virtualization/virtio-win** container disk as a CD drive by editing the **VirtualMachine** manifest:

```
# ...
spec:
  domain:
    devices:
```

```
      disks:
        - name: virtiocontainerdisk
          bootOrder: 2
          cdrom:
            bus: sata
volumes:
  - containerDisk:
      image: registry.redhat.io/container-native-virtualization/virtio-win-rhel9:v4.22
    name: virtiocontainerdisk
```

OpenShift Virtualization boots the VM disks in the order defined in the **VirtualMachine** manifest. You can either define other VM disks that boot before the **container-native-virtualization/virtio-win** container disk, or use the optional **bootOrder** parameter to ensure the VM boots from the correct disk. If you configure the boot order for a disk, you must configure the boot order for the other disks.

2. Apply the changes:
   - If the VM is not running, run the following command:

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

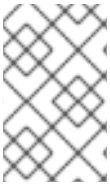
   - If the VM is running, reboot the VM or run the following command:

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

3. After the VM has started, install the VirtIO drivers from the SATA CD drive.

7.2.5. Install VirtIO drivers during Windows setup

You can install the VirtIO drivers while installing Windows on a virtual machine (VM).



NOTE

This procedure uses a generic approach to the Windows installation and the installation method might differ between versions of Windows. See the documentation for the version of Windows that you are installing.

Prerequisites

- A storage device containing the **virtio** drivers must be attached to the VM.

Procedure

1. In the Windows operating system, use the **File Explorer** to navigate to the **virtio-win** CD drive.
2. Double-click the drive to run the appropriate installer for your VM.
   For a 64-bit vCPU, select the **virtio-win-gt-x64** installer. 32-bit vCPUs are no longer supported.
3. Optional: During the **Custom Setup** step of the installer, select the device drivers you want to install. The recommended driver set is selected by default.
4. After the installation is complete, select **Finish**.

5. Reboot the VM.

Verification

1. Open the system disk on the PC. This is typically **C:**.
2. Navigate to **Program Files → Virtio-Win**.

If the **Virtio-Win** directory is present and contains a sub-directory for each driver, the installation was successful.

7.2.6. Install VirtIO drivers from a SATA CD on an existing VM

You can install the VirtIO drivers from a SATA CD drive on an existing Windows virtual machine (VM).



NOTE

This procedure uses a generic approach to adding drivers to Windows. See the installation documentation for your version of Windows for specific installation steps.

Prerequisites

- A storage device containing the virtio drivers must be attached to the VM as a SATA CD drive.

Procedure

1. Start the VM and connect to a graphical console.
2. Log in to a Windows user session.
3. Open **Device Manager** and expand **Other devices** to list any **Unknown device**.
   a. Open the **Device Properties** to identify the unknown device.
   b. Right-click the device and select **Properties**.
   c. Click the **Details** tab and select **Hardware Ids** in the **Property** list.
   d. Compare the **Value** for the **Hardware Ids** with the supported VirtIO drivers.
4. Right-click the device and select **Update Driver Software**.
5. Click **Browse my computer for driver software** and browse to the attached SATA CD drive, where the VirtIO drivers are located. The drivers are arranged hierarchically according to their driver type, operating system, and CPU architecture.
6. Click **Next** to install the driver.
7. Repeat this process for all the necessary VirtIO drivers.
8. After the driver installs, click **Close** to close the window.
9. Reboot the VM to complete the driver installation.

7.2.7. Additional resources

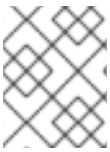
- Update VirtIO drivers

7.3. UPDATE VIRTIO DRIVERS

Update VirtIO drivers and the QEMU guest agent in guest operating systems. Using the latest VirtIO drivers increases performance and stability.

7.3.1. Download the VirtIO drivers ISO from the web console

You can download the **virtio-win** ISO file from the OpenShift Dedicated web console. The ISO file includes the VirtIO drivers and the QEMU guest agent installer for Microsoft Windows virtual machines (VMs).



NOTE

If your cluster has Windows VMs, a notification on the **Virtualization → VirtualMachines** page provides a link to the **Downloads** tab where you can download the ISO file.

Procedure

1. In the OpenShift Dedicated web console, navigate to **Virtualization → Settings**.
2. Click the **Downloads** tab.
3. In the **Windows drivers** section, click **Download ISO**.
   The **virtio-win** ISO file downloads to your local machine.

7.3.2. Update VirtIO drivers and the guest agent on a Windows VM

You can update the VirtIO drivers and the QEMU guest agent on a Windows virtual machine (VM) by using the **virtio-win** guest tools installer. This method updates both the drivers and the guest agent in one step.

Prerequisites

- The **container-native-virtualization/virtio-win** container disk must be attached to the VM as a SATA CD drive. You can mount the disk from the web console by selecting the **Mount Windows drivers disk** checkbox on the **Configuration → Storage** tab.

Procedure

1. Start the VM and connect to a graphical console.
2. Log in to the Windows guest operating system.
3. Open **File Explorer** and navigate to the **virtio-win** CD drive.
4. Double-click the **virtio-win-gt-x64** installer to launch the guest tools setup wizard.
5. Follow the prompts in the setup wizard. The default options update the VirtIO drivers and the QEMU guest agent.
6. After the update is complete, click **Finish**.

7. Reboot the VM.

Verification

1. On the Windows VM, navigate to **Device Manager**.
2. Select a device.
3. Select the **Driver** tab.
4. Click **Driver Details** and confirm that the **virtio** driver details displays the correct version.

7.3.3. Enable automatic updates for Red Hat virtio-win drivers

If the Windows Update service (WUS) is restricted to allow only drivers explicitly signed and published by Microsoft, you must manually configure automatic Red Hat **virtio-win** driver updates. Otherwise, automatic updates are disabled.

Prerequisites

- The cluster must have internet connectivity. Disconnected clusters cannot reach the WUS.

Procedure

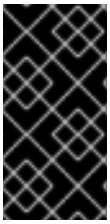
1. Import the Red Hat Release Certificate into the Trusted Publishers store.
   Example command:

   ```
   Import-Certificate -FilePath "redhat-driver-cert.cer" -CertStoreLocation
   Cert:\LocalMachine\TrustedPublisher
   ```

2. In the Group Policy Management Console (GPMC):
   a. Set the **Allow signed updates from an intranet Microsoft update service location** policy to **Enabled**.
      If a driver is signed by a certificate in the Trusted Publishers store, it is now accepted, even if it didn't come from Microsoft directly.
   b. Set the **Do not include drivers with Windows Updates** policy to **Disabled**.

7.3.4. Update VirtIO drivers on a Windows VM

You can update the VirtIO drivers on a Windows virtual machine (VM) by using the Windows Update service (WUS).



IMPORTANT

If you restrict the WUS to only allow drivers explicitly signed and published by Microsoft, automatic Red Hat **virtio-win** driver updates are disabled. For information about enabling automatic Red Hat VirtIO driver updates, see "Enable automatic updates for Red Hat virtio-win drivers".

Prerequisites

- The cluster must have internet connectivity. Disconnected clusters cannot reach the WUS.

Procedure

1. In the Windows Guest operating system, click the **Windows** key and select **Settings**.
2. Navigate to **Windows Update → Advanced Options → Optional Updates**.
3. Install all updates from **Red Hat, Inc.**
4. Reboot the VM.

Verification

1. On the Windows VM, navigate to the **Device Manager**.
2. Select a device.
3. Select the **Driver** tab.
4. Click **Driver Details** and confirm that the **virtio** driver details displays the correct version.

TIP

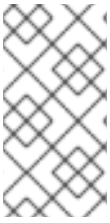
To view the individual driver versions included in the **virtio-win** container disk, open the **release-drivers-versions.txt** file at the root of the **virtio-win** CD drive.

7.3.5. Additional resources

- Allow signed updates from an intranet Microsoft update service location
- Do not include drivers with Windows Updates

7.4. CONNECT TO A VIRTUAL MACHINE CONSOLE

By using VNC, serial, or desktop viewer consoles, you can access the console of your virtual machine for troubleshooting when the VM does not have network connectivity.



NOTE

Connecting to a guest VM through the VNC or serial console does not provide a full set of access features and cannot replace the Virtual Desktop Infrastructure (VDI) access. However, these consoles are useful for troubleshooting, as they allow access even if the guest VM has no network.

7.4.1. VNC console access requirements and limits

Use a VNC connection to troubleshoot a virtual machine (VM) that has lost network connectivity. You can reach the VNC console from the OpenShift Dedicated web console or with the **virtctl** command-line tool, but only one connection to a VM is possible at a time.

Take into account the following considerations:

- Using the VNC console is recommended for troubleshooting VMs.
- Using the VNC console is not recommended for high-traffic applications, such as Virtual Desktop Infrastructure (VDI), because of the burden on the API server.

- The API server must be able to handle the traffic load.
- The clients must be able to access the API server.
- The clients must have access credentials for the cluster.
- The VNC connection is expected to disconnect during live migration of a VM to another node.
- Using the VNC console allows only a single connection per VM at a time.

7.4.2. Connect to the VNC console by using the web console

You can open the VNC console of a virtual machine (VM) from the OpenShift Dedicated web console without installing a local client. If the VM runs Windows with a vGPU assigned as a mediated device, you can switch between the default display and the vGPU display.

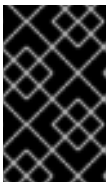


NOTE

If you connect to a Windows VM with a vGPU assigned as a mediated device, you can switch between the default display and the vGPU display.

Procedure

1. On the **Virtualization → VirtualMachines** page, click a VM to open the **VirtualMachine details** page.
2. In the navigation panel, right-click the virtual machine and select **Open Console**.
3. Click the **Console** tab. The VNC console session starts automatically.



IMPORTANT

Only one connection to the VNC console is possible at a time. If you try to create a second connection to the same VNC console, a warning is displayed. You must disconnect the existing session before you create the new session.

4. Optional: To switch to the vGPU display of a Windows VM, select **Ctl + Alt + 2**from the **Send key** list.
   - Select **Ctl + Alt + 1**from the **Send key** list to restore the default display.
5. To end the console session, click outside the console pane and then click **Disconnect**.

7.4.3. Connect to the VNC console by using virtctl

You can use the **virtctl** command-line tool to open the VNC console of a running virtual machine from your terminal, and preserve an existing console session when a new connection starts.



NOTE

If you run the **virtctl vnc** command on a remote machine over an SSH connection, you must forward the X session to your local machine by running the **ssh** command with the **-X** or **-Y** flags.

Prerequisites

- You installed the **virt-viewer** package.

Procedure

1. Run the following command to start the console session:

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

where:

<vm_name>
The name of the VM.

<namespace>
The namespace that contains the VM.

--preserve-session
Prevents an existing VNC console connection from being disconnected if you try to start a new session.



IMPORTANT

Only one connection to the VNC console is possible at a time. If you try to create a second connection to the same VNC console, an error is displayed and the connection fails. If you try to create a second connection to the same VNC console without using the **--preserve-session** flag, this forces the existing connection to disconnect to allow the new connection.

2. If the connection fails, run the following command to collect troubleshooting information:

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

7.4.4. Grant permission to generate VNC console tokens

You can install a cluster role and bind it to a user or a service account to allow access to the endpoint that generates tokens for the VNC console.

Procedure

- Choose to bind the cluster role to either a user or service account.
  - Run the following command to bind the cluster role to a user:

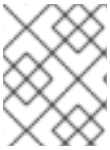
```
$ kubectl create rolebinding "${ROLE_BINDING_NAME}" --
clusterrole="token.kubevirt.io:generate" --user="${USER_NAME}"
```

  - Run the following command to bind the cluster role to a service account:

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

7.4.5. Generate a temporary token for the VNC console

You can generate a temporary authentication bearer token for the Kubernetes API to access the VNC console of a virtual machine (VM). The token expires after a duration that you set, with a minimum of 10 minutes.



NOTE

Kubernetes also supports authentication using client certificates, instead of a bearer token, by modifying the curl command.

Prerequisites

- A running VM with OpenShift Virtualization 4.14 or later and **ssp-operator** 4.14 or later.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Set the **deployVmConsoleProxy** field value in the **HyperConverged** (**HCO**) custom resource (CR) to **true**:

```
$ oc patch hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv --type json -p '[{"op": "replace", "path": "/spec/deployVmConsoleProxy", "value": true}]'
```

2. Generate a token by entering the following command:

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

   You can set the **<duration>** parameter in hours and minutes, with a minimum duration of 10 minutes. For example: **5h30m**. If you do not set this parameter, the token is valid for 10 minutes by default.

   Sample output:

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

3. Optional: Use the token provided in the output to create a variable:

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

   You can now use the token to access the VNC console of a VM.

Verification

1. Log in to the cluster by entering the following command:

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

2. Test access to the VNC console of the VM by using the **virtctl** command:

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



WARNING

You cannot currently revoke a specific token.

To revoke a token, you must delete the service account that you used to create it. However, this also revokes all other tokens that you created by using the service account. Use the following command with caution:

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

7.4.6. Serial console access requirements and limits

Use a serial connection to troubleshoot a virtual machine (VM) that has lost network connectivity. You can reach the serial console from the OpenShift Dedicated web console or with the **virtctl** command-line tool, but only one connection to a VM is possible at a time.

Take into account the following considerations:

- The clients must be able to access the API server.
- The clients must have access credentials for the cluster.
- The API server must be able to handle the traffic load.
- The serial connection is expected to disconnect during live migration of a VM to another node.
- Using the serial console allows only a single connection per VM at a time.

7.4.7. Connect to the serial console by using the web console

You can open the serial console of a virtual machine (VM) from the OpenShift Dedicated web console without installing a local client.

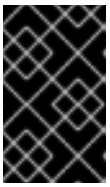


NOTE

Running concurrent VNC connections to a single virtual machine is not currently supported.

Procedure

1. On the **Virtualization → VirtualMachines** page, click a VM to open the **VirtualMachine details** page.
2. In the navigation panel, right-click the virtual machine and select **Open Console**.
3. Click the **Console** tab. The VNC console session starts automatically.



IMPORTANT

Only one connection to the VNC console is possible at a time. If you try to create a second connection to the same VNC console, a warning is displayed. You must disconnect the existing session before you create the new session.

4. Click **Disconnect** to end the VNC console session. Otherwise, the VNC console session continues to run in the background.
5. Select **Serial console** from the console list.
6. Optional: To switch to the vGPU display of a Windows VM, select **Ctl + Alt + 2**from the **Send key** list.
   - Select **Ctl + Alt + 1**from the **Send key** list to restore the default display.
7. To end the console session, click outside the console pane and then click **Disconnect**.

7.4.8. Connect to the serial console by using virtctl

You can use the **virtctl** command-line tool to open the serial console of a running virtual machine from your terminal.



NOTE

If you run the **virtctl vnc** command on a remote machine over an SSH connection, you must forward the X session to your local machine by running the **ssh** command with the **-X** or **-Y** flags.

Prerequisites

- You installed the **virt-viewer** package.

Procedure

1. Run the following command to start the console session:

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

2. Press **Ctrl+]** to end the console session.

7.4.9. Connect to the desktop viewer by using the web console

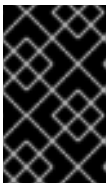
You can connect to the desktop viewer of a Windows virtual machine (VM) from the OpenShift Dedicated web console. The desktop viewer uses RDP, which gives a fuller desktop experience than the VNC console.

Prerequisites

- You installed the QEMU guest agent on the Windows VM.
- You have an RDP client installed.

Procedure

1. On the **Virtualization → VirtualMachines** page, click a VM to open the **VirtualMachine details** page.
2. In the navigation panel, right-click the virtual machine and select **Open Console**.
3. Click the **Console** tab. The VNC console session starts automatically.



IMPORTANT

Only one connection to the VNC console is possible at a time. If you try to create a second connection to the same VNC console, a warning is displayed. You must disconnect the existing session before you create the new session.

4. Click **Disconnect** to end the VNC console session. Otherwise, the VNC console session continues to run in the background.
5. Select **Desktop viewer** from the console list.
6. Click **Create RDP Service** to open the **RDP Service** dialog.
7. Select **Expose RDP Service** and click **Save** to create a node port service.
8. Click **Launch Remote Desktop** to download an **.rdp** file and launch the desktop viewer.
9. Optional: To switch to the vGPU display of a Windows VM, select **Ctl + Alt + 2** from the **Send key** list.
    - Select **Ctl + Alt + 1** from the **Send key** list to restore the default display.
10. To end the console session, click outside the console pane and then click **Disconnect**.

7.4.10. Additional resources

- Scheduling, Scale, and Performance (SSP) Operator
- Connect to the serial console by using the web console
- Connect to the VNC console by using virtctl
- Installing virtctl

7.5. EDIT THE CONFIGURATION OF A VIRTUAL MACHINE

You can update virtual machine (VM) configuration details like CPU, memory, and networking by using the CLI or the OpenShift Dedicated web console. In the web console, you can modify settings on the **VirtualMachine details** page or by editing the YAML file directly.

7.5.1. Change a VM instance type in the web console

You can change the instance type associated with a running virtual machine (VM) by using the web console. The change takes effect immediately.

Prerequisites

- You created the VM by using an instance type.

Procedure

1. In the OpenShift Dedicated web console, click **Virtualization → VirtualMachines**.
2. Select a VM to open the **VirtualMachine details** page.
3. Click the **Configuration** tab.
4. On the **Details** tab, click the instance type text to open the **Edit Instancetype** dialog. For example, click **1 CPU | 2 GiB Memory**
5. Edit the instance type by using the **Series** and **Size** lists.
   a. Select an item from the **Series** list to show the relevant sizes for that series. For example, select **General Purpose**.
   b. Select the new instance type for the VM from the **Size** list. For example, select **medium: 1 CPUs, 4Gi Memory**, which is available in the **General Purpose** series.
6. Click **Save**.

Verification

1. Click the **YAML** tab.
2. Click **Reload**.
3. Review the VM YAML to confirm that the instance type changed.

7.5.2. Change VM memory without a restart

You can add or remove the amount of memory allocated to a virtual machine (VM) without having to restart the VM by using the OpenShift Dedicated web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines**.
2. Select the required VM to open the **VirtualMachine details** page.
3. On the **Configuration** tab, click **Edit CPU|Memory**.
4. Enter the required amount of memory and click **Save**.



NOTE

By hot plugging, you can increase the total amount of memory of a VM up to four times the default initial amount. Exceeding this limit requires a restart.

The system applies these changes immediately. If the VM is able to be migrated, a live migration is triggered. If not, or if the changes cannot be live-updated, a **RestartRequired** condition is added to the VM.



NOTE

Memory hot plugging for virtual machines requires guest operating system support for the **virtio-mem** driver. This support depends on the driver being included and enabled within the guest operating system, not on specific upstream kernel versions.

Supported guest operating systems:

- RHEL 9.4 and later
- RHEL 8.10 and later (hot-unplug is disabled by default)
- Other Linux guests require kernel version 5.16 or later and the **virtio-mem** kernel module
- Windows guests require **virtio-mem** driver version 100.95.104.26200 or later

7.5.3. Change VM CPU sockets without a restart

You can increase or decrease the number of CPU sockets allocated to a virtual machine (VM) without having to restart the VM by using the OpenShift Dedicated web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines**.
2. Select the required VM to open the **VirtualMachine details** page.
3. On the **Configuration** tab, click **Edit CPU|Memory**.
4. Select the **vCPU** radio button.
5. Enter the required number of vCPU sockets and click **Save**.



NOTE

By hot plugging, you can increase the total number of vCPU sockets of a VM up to four times the default initial number. Exceeding this limit requires a restart.

If the VM is migratable, a live migration is triggered. If not, or if the changes cannot be live-updated, a **RestartRequired** condition is added to the VM.



NOTE

If a VM has the **spec.template.spec.domain.devices.networkInterfaceMultiQueue** field enabled and CPUs are hot plugged, the following behavior occurs:

- Existing network interfaces that you attach before the CPU hot plug retain their original queue count, even after you add more virtual CPUs (vCPUs). The underlying virtualization technology causes this expected behavior.
- To update the queue count of existing interfaces to match the new vCPU configuration, you can restart the VM. A restart is only necessary if the update improves performance.
- New VirtIO network interfaces that you hot plugged after the CPU hot plug automatically receive a queue count that matches the updated vCPU configuration.

7.5.3.1. Supported platforms and versions for CPU hot-plug

Different operating systems provide different levels of support for hot-plugging CPUs.

Operating system	Version	Architecture	Hot plug support
Red Hat Enterprise Linux (RHEL)	8.10	x86	Yes
Red Hat Enterprise Linux (RHEL)	9.0 and later	x86	Yes
Red Hat Enterprise Linux (RHEL)	10	x86	Yes
Windows Server 2016	All	x86	Yes (hot unplug not supported)
Windows Server 2019	All	x86	Yes (hot unplug not supported)
Windows Server 2022	All	x86	Yes (hot unplug not supported)
Windows Server 2025	All	x86	Yes (hot unplug not supported)

For information on how to troubleshoot issues with offline hot plugged CPUs, see Hotplugged CPU is offline in the RHEL Guest, unable to bring online.

OpenShift Virtualization may inadverently pass nested virtualization extensions (VMX, SVM) to the guest. Nested Virtualization is not officially supported and generally not recommended, except in specific cases. For more information about this issue, see CNV-68983: Nested Virtualization may get enabled inadvertently if no default cluster cpu model set.

7.5.4. Edit a virtual machine by using the CLI

You can edit a virtual machine (VM) by using the command line.

Prerequisites

- You installed the **oc** CLI.

Procedure

1. Obtain the virtual machine configuration by running the following command:

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

2. Edit the YAML configuration.
3. If you edit a running virtual machine, you need to do one of the following:
   - Restart the virtual machine.
   - Run the following command for the new configuration to take effect:

```
$ oc apply vm <vm_name> -n <namespace>
```

7.5.5. Add a disk to a virtual machine

You can add a virtual disk to a virtual machine (VM) by using the OpenShift Dedicated web console.

Procedure

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



NOTE

If the VM is running, you must restart the VM to apply the change.

7.5.5.1. Virtual machine storage fields

To optimize storage performance and ensure data availability for your workloads, configure the storage fields to define the source, size, and disk characteristics of your virtual machine (VM).

Field	Description
Blank (creates PVC)	Create an empty disk.
Import via URL (creates PVC)	Import content via URL (HTTP or HTTPS endpoint).
Use an existing PVC	Use a PVC that is already available in the cluster.
Clone existing PVC (creates PVC)	Select an existing PVC available in the cluster and clone it.
Import via Registry (creates PVC)	Import content via container registry.
Name	Name of the disk. The name can contain lowercase letters (**a-z**), numbers (**0-9**), hyphens (**-**), and periods (**.**), up to a maximum of 253 characters. The first and last characters must be alphanumeric. The name must not contain uppercase letters, spaces, or special characters.
Size	Size of the disk in GiB.
Type	Type of disk. Example: Disk or CD-ROM
Interface	Type of disk device. Supported interfaces are **virtIO**, **SATA**, and **SCSI**.
Storage Class	The storage class that is used to create the disk.

7.5.5.1.1. Advanced storage settings

The following advanced storage settings are optional and available for **Blank**, **Import via URL**, and **Clone existing PVC** disks.

If you do not specify these parameters, the system uses the default storage profile values.

Parameter	Option	Parameter description
Volume Mode	Filesystem	Stores the virtual disk on a file system-based volume.
	Block	Stores the virtual disk directly on the block volume. Only use **Block** if the underlying storage supports it.
Access Mode	ReadWriteOnce (RWO)	Volume can be mounted as read-write by a single node.

Parameter	Option	Parameter description
	ReadOnlyMany (ROX)	Volume can be mounted as read only by many nodes.

7.5.6. Mount a Windows driver disk on a virtual machine

You can mount a Windows driver disk on a virtual machine (VM) by using the OpenShift Dedicated web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines**.
2. Select the required VM to open the **VirtualMachine details** page.
3. On the **Configuration** tab, click **Storage**.
4. Select the **Mount Windows drivers disk** checkbox.
   The Windows driver disk is displayed in the list of mounted disks.

7.5.7. Add a secret, config map, or service account to a VM

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

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

If the virtual machine is running, changes do not take effect until you restart the virtual machine. The newly added resources are marked as pending changes at the top of the page.

Prerequisites

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

Procedure

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

7. Click **Save**.

Verification

1. On the **VirtualMachine details** page, click **Configuration → Disks** and verify that the resource is displayed in the list of disks.
2. Restart the virtual machine by clicking **Actions → Restart**.

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

7.5.8. Update multiple virtual machines

You can use the command line interface (CLI) to update multiple virtual machines (VMs) at the same time.

Prerequisites

- You installed the OpenShift CLI (**oc**).
- You have access to the OpenShift Dedicated cluster, and you have **cluster-admin** permissions.

Procedure

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

```
$ oc adm new-project kubevirt-api-lifecycle-automation
```

```
$ oc create sa kubevirt-api-lifecycle-automation -n kubevirt-api-lifecycle-automation
```

```
$ oc create clusterrolebinding kubevirt-api-lifecycle-automation --clusterrole=cluster-admin --
serviceaccount=kubevirt-api-lifecycle-automation:kubevirt-api-lifecycle-automation
```

2. Determine the pull URL for the **kubevirt-api-lifecycle** image by running the following command:

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

3. Deploy **Kubevirt-Api-Lifecycle-Automation** by creating a job object as shown in the following example:

```
apiVersion: batch/v1
kind: Job
metadata:
 name: kubevirt-api-lifecycle-automation
 namespace: kubevirt-api-lifecycle-automation
spec:
 template:
  spec:
   containers:
   - name: kubevirt-api-lifecycle-automation
```

```
      image: quay.io/openshift-virtualization/kubevirt-api-lifecycle-automation:v4.22
      imagePullPolicy: Always
      env:
      - name: MACHINE_TYPE_GLOB
        value: smth-glob9.10.0
      - name: RESTART_REQUIRED
        value: "true"
      - name: NAMESPACE
        value: "default"
      - name: LABEL_SELECTOR
        value: my-vm
      securityContext:
       allowPrivilegeEscalation: false
       capabilities:
        drop:
        - ALL
       privileged: false
       runAsNonRoot: true
       seccompProfile:
        type: RuntimeDefault
    restartPolicy: Never
    serviceAccountName: kubevirt-api-lifecycle-automation
```

where:

quay.io/openshift-virtualization/kubevirt-api-lifecycle-automation:v4.22

Specifies the pull URL for your image. Replace the image value in this example with your pull URL for the image.

MACHINE_TYPE_GLOB

Specifies the pattern that is used to detect deprecated machine types that need to be upgraded. Replace the **MACHINE_TYPE_GLOB** value with your own pattern.

RESTART_REQUIRED

Specifies whether VMs should be restarted after the machine type is updated. If the **RESTART_REQUIRED** environment variable is set to **true**, VMs are restarted after the machine type is updated. If you do not want VMs to be restarted, set this value to **false**.

NAMESPACE

Specifies the namespace to look for VMs in. Leave the parameter empty for the job to go over all namespaces in the cluster.

LABEL_SELECTOR

Specifies which VMs receive the job action. If you want the job to go over all VMs in the cluster, do not assign a value to the parameter.

7.5.8.1. Bulk actions on virtual machines

You can perform bulk actions on multiple virtual machines (VMs) simultaneously by using the **VirtualMachines** list view in the web console. This allows you to efficiently manage a group of VMs with minimal manual effort.

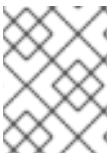
Available bulk actions:

- **Label VMs** - Add, edit, or remove labels that are applied across selected VMs.

- **Delete VMs** - Select multiple VMs to delete. The confirmation dialog displays the number of VMs selected for deletion.
- **Move VMs to folder** - Move selected VMs to a folder. All VMs must belong to the same namespace.
- **LiveMigration** - Perform live migration of multiple selected VMs. The confirmation dialog displays the number of VMs selected for migration. The target node is chosen automatically; there is no option of specifying it.
- **Take snapshot** - Take snapshots of multiple VMs. The **Take snapshots** dialog allows you to enter a suffix for the names of the resulting snapshots.

7.5.9. Configure multiple I/O threads for fast storage access

You can improve storage performance by configuring multiple I/O threads for a virtual machine (VM) that uses fast storage, such as solid-state drive (SSD) or non-volatile memory express (NVMe). This configuration option is only available by editing YAML of the VM.



NOTE

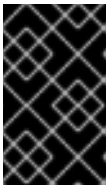
Multiple I/O threads are supported only when **blockMultiQueue** is enabled and the disk bus is set to **virtio**. You must set this configuration for the configuration to work correctly.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **YAML** tab to open the VM manifest.
4. In the YAML editor, locate the **spec.template.spec.domain** section and add or modify the following fields:

```
domain:
  ioThreadsPolicy: supplementalPool
  ioThreads:
    supplementalPoolThreadCount: 4
  devices:
    blockMultiQueue: true
    disks:
    - name: datavolume
      disk:
        bus: virtio
# ...
```

5. Click **Save**.



IMPORTANT

The **spec.template.spec.domain** setting cannot be changed while the VM is running. You must stop the VM before applying the changes, and then restart the VM for the new settings to take effect.

7.5.10. Additional resources

- Understanding config maps
- Providing sensitive data to pods
- Understanding and creating service accounts

7.6. EDIT THE BOOT ORDER OF A VIRTUAL MACHINE

You can configure the boot order of disks and network devices on your virtual machine (VM) by using the web console or the CLI.

With **Boot Order** in the **VirtualMachine details** page, you can:

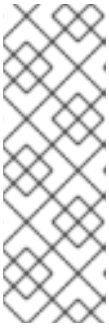
- Select a disk or network interface controller (NIC) and add it to the boot order list.
- Edit the order of the disks or NICs in the boot order list.
- Remove a disk or NIC from the boot order list, and return it back to the inventory of bootable sources.

7.6.1. Add items to a boot order list in the web console

You can add items to a boot order list by using the web console.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. Click the **Virtual machines** tab.
3. Select a virtual machine to open the **VirtualMachine details** page.
4. Click the **Configuration** tab.
5. Expand **Boot management**.
6. Click the pencil icon that is located on the right side of **Boot Order**. If a YAML configuration does not exist, or if this is the first time that you are creating a boot order list, the following message displays: **No resource selected. VM will attempt to boot from disks by order of appearance in YAML file.**
7. Click **Add Source** and select a bootable disk or network interface controller (NIC) for the virtual machine.
8. Add any additional disks or NICs to the boot order list.
9. Click **Save**.



NOTE

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

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

7.6.2. Edit a boot order list in the web console

You can edit the boot order list in the web console.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. Click the **Virtual machines** tab.
3. Select a virtual machine to open the **VirtualMachine details** page.
4. Click the **Configuration** tab.
5. Expand **Boot management**.
6. Click the pencil icon that is located on the right side of **Boot Order**.
7. 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.
8. Click **Save**.



NOTE

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

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

7.6.3. Edit a boot order list by using the CLI

You can edit the boot order list in a YAML configuration file by using the CLI.

Prerequisites

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

Procedure

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

```
$ oc edit vm <vm_name> -n <namespace>
```

2. Edit the YAML file and modify the values for the boot order associated with a disk or network interface controller (NIC). For example:

```
disks:
  - bootOrder: 1
    disk:
      bus: virtio
    name: containerdisk
  - disk:
      bus: virtio
    name: cloudinitdisk
  - cdrom:
      bus: virtio
    name: cd-drive-1
interfaces:
  - boot Order: 2
    macAddress: '02:96:c4:00:00'
    masquerade: {}
    name: default
```

- **disks.bootOrder** defines the boot order value specified for the disk.
- **interfaces.bootOrder** defines the boot order value specified for the network interface controller.

3. Save the YAML file.

7.6.4. Remove items from a boot order list in the web console

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

Procedure

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



NOTE

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

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

7.7. DELETE A VIRTUAL MACHINE

You can remove virtual machines (VMs) from your cluster to free up resources using either the web console or CLI. Deleting a VM removes the virtual machine definition and optionally its associated storage resources.

7.7.1. Delete a VM by using the web console

You can permanently remove a virtual machine (VM) from the cluster by using the OpenShift Dedicated web console. The **Delete** action is disabled while the VM is delete protected.

Prerequisites

- You have disabled the VM's delete protection setting.
- You have stopped the VM.

Procedure

1. From the OpenShift Dedicated web console, choose your view:
   - For a virtualization-focused view, select **Administrator → Virtualization → VirtualMachines**.
   - For a general view, navigate to **Virtualization → VirtualMachines**.
2. Click the **Options** menu  beside a VM and select **Delete**.
   Alternatively, click the VM's name to open the **VirtualMachine details** page and click **Actions → Delete**.

   You can also right-click the VM in the tree view and select **Delete** from the pop-up menu.
3. Optional: Select **With grace period** or clear **Delete disks**.
4. Click **Delete** to permanently delete the VM.

7.7.2. Delete a VM by using the CLI

You can delete a virtual machine (VM) by using the OpenShift CLI (**oc**). The command deletes a VM in the current project unless you specify another namespace.

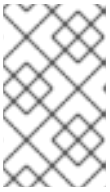
Prerequisites

- You have disabled the VM's delete protection setting.
- You have stopped the VM.
- You have installed the OpenShift CLI (**oc**).

Procedure

- Delete the VM by running the following command:

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



NOTE

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

7.8. EXPORT A VIRTUAL MACHINE

Export a virtual machine (VM) and its associated disks to import it into another cluster, or for another use case, such as forensic volume analysis.

You create a **VirtualMachineExport** custom resource (CR) by using the command-line interface.

Alternatively, you can use the **virtctl vmexport** command to create a **VirtualMachineExport** CR and to download exported volumes.



NOTE

You can migrate virtual machines between OpenShift Virtualization clusters by using the Migration Toolkit for Virtualization.

7.8.1. Create a VirtualMachineExport custom resource

You can create a **VirtualMachineExport** custom resource (CR) to export the persistent volume claims (PVCs) of a virtual machine (VM), a VM snapshot, or a single PVC. The export provides links for downloading the volume.

You can export the following objects:

- VM: Exports the persistent volume claims of a specified VM.
- VM snapshot: Exports PVCs contained in a **VirtualMachineSnapshot** CR.
- PVC: Exports a PVC. If the PVC is used by another pod, such as the **virt-launcher** pod, the export remains in a **Pending** state until the PVC is no longer in use.

The **VirtualMachineExport** CR creates internal and external links for the exported volumes. Internal links are valid within the cluster. External links can be accessed by using an **Ingress** or **Route**.

The export server supports the following file formats:

- **raw**: Raw disk image file.

- **gzip**: Compressed disk image file.
- **dir**: PVC directory and files.
- **tar.gz**: Compressed PVC file.

Prerequisites

- The VM must be shut down for a VM export.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **VirtualMachineExport** manifest to export a volume from a **VirtualMachine**, **VirtualMachineSnapshot**, or **PersistentVolumeClaim** CR according to the following example and save it as **example-export.yaml**.
VirtualMachineExport example:

```
apiVersion: export.kubevirt.io/v1beta1
kind: VirtualMachineExport
metadata:
  name: example-export
spec:
  source:
    apiGroup: "kubevirt.io"
    kind: VirtualMachine
    name: example-vm
  ttlDuration: 1h
```

   - **spec.source.apiGroup** defines the API group of the resource that you want to export:
     - Use **"kubevirt.io"** for **VirtualMachine**.
     - Use **"snapshot.kubevirt.io"** for **VirtualMachineSnapshot**.
     - Use **""** for **PersistentVolumeClaim**.
   - **spec.source.kind** defines the data source for the export. There are three primary values used for this field:
     - **VirtualMachine**
     - **VirtualMachineSnapshot**
     - **PersistentVolumeClaim**
   - **spec.ttlDuration** defines the length of time before the export resource is automatically deleted. The default is 2 hours.

2. Create the **VirtualMachineExport** CR:

```
$ oc create -f example-export.yaml
```

3. Get the **VirtualMachineExport** CR:

```
$ oc get vmexport example-export -o yaml
```

The internal and external links for the exported volumes are displayed in the **status** stanza:

Output example:

```
apiVersion: export.kubevirt.io/v1beta1
kind: VirtualMachineExport
metadata:
  name: example-export
  namespace: example
spec:
  source:
    apiGroup: ""
    kind: PersistentVolumeClaim
    name: example-pvc
  tokenSecretRef: example-token
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2022-06-21T14:10:09Z"
    reason: podReady
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: "2022-06-21T14:09:02Z"
    reason: pvcBound
    status: "True"
    type: PVCReady
  links:
    external:
      cert: |-
        -----BEGIN CERTIFICATE-----
        ...
        -----END CERTIFICATE-----
      volumes:
      - formats:
        - format: raw
          url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1beta1/namespaces/example/virtualmachineexports/examp
le-export/volumes/example-disk/disk.img
        - format: gzip
          url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1beta1/namespaces/example/virtualmachineexports/examp
le-export/volumes/example-disk/disk.img.gz
        name: example-disk
    internal:
      cert: |-
        -----BEGIN CERTIFICATE-----
        ...
        -----END CERTIFICATE-----
      volumes:
      - formats:
        - format: raw
          url: https://virt-export-example-export.example.svc/volumes/example-disk/disk.img
        - format: gzip
```

```
      url: https://virt-export-example-export.example.svc/volumes/example-disk/disk.img.gz
    name: example-disk
phase: Ready
serviceName: virt-export-example-export
```

- **status.links.external** defines external links that are accessible from outside the cluster by using an **Ingress** or **Route**.
- **status.links.internal** defines internal links that are valid only inside the cluster.

7.8.2. Access exported VM manifests

You can retrieve the **VirtualMachine** manifest from the export server after you export a virtual machine (VM) or a VM snapshot. Use the manifest to create the VM on a target cluster.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You exported a virtual machine or VM snapshot by creating a **VirtualMachineExport** custom resource (CR).



NOTE

VirtualMachineExport objects that have the **spec.source.kind: PersistentVolumeClaim** parameter do not generate virtual machine manifests.

Procedure

1. To access the manifests, you must first copy the certificates from the source cluster to the target cluster.
   a. Log in to the source cluster.
   b. Save the certificates to the **cacert.crt** file by running the following command:

   ```
   $ oc get vmexport <export_name> -o jsonpath={.status.links.external.cert} > cacert.crt
   ```

   Replace **<export_name>** with the **metadata.name** value from the **VirtualMachineExport** object.
   c. Copy the **cacert.crt** file to the target cluster.
2. Decode the token in the source cluster and save it to the **token_decode** file by running the following command:

   ```
   $ oc get secret export-token-<export_name> -o jsonpath={.data.token} | base64 --decode > token_decode
   ```

   Replace **<export_name>** with the **metadata.name** value from the **VirtualMachineExport** object.
3. Copy the **token_decode** file to the target cluster.

4. Get the **VirtualMachineExport** custom resource by running the following command:

```
$ oc get vmexport <export_name> -o yaml
```

5. Review the **status.links** stanza, which is divided into **external** and **internal** sections. Note the **manifests.url** fields within each section, for example:

```
apiVersion: export.kubevirt.io/v1beta1
kind: VirtualMachineExport
metadata:
  name: example-export
spec:
  source:
    apiGroup: "kubevirt.io"
    kind: VirtualMachine
    name: example-vm
  tokenSecretRef: example-token
status:
#...
  links:
    external:
#...
      manifests:
      - type: all
        url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1beta1/namespaces/example/virtualmachineexports/examp
le-export/external/manifests/all
      - type: auth-header-secret
        url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1beta1/namespaces/example/virtualmachineexports/examp
le-export/external/manifests/secret
    internal:
#...
      manifests:
      - type: all
        url: https://virt-export-export-pvc.default.svc/internal/manifests/all
      - type: auth-header-secret
        url: https://virt-export-export-pvc.default.svc/internal/manifests/secret
  phase: Ready
  serviceName: virt-export-example-export
```

- **status.links.external.manifests.url** where the **type** is **all** contains the **VirtualMachine** manifest, **DataVolume** manifest, if present, and a **ConfigMap** manifest that contains the public certificate for the external URL's ingress or route.

- **status.links.external.manifests.url** where the **type** is **auth-header-secret** contains a secret containing a header that is compatible with Containerized Data Importer (CDI). The header contains a text version of the export token.

6. Log in to the target cluster.

7. Get the **Secret** manifest by running the following command:

```
$ curl --cacert cacert.crt <secret_manifest_url> -H \
"x-kubevirt-export-token:token_decode" -H \
"Accept:application/yaml"
```

- Replace **<secret_manifest_url>** with an **auth-header-secret** URL from the **VirtualMachineExport** YAML output.
- Reference the **token_decode** file that you created earlier.
  For example:

```
$ curl --cacert cacert.crt https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1beta1/namespaces/example/virtualmachineexports/ex
ample-export/external/manifests/secret -H "x-kubevirt-export-token:token_decode" -H
"Accept:application/yaml"
```

8. Get the manifests of **type: all**, such as the **ConfigMap** and **VirtualMachine** manifests, by running the following command:

```
$ curl --cacert cacert.crt <all_manifest_url> -H \
"x-kubevirt-export-token:token_decode" -H \
"Accept:application/yaml"
```

- Replace **<all_manifest_url>** with a URL from the **VirtualMachineExport** YAML output.
- Reference the **token_decode** file that you created earlier.
  For example:

```
$ curl --cacert cacert.crt https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1beta1/namespaces/example/virtualmachineexports/ex
ample-export/external/manifests/all -H "x-kubevirt-export-token:token_decode" -H
"Accept:application/yaml"
```

Next steps

- You can now create the **ConfigMap** and **VirtualMachine** objects on the target cluster by using the exported manifests.

7.8.3. Additional resources

- Migration Toolkit for Virtualization
- virtctl vmexport command

7.9. MANAGE VIRTUAL MACHINE INSTANCES

Manage standalone virtual machine instances (VMIs) that were created independently outside of the OpenShift Virtualization environment through the web console by using **oc** or **virtctl** commands from the command-line interface (CLI).

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

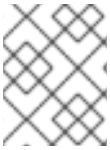
7.9.1. Virtual machine instances

In OpenShift Virtualization, a virtual machine instance (VMI) is an active, powered-on virtual instance, as opposed to a virtual machine, which is the saved configuration. You can create standalone VMIs independently without a parent template, and you can manage them directly.

A standalone VMI is created and started independently with a script, through automation, or by using other methods in the CLI. In your environment, you might have standalone VMIs that were developed and started outside of the OpenShift Virtualization environment. You can continue to manage those standalone VMIs by using the CLI. You can also use the web console for specific tasks associated with standalone VMIs:

- List standalone VMIs and their details.
- Edit labels and annotations for a standalone VMI.
- Delete a standalone VMI.

When you delete a VM, the associated VMI is automatically deleted. You delete a standalone VMI directly because it is not owned by VMs or other objects.



NOTE

Before you uninstall OpenShift Virtualization, list and view the standalone VMIs by using the CLI or the web console. Then, delete any outstanding VMIs.

When you edit a VM, some settings might be applied to the VMIs dynamically and without the need for a restart. Any change made to a VM object that cannot be applied to the VMIs dynamically will trigger the **RestartRequired** VM condition. Changes are effective on the next reboot, and the condition is removed.

7.9.2. List all VMIs by using the CLI

You can use the OpenShift CLI (**oc**) to list all virtual machine instances (VMIs) in your cluster. The list includes standalone VMIs and VMIs that are owned by virtual machines (VMs).

Prerequisites

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

Procedure

- List all VMIs by running the following command:

```
$ oc get vmis -A
```

7.9.3. List standalone VMIs by using the web console

You can list the standalone virtual machine instances (VMIs) in your cluster by using the OpenShift Dedicated web console. The console does not show VMIs that are owned by a virtual machine (VM).



NOTE

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

Procedure

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

7.9.4. Search for standalone VMIs by using the web console

You can search for standalone virtual machine instances (VMIs) from the search bar on the **VirtualMachines** page of the OpenShift Dedicated web console. Advanced search filters narrow the results by name, project, labels, and IP address.

Procedure

1. In the OpenShift Dedicated console, click **Virtualization → VirtualMachines** from the side menu.
2. In the search bar at the top of the page, type a VM name, label, or IP address.
3. In the suggestions list, choose one of the following options:
   - Click a VM name to open its details page.
   - Click **All search results found for ...**to view results on a dedicated page.
   - Click a related suggestion to prefill search filters.
4. Optional: To open advanced search options, click the sliders icon next to the search bar. Expand the **Details** section and specify one or more of the available filters: **Name**, **Project**, **Description**, **Labels**, **Date created**, **vCPU**, and **Memory**.
5. Optional: Expand the **Network** section and enter an IP address to filter by.
6. Click **Search**.
7. Optional: If Advanced Cluster Management (ACM) is installed, use the **Cluster** dropdown to search across multiple clusters.
8. Optional: Click the **Save search** icon to store your search in the **kubevirt-user-settings** ConfigMap.

7.9.5. Edit a standalone VMI by using the web console

You can edit the annotations and labels of a standalone virtual machine instance (VMI) by using the OpenShift Dedicated web console. No other fields are editable.

Procedure

1. In the OpenShift Dedicated console, click **Virtualization → VirtualMachines** from the side menu.

2. Select a standalone VMI to open the **VirtualMachineInstance details** page.
3. On the **Details** tab, click the pencil icon beside **Annotations** or **Labels**.
4. Make the relevant changes and click **Save**.

7.9.6. Delete a standalone VMI by using the CLI

You can delete a standalone virtual machine instance (VMI) by using the OpenShift CLI (**oc**). A standalone VMI has no owning VM, so no other object deletes it for you.

Prerequisites

- Identify the name of the VMI that you want to delete.
- You have installed the OpenShift CLI (**oc**).

Procedure

- Delete the VMI by running the following command:

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

7.9.7. Delete a standalone VMI by using the web console

You can delete a standalone virtual machine instance (VMI) by using the OpenShift Dedicated web console. The deletion is permanent and you cannot recover the VMI.

Procedure

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

7.9.8. Additional resources

- Using the CLI tools

7.10. CONTROL VIRTUAL MACHINE STATES

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

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

7.10.1. Configure RBAC permissions for managing VM states

You can grant users permission to start and stop virtual machines (VMs) from the OpenShift Dedicated web console. Create a role-based access control (RBAC) cluster role and a cluster role binding to apply the permission.

Prerequisites

- You have cluster administrator access to an OpenShift Dedicated cluster where OpenShift Virtualization is installed.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **ClusterRole** object that allows the target user or group to manage VM states:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: vm-manager-access
rules:
  - apiGroups:
      - subresources.kubevirt.io
    resources:
      - virtualmachines/start
      - virtualmachines/stop
    verbs:
      - update
# ...
```

2. Run the following command to apply the cluster role:

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

3. Confirm that the cluster role was created by running the following command and observing the output:

```
$ oc get clusterrole <name>
```

Example output:

```
NAME                AGE
vm-manager-access   15s
```

4. Inspect the details of the cluster role, and ensure the intended rules for **subresources.kubevirt.io** are present, specifically the **virtualmachines/start** and **virtualmachines/stop** subresources.
Run the following command and observe the output:

```
$ oc describe clusterrole <name>
```

Example output:

```
Name:         vm-manager-access
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  ---------  -----------------  --------------  -----
```

```
  virtualmachines/start, virtualmachines/stop with subresources.kubevirt.io group  []  []
[update]
```

5. Create a **ClusterRoleBinding** object to bind the cluster role you have created to the target user or group:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: vm-manager-access-binding
subjects:
  - kind: User
    name: test-user
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: vm-manager-access
  apiGroup: rbac.authorization.k8s.io
```

6. Run the following command to apply the cluster role binding:

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

7. Confirm that the cluster role binding was created by running the following command and observing the output:

```
$ oc get clusterrolebinding <name>
```

Example output:

```
NAME                        AGE
vm-manager-access-binding   15s
```

Verification

1. Check if the user can start a VM by running the following command:

```
$ oc auth can-i update virtualmachines/start --namespace=<namespace> --as=<user_name>
--subresource=subresources.kubevirt.io
```

Example output:

```
yes
```

2. Check if the user can stop a VM by running the following command:

```
$ oc auth can-i update virtualmachines/stop --namespace=<namespace> --as=<user_name>
--group=subresources.kubevirt.io
```

Example output:

```
yes
```

7.10.2. Enable confirmations of VM actions

You can require a confirmation dialog before the **Stop**, **Restart**, and **Pause** actions run on a virtual machine (VM). Confirmation is disabled by default.

Procedure

1. Click **Virtualization → Settings → Cluster → General settings**.
2. Click **VirtualMachine actions confirmation**.
3. Toggle the **VirtualMachine actions confirmation** setting to **On**.

7.10.3. Start a VM

You can start a virtual machine (VM) from the **VirtualMachines** page, the tree view, or the **VirtualMachine details** page of the OpenShift Dedicated web console.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. In the tree view, select the project that contains the VM that you want to start.
3. Navigate to the appropriate menu for your use case:
   - To stay on this page, where you can perform actions on multiple VMs:
     a. Click the Options menu  located at the far right end of the row and click **Control → Start VirtualMachine**.
   - To start the VM from the tree view:
     a. Click the **>** icon next to the project name to open the list of VMs.
     b. Right-click the name of the VM and select **Control → Start**.
   - To view comprehensive information about the selected VM before you start it:
     a. Access the **VirtualMachine details** page by clicking the name of the VM.
     b. Click **Actions → Control → Start**.



NOTE

When you start VM that is provisioned from a **URL** source for the first time, the VM has a status of **Importing** while OpenShift Virtualization imports the container from the URL endpoint. Depending on the size of the image, this process might take several minutes.

7.10.4. Stop a VM

You can stop a virtual machine (VM) from the **VirtualMachines** page, the tree view, or the **VirtualMachine details** page of the OpenShift Dedicated web console.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. In the tree view, select the project that contains the VM that you want to stop.
3. Navigate to the appropriate menu for your use case:
   - To stay on this page, where you can perform actions on multiple VMs:
     a. Click the Options menu  located at the far right end of the row and click **Control → Stop VirtualMachine**.
     b. If action confirmation is enabled, click **Stop** in the confirmation dialog.
   - To stop the VM from the tree view:
     a. Click the > icon next to the project name to open the list of VMs.
     b. Right-click the name of the VM and select **Control → Stop**.
     c. If action confirmation is enabled, click **Stop** in the confirmation dialog.
   - To view comprehensive information about the selected VM before you stop it:
     a. Access the **VirtualMachine details** page by clicking the name of the VM.
     b. Click **Actions → Control → Stop**.
     c. If action confirmation is enabled, click **Stop** in the confirmation dialog.

7.10.5. Restart a VM

You can restart a running virtual machine (VM) from the OpenShift Dedicated web console. The **Restart** action shuts down the VM and starts a new pod for it.



IMPORTANT

The **Restart** action shuts down the VM and starts a new pod. This action removes all related resources including the **virt-launcher** pod and recreates them.

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

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. In the tree view, select the project that contains the VM that you want to restart.
3. Navigate to the appropriate menu for your use case:
   - To stay on this page, where you can perform actions on multiple VMs:

a. Click the Options menu  located at the far right end of the row and click **Control → Restart**.

b. If action confirmation is enabled, click **Restart** in the confirmation dialog.

- To restart the VM from the tree view:

  a. Click the > icon next to the project name to open the list of VMs.

  b. Right-click the name of the VM and select **Control → Restart**.

  c. If action confirmation is enabled, click **Restart** in the confirmation dialog.

- To view comprehensive information about the selected VM before you restart it:

  a. Access the **VirtualMachine details** page by clicking the name of the virtual machine.

  b. Click **Actions → Restart**.

  c. If action confirmation is enabled, click **Restart** in the confirmation dialog.

7.10.6. Reset a VM

You can reset a virtual machine (VM) from the OpenShift Dedicated web console when the VM is unresponsive or has failed to boot. Unlike **Restart**, the **Reset** action keeps the existing pod and hard resets the VM inside it.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. In the tree view, select the project that contains the VM that you want to restart.
3. Navigate to the appropriate menu for your use case:
   - To stay on this page, where you can perform actions on multiple VMs:

     a. Click the Options menu  located at the far right end of the row and click **Control → Reset**.

     b. If action confirmation is enabled, click **Reset** in the confirmation dialog.

   - To reset the VM from the tree view:

     a. Click the > icon next to the project name to open the list of VMs.

     b. Right-click the name of the VM and select **Control → Reset**.

     c. If action confirmation is enabled, click **Reset** in the confirmation dialog.

   - To view comprehensive information about the selected VM before you reset it:

     a. Access the **VirtualMachine details** page by clicking the name of the virtual machine.

     b. Click **Actions → Control → Reset**.

c. If action confirmation is enabled, click **Reset** in the confirmation dialog.

7.10.7. Pause a VM

You can pause a virtual machine (VM) from the **VirtualMachines** page, the tree view, or the **VirtualMachine details** page of the OpenShift Dedicated web console.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. In the tree view, select the project that contains the VM that you want to pause.
3. Navigate to the appropriate menu for your use case:
   - To stay on this page, where you can perform actions on multiple VMs:
     a. Click the Options menu  located at the far right end of the row and click **Control → Pause VirtualMachine**.
     b. If action confirmation is enabled, click **Pause** in the confirmation dialog.
   - To pause the VM from the tree view:
     a. Click the > icon next to the project name to open the list of VMs.
     b. Right-click the name of the VM and select **Control → Pause**.
     c. If action confirmation is enabled, click **Pause** in the confirmation dialog.
   - To view comprehensive information about the selected VM before you pause it:
     a. Access the **VirtualMachine details** page by clicking the name of the VM.
     b. Click **Actions → Control → Pause**.
     c. If action confirmation is enabled, click **Pause** in the confirmation dialog.

7.10.8. Unpause a VM

You can unpause a paused virtual machine (VM) from the **VirtualMachines** page, the tree view, or the **VirtualMachine details** page of the OpenShift Dedicated web console.

Prerequisites

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

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. In the tree view, select the project that contains the VM that you want to unpause.
3. Navigate to the appropriate menu for your use case:

- To stay on this page, where you can perform actions on multiple VMs:

  a. Click the Options menu  located at the far right end of the row and click **Control → Unpause VirtualMachine**.

- To unpause the VM from the tree view:

  a. Click the > icon next to the project name to open the list of VMs.

  b. Right-click the name of the VM and select **Control → Unpause**.

- To view comprehensive information about the selected VM before you unpause it:

  a. Access the **VirtualMachine details** page by clicking the name of the virtual machine.

  b. Click **Actions → Control → Unpause**.

7.10.9. Control the state of multiple VMs

You can start, stop, restart, pause, and unpause several virtual machines (VMs) at once from the OpenShift Dedicated web console. The action applies to every VM in a project or only to the VMs that you select.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Optional: Enable the **Show only projects with VirtualMachines** option above the tree view to limit the displayed projects.
3. Select a relevant project from the tree view.
4. Navigate to the appropriate menu for your use case:

   - To change the state of all VMs in the selected project:

     a. Right-click the name of the project in the tree view and select the intended action from the menu.

     b. If action confirmation is enabled, confirm the action in the confirmation dialog.

   - To change the state of specific VMs:

     a. Select a checkbox next to the VMs you want to work with. To select all VMs, click the checkbox in the **VirtualMachines** table header.

     b. Click **Actions** and select the intended action from the menu.

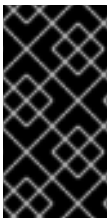
     c. If action confirmation is enabled, confirm the action in the confirmation dialog.

7.10.10. Additional resources

- Using the CLI tools

7.11. USE A VIRTUAL TRUSTED PLATFORM MODULE (VTPM) DEVICE

To run Windows 11 or other workloads that require a Trusted Platform Module, you can add a virtual TPM (vTPM) to a new or existing virtual machine. Enable this by editing the **VirtualMachine** or **VirtualMachineInstance** manifest.



IMPORTANT

With OpenShift Virtualization 4.18 and newer, you can export virtual machines (VMs) with attached vTPM devices, create snapshots of these VMs, and restore VMs from these snapshots. However, cloning a VM with a vTPM device attached to it or creating a new VM from its snapshot is not supported.

7.11.1. vTPM devices and state persistence

A virtual Trusted Platform Module (vTPM) device functions like a physical Trusted Platform Module (TPM) hardware chip. You can use a vTPM device with any operating system, but Windows 11 requires the presence of a TPM chip to install or boot.

A vTPM device allows VMs created from a Windows 11 image to function without a physical TPM chip.

OpenShift Virtualization supports persisting vTPM device state by using Persistent Volume Claims (PVCs) for VMs. If you do not specify the storage class for this PVC, OpenShift Virtualization uses the default storage class for virtualization workloads. If the default storage class for virtualization workloads is not set, OpenShift Virtualization uses the default storage class for the cluster.



NOTE

The storage class that is marked as default for virtualization workloads has the annotation **storageclass.kubevirt.io/is-default-virt-class** set to "true". You can find this storage class by running the following command:

```
$ oc get sc -o jsonpath='{range .items[?
(.metadata.annotations.storageclass\.kubevirt\.io/is-default-virt-class=="true")]}
{.metadata.name}{"\n"}{end}'
```

Similarly, the default storage class for the cluster has the annotation **storageclass.kubernetes.io/is-default-class** set to "true". To find this storage class, run the following command:

```
$ oc get sc -o jsonpath='{range .items[?
(.metadata.annotations.storageclass\.kubernetes\.io/is-default-class=="true")]}
{.metadata.name}{"\n"}{end}'
```

To ensure consistent behavior, configure only one storage class as the default for virtualization workloads and for the cluster respectively.

It is recommended that you specify the storage class explicitly by setting the **vmStateStorageClass** attribute in the **HyperConverged** custom resource (CR):

```
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
```

```
  vmStateStorageClass: <storage_class_name>

# ...
```

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

7.11.2. Add a vTPM device to a virtual machine

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



IMPORTANT

When you add a virtual Trusted Platform Module (vTPM) device to a Windows VM, it is important to make the vTPM device persistent. The BitLocker Drive is encrypted successfully and the encryption system check passes even if the vTPM device is not persistent. If the vTPM device is not persistent, it is discarded on shutdown.

Prerequisites

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

Procedure

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

```
$ oc edit vm <vm_name> -n <namespace>
```

2. Edit the VM specification to add the vTPM device. For example:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
   name: example-vm
spec:
 template:
   spec:
     domain:
       devices:
         tpm: 1
           persistent: true 2
# ...
```

   - **spec.template.spec.domain.devices.tpm** specifies the vTPM device to add to the VM.
   - **spec.template.spec.domain.devices.tpm.persistent** specifies that the vTPM device state persists after the VM is shut down. The default value is **false**.

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

7.11.3. Additional resources

- Exporting virtual machines
- Creating a snapshot by using the CLI
- Restoring a virtual machine from a snapshot by using the CLI

7.12. MANAGE VIRTUAL MACHINES WITH OPENSHIFT PIPELINES

Red Hat OpenShift Pipelines is a Kubernetes-native CI/CD framework that allows developers to design and run each step of the CI/CD pipeline in its own container.

By using OpenShift Pipelines tasks and the example pipeline, you can do the following:

- Create and manage virtual machines (VMs), persistent volume claims (PVCs), data volumes, and data sources.
- Run commands in VMs.
- Manipulate disk images with **libguestfs** tools.

The tasks are located in the task catalog (ArtifactHub).

The example Windows pipeline is located in the pipeline catalog (ArtifactHub).

7.12.1. Prerequisites

- You have access to an OpenShift Dedicated cluster with **cluster-admin** permissions.
- You have installed the OpenShift CLI (**oc**).
- You have installed OpenShift Pipelines.

7.12.2. Supported virtual machine tasks

Automate virtual machine (VM) provisioning and management in your CI/CD workflows with OpenShift Pipelines tasks designed for virtualization. These tasks allow you to manage VMs and their disks as part of your automated deployment pipelines, streamlining VM lifecycle management.

Table 7.2. Supported virtual machine tasks

Task	Description
create-vm-from-manifest	Create a virtual machine from a provided manifest or with **virtctl**.
create-vm-from-template	Create a virtual machine from a template.
copy-template	Copy a virtual machine template.
modify-vm-template	Modify a virtual machine template.

Task	Description
modify-data-object	Create or delete data volumes or data sources.
cleanup-vm	Run a script or a command in a virtual machine and stop or delete the virtual machine afterward.
disk-virt-customize	Use the **virt-customize** tool to run a customization script on a target PVC.
disk-virt-sysprep	Use the **virt-sysprep** tool to run a sysprep script on a target PVC.
wait-for-vmi-status	Wait for a specific status of a virtual machine instance and fail or succeed based on the status.



NOTE

Virtual machine creation in pipelines now utilizes **ClusterInstanceType** and **ClusterPreference** instead of template-based tasks, which have been deprecated. The **create-vm-from-template**, **copy-template**, and **modify-vm-template** commands remain available but are not used in default pipeline tasks.

7.12.3. Windows EFI installer pipeline

You can run the Windows EFI installer pipeline by using the web console or CLI.

The Windows EFI installer pipeline installs Windows 10, Windows 11, or Windows Server 2022 into a new data volume from a Windows installation image (ISO file). A custom answer file is used to run the installation process.



NOTE

The Windows EFI installer pipeline uses a config map file with **sysprep** predefined by OpenShift Dedicated and suitable for Microsoft ISO files. For ISO files pertaining to different Windows editions, it may be necessary to create a new config map file with a system-specific **sysprep** definition.

7.12.3.1. Run the example pipelines in the web console

You can run the example pipelines from the **Pipelines** menu in the web console.

Procedure

1. Click **Pipelines → Pipelines** in the side menu.
2. Select a pipeline to open the **Pipeline details** page.
3. From the **Actions** list, select **Start**. The **Start Pipeline** dialog is displayed.

4. Keep the default values for the parameters and then click **Start** to run the pipeline. The **Details** tab tracks the progress of each task and displays the pipeline status.

7.12.3.2. Run the example pipelines by using the CLI

Use a **PipelineRun** resource to run the example pipelines. A **PipelineRun** object is the running instance of a pipeline. It instantiates a pipeline for execution with specific inputs, outputs, and execution parameters on a cluster. It also creates a **TaskRun** object for each task in the pipeline.

Prerequisites

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

Procedure

1. To run the Microsoft Windows 11 installer pipeline, create the following **PipelineRun** manifest:

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  generateName: windows11-installer-run-
  labels:
    pipelinerun: windows11-installer-run
spec:
    params:
    - name: winImageDownloadURL
      value: <windows_image_download_url>
    - name: acceptEula
      value: false
    pipelineRef:
      params:
      - name: catalog
        value: redhat-pipelines
      - name: type
        value: artifact
      - name: kind
        value: pipeline
      - name: name
        value: windows-efi-installer
      - name: version
        value: 4
      resolver: hub
    taskRunSpecs:
    - pipelineTaskName: modify-windows-iso-file
      PodTemplate:
        securityContext:
          fsGroup: 107
          runAsUser: 107
```

- For **<windows_image_download_url>**, specify the URL for the Windows 11 64-bit ISO file. The product's language must be English (United States).
- Example **PipelineRun** objects have a special parameter, **acceptEula**. By setting this parameter, you are agreeing to the applicable Microsoft user license agreements for each deployment or installation of the Microsoft products. If you set it to false, the pipeline exits

at the first task.

2. Apply the **PipelineRun** manifest:

```
$ oc apply -f windows11-customize-run.yaml
```

7.12.4. Remove deprecated or unused resources

You can clean up deprecated or unused resources associated with the Red Hat OpenShift Pipelines Operator.

Procedure

- Remove any remaining OpenShift Pipelines resources from the cluster by running the following command:

```
$ oc delete clusterroles,rolebindings,serviceaccounts,configmaps,pipelines,tasks \
  --selector 'app.kubernetes.io/managed-by=ssp-operator' \
  --selector 'app.kubernetes.io/component in (tektonPipelines,tektonTasks)' \
  --selector 'app.kubernetes.io/name in (tekton-pipelines,tekton-tasks)' \
  --ignore-not-found \
  --all-namespaces
```

  If the Red Hat OpenShift Pipelines Operator custom resource definitions (CRDs) have already been removed, the command may return an error. You can safely ignore this, as all other matching resources will still be deleted.

7.12.5. Additional resources

- Understanding OpenShift Pipelines
- Task catalog (ArtifactHub)
- Windows EFI installer pipeline (ArtifactHub)
- Installing OpenShift Pipelines
- Creating CI/CD solutions for applications using Red Hat OpenShift Pipelines
- Create a Windows VM from an uploaded image

7.13. ADVANCED VIRTUAL MACHINE MANAGEMENT

7.13.1. Working with resource quotas for virtual machines

Manage and control virtual machine resource consumption to prevent overcommitment and ensure stable workload performance.

Create and manage resource quotas for virtual machines.

7.13.1.1. Set resource quota limits for VMs by using the CLI

OpenShift Virtualization manages CPU and memory limits for virtual machines (VMs) automatically. If a namespace enforces a resource quota that requires limits, you can customize the memory limit ratio or set limits manually so that your VMs comply with the quota.

By default, the memory limit is set to twice the requested memory and the CPU limit is set to one per vCPU.

You can customize the memory limit ratio for a specific namespace by adding the **alpha.kubevirt.io/auto-memory-limits-ratio** label to the namespace. For example, the following command sets the memory limit ratio to 1.2:

```
$ oc label ns/my-virtualization-project alpha.kubevirt.io/auto-memory-limits-ratio=1.2
```



WARNING

Avoid managing resource quota limits manually. To prevent misconfigurations or scheduling issues, rely on the automatic resource limit management provided by OpenShift Virtualization unless you have a specific need to override the defaults.

Resource quotas that only use requests automatically work with VMs. If your resource quota uses limits, you must manually set resource limits on VMs. Memory resource limits, defined by the **spec.template.spec.domain.resources.limits.memory** value, must be at least 500 MiB, or 2% larger than the **spec.template.spec.domain.memory.guest** value.

Procedure

1. Set limits for a VM by editing the **VirtualMachine** manifest. For example:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: with-limits
spec:
  runStrategy: Halted
  template:
    spec:
      domain:
        memory:
          guest: 128Mi
        resources:
          limits:
            memory: 256Mi
```

where

spec.template.spec.domain.memory.guest

Specifies the actual amount of RAM that is shown to the guest operating system in the VM.

spec.template.spec.domain.resources.limits.memory

Specifies the hard limit for total memory consumption by the **virt-launcher** pod that hosts the VM. This limit must account for the guest operating system RAM plus the hypervisor runtime resource consumption.
This example configuration is supported because the **spec.template.spec.domain.resources.limits.memory** value is at least **100Mi** larger than the **spec.template.spec.domain.memory.guest** value.

2. Save the **VirtualMachine** manifest.

7.13.1.2. Accurate VM resource measurement with AAQ

The Application Aware Quota (AAQ) Operator measures the CPU and memory that your virtual machines (VMs) actually use. The **ResourceQuota** object counts the whole VM pod, including the resources that the hypervisor consumes, so it overstates VM consumption.

AAQ provides quota enforcement designed for virtualization workloads:

Kubernetes ResourceQuota

Measures and enforces resource limits at the pod level. When applied to VMs, it measures the entire VM pod's resource consumption, including the resources consumed by the hypervisor and the pod infrastructure, which does not reflect actual VM resource allocation.

Application Aware Quota (AAQ)

Extends ResourceQuota functionality to accurately measure and enforce VM resource consumption. AAQ understands the virtualization layer and can measure actual virtual CPU and memory allocated to VMs, separate from what the pod itself consumes.

AAQ supports two types of quotas:

Project-scoped quotas

Apply resource limits to a specific namespace or project. These are the most common use case for virtualization workloads.

Cluster-scoped quotas

Apply resource limits across the entire cluster. These are useful for advanced scenarios where you need cluster-wide resource enforcement.

When you enable AAQ, you must configure a quota calculation method. This setting determines how AAQ measures VM resource consumption across your cluster:

Virtual resources

Measures only the virtual CPU and memory allocated to virtual machines (VMs), excluding the CPU and memory that the pod runtime consumes. This is the default option for virtualization workloads.

Virtual Machine Instance (VMI) pod usage

Measures the total virtual CPU and memory consumption of the VM pod, including both the virtual machine and the pod runtime.

Dedicated virtual resources

Measures the virtual CPU and memory resources assigned to virtual machines and their associated pods, tracking quota usage separately for each type.

7.13.1.3. Enable the Application Aware Quota Operator

Enable the Application Aware Quota (AAQ) Operator and choose a quota calculation method in the web console. Using the web console avoids editing the **HyperConverged** custom resource (CR) by hand.

Prerequisites

- You have cluster administrator privileges.
- The OpenShift Virtualization operator is installed and running.

Procedure

1. In the OpenShift Dedicated web console, click **Virtualization → Settings**.
2. Under **Resource Management**, toggle the **Application Aware Quotas (AAQ)** to On.
   The operator deployment begins automatically. The status changes from **Disabled** to **Enabled** after the operator pods are running.
3. Optional: Click the **Edit** icon next to the **Quota calculation method** field to change the calculation method.
   A modal dialog is displayed with the following options:
   - **Virtual resources**: Measures only the virtual CPU and memory allocated to virtual machines (VMs), excluding the CPU and memory that the pod runtime consumes. This is the default option for virtualization workloads.
   - **Virtual Machine Instance (VMI) pod usage**: Measures the total virtual CPU and memory consumption of the VM pod, including both the virtual machine and the pod runtime.
   - **Dedicated virtual resources**: Measures the virtual CPU and memory resources assigned to virtual machines and their associated pods, tracking quota usage separately for each type.
4. Select your preferred quota calculation method and click **Save**.

Verification

1. Verify that a new **Quotas** option is displayed in the left navigation menu under **Virtualization**. This indicates that AAQ is ready for quota creation and management.

7.13.1.4. Create VM quotas by using the web console

Use the guided form in the OpenShift Dedicated web console to create an Application Aware Quota (AAQ) that limits virtual machine (VM) resource use in a project. The form is the simplest way to set quota limits without writing YAML.

Prerequisites

- You have cluster administrator privileges.
- The Application Aware Quota Operator is enabled in your cluster. See Enable the Application Aware Quota Operator.
- The project where you want to create the quota exists.

Procedure

1. In the OpenShift Dedicated web console, click **Virtualization → Quotas**.
2. Click **Create quota**.
3. In the dropdown menu, select **With form**.
   The create quota form is displayed.
4. Enter a name for your quota in the **Name** field.
   The name must be unique within the project.
5. Select the project where you want to apply this quota.
   By default, the current project is selected. You can click the dropdown to select a different project if needed.



NOTE

If a Kubernetes **ResourceQuota** already exists in the selected project, an alert is displayed with a link to view the existing ResourceQuota. You can still create the AAQ quota, but both quotas will apply to the project.

6. Configure CPU, memory, and VMI limits for the quota:
   The fields displayed vary based on the selected quota calculation method. For example:
   - For the **Virtual resources** or **Dedicated virtual resources** quota calculation method, the quotas exclude the resources that the pod consumes:
     - Use the **vCPU allocation** field to set the maximum number of virtual CPUs permitted for all VMs in the project, excluding the CPU that infrastructure consumes.
     - Use the **Virtual memory allocation** field to set the maximum memory capacity, in GiB, excluding the memory that infrastructure consumes.
     - Use the **VMI limits** field to set the maximum number of virtual machine instances.
   - For the **Virtual Machine Instance (VMI) pod usage** quota calculation method, the quotas include the resources that the pod consumes:
     - Use the **CPU allocation** field to set the maximum number of CPU cores for pods that run VM workloads
     - Use the **Memory allocation** field to set the maximum memory capacity, in GiB, for pods that run VM workloads.
     - Use the **VMI limits** field to set the maximum number of virtual machine instances.



NOTE

The resources consumed by live migration are excluded from the **Virtual Machine Instance (VMI) pod usage** quota calculation method.

7. Review the configuration and click **Create**.
   The quota is created and you are redirected to the quota details page. The new quota is now active and enforcing resource limits on virtual machines in the selected project.

Verification

1. In the OpenShift Dedicated web console, click **Virtualization → Quotas**.
2. In the quotas list, locate your newly created quota.
3. Verify that the quota name is displayed in the list with the correct project and resource limits configured.
4. Click the quota name to open its details page and confirm all settings are correct.

7.13.1.5. Manage VM quotas by using the list view

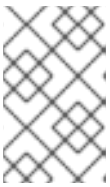
You can use the **Quotas** page in the OpenShift Dedicated web console to view the status and usage of every Application Aware Quota (AAQ) in your cluster, and to create, edit, or delete a quota, without searching through individual projects.

Prerequisites

- You have cluster administrator privileges.
- The Application Aware Quota Operator is enabled in your cluster.
- At least one Application Aware Quota exists in a namespace that matches the label selector configured in the HyperConverged Operator (HCO).

Procedure

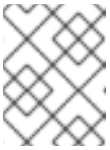
1. In the OpenShift Dedicated web console, click **Virtualization → Quotas**.
   The quotas list view is displayed, showing all project-scoped AAQ quotas by default.
2. Click the **Cluster-scoped quotas** tab to view cluster scoped quotas.
   This tab is displayed only if cluster-scoped quotas have been defined in your cluster.



NOTE

Cluster-scoped quotas can only be created or edited using YAML. The form-based creation option is not available for cluster-scoped quotas due to their complexity.

3. To view detailed information about a specific quota, click the quota name to open the quota details page.
4. To edit a quota, click the Options menu  at the end of the quota row
   a. Select **Edit quota**.
      The edit form is displayed with the current quota configuration.
   b. Click **Save** after you have completed modifying the resource limits.



NOTE

If the quota was initially created with advanced settings via YAML, clicking **Edit quota** opens the YAML editor instead of the form.

5. To delete a quota, click the Options menu  at the end of the quota row and select **Delete quota**. A confirmation dialog is displayed.

    a. Click **Delete** to remove the quota.



WARNING

Deleting a quota removes all resource enforcement for the quota. Virtual machines in the affected project are no longer subject to the quota limits.

6. To create a new quota from the list view, click **Create quota**.

    a. Select an option from the dropdown menu:

    - **With form**: Opens the form-based creation experience for simple VM quota use cases.
    - **With YAML**: Opens the YAML editor for advanced quota configurations that include pod limits or other resource types.

7.13.1.6. Manage VM quota details by using the web console

You can use the **Quota details** page in the OpenShift Dedicated web console to review the resource limits, current usage, and utilization charts for an Application Aware Quota (AAQ), so you can tell how close a project is to its limits.



NOTE

Cluster scoped quotas show only the YAML definition due to the complexity that these quotas require.

Prerequisites

- You have cluster administrator privileges.
- The Application Aware Quota Operator is enabled in your cluster.
- An Application Aware Quota exists in your cluster.

Procedure

1. In the OpenShift Dedicated web console, click **Virtualization → Quotas**.
2. Click the quota name to open the **Quota details** view.
   The **Quota details** view displays visual indicators, quota metadata, and tabulated data:

    - Circular utilization charts visually represent resource consumption in your project

- The **Details** element provides metadata such as the Quota name, scope, applied labels and project.
- The **Quota details** element provides a tabulated views that display the current and maximum utilization of each resource type.



NOTE

If a Kubernetes **ResourceQuota** also exists in the same project as this AAQ quota, an alert is displayed at the top of the details page with a link to view the ResourceQuota. Both quotas apply simultaneously to the project.

3. Optional: Click the **YAML** tab to review the resource configuration of the quota.
4. Optional: Use the **Actions** panel to edit or delete the quota.

7.13.1.7. Additional resources

- Resource quotas per project
- Resource quotas across many projects
- Understanding the Application Aware Quota operator
- Enabling the Application Aware Quota operator
- Creating virtual machine quotas using the web console
- Managing virtual machine quotas using the list view
- Managing virtual machine quota details using the web console

7.13.2. Specifying nodes for virtual machines

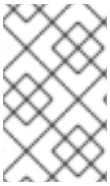
You can place virtual machines (VMs) on specific nodes by using node placement rules.

7.13.2.1. Node placement rules for VMs

Node placement rules control which nodes a virtual machine (VM) can run on. Use them to spread VMs across nodes for fault tolerance, to keep related VMs together, or to place a VM on a node with hardware that the VM requires.

You might want to do this if:

- You have several VMs. To ensure fault tolerance, you want them to run on different nodes.
- You have two chatty VMs. To avoid redundant inter-node routing, you want the VMs to run on the same node.
- Your VMs require specific hardware features that are not present on all available nodes.
- You have a pod that adds capabilities to a node, and you want to place a VM on that node so that it can use those capabilities.



NOTE

Virtual machine placement relies on any existing node placement rules for workloads. If workloads are excluded from specific nodes on the component level, virtual machines cannot be placed on those nodes.

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

nodeSelector

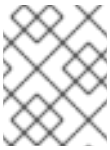
Allows virtual machines to be scheduled on nodes that are labeled with the key-value pair or pairs that you specify in this field. The node must have labels that exactly match all listed pairs.

affinity

Enables you to use more expressive syntax to set rules that match nodes with virtual machines. For example, you can specify that a rule is a preference, rather than a hard requirement, so that virtual machines are still scheduled if the rule is not satisfied. Pod affinity, pod anti-affinity, and node affinity are supported for virtual machine placement. Pod affinity works for virtual machines because the **VirtualMachine** workload type is based on the **Pod** object.

tolerations

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



NOTE

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

7.13.2.2. Node placement examples

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

7.13.2.2.1. VM node placement with nodeSelector

Use a **nodeSelector** rule to run a virtual machine (VM) only on nodes that carry every label you specify.

In this example, the virtual machine requires a node that has metadata containing both **example-key-1 = example-value-1** and **example-key-2 = example-value-2** labels.



WARNING

If there are no nodes that fit this description, the virtual machine is not scheduled.

Example 7.1. Example VM manifest

```
metadata:
  name: example-vm-node-selector
```

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      nodeSelector:
        example-key-1: example-value-1
        example-key-2: example-value-2
# ...
```

7.13.2.2.2. VM node placement with pod affinity and pod anti-affinity

Use pod affinity and pod anti-affinity rules to place a virtual machine (VM) on the same node as specific pods, or away from them.

In this example, the VM must be scheduled on a node that has a running pod with the label **example-key-1 = example-value-1**. If there is no such pod running on any node, the VM is not scheduled.

If possible, the VM is not scheduled on a node that has any pod with the label **example-key-2 = example-value-2**. However, if all candidate nodes have a pod with this label, the scheduler ignores this constraint.

Example 7.2. Example VM manifest

```
metadata:
  name: example-vm-pod-affinity
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: example-key-1
                operator: In
                values:
                - example-value-1
            topologyKey: kubernetes.io/hostname
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: example-key-2
                  operator: In
                  values:
                  - example-value-2
              topologyKey: kubernetes.io/hostname
# ...
```

- If you use the **requiredDuringSchedulingIgnoredDuringExecution** rule type, the VM is not scheduled if the constraint is not met.
- If you use the **preferredDuringSchedulingIgnoredDuringExecution** rule type, the VM is still scheduled if the constraint is not met, provided that all required constraints are met.

7.13.2.2.3. VM node placement with node affinity

Use a node affinity rule to run a virtual machine (VM) on nodes that match the labels you specify, as either a hard requirement or a preference.

In this example, the VM must be scheduled on a node that has the label **example.io/example-key = example-value-1** or the label **example.io/example-key = example-value-2**. The constraint is met if only one of the labels is present on the node. If neither label is present, the VM is not scheduled.

If possible, the scheduler avoids nodes that have the label **example-node-label-key = example-node-label-value**. However, if all candidate nodes have this label, the scheduler ignores this constraint.

Example 7.3. Example VM manifest

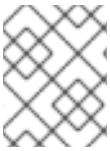
```
metadata:
  name: example-vm-node-affinity
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: example.io/example-key
                operator: In
                values:
                - example-value-1
                - example-value-2
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 1
            preference:
              matchExpressions:
              - key: example-node-label-key
                operator: In
                values:
                - example-node-label-value
# ...
```

- If you use the **requiredDuringSchedulingIgnoredDuringExecution** rule type, the VM is not scheduled if the constraint is not met.
- If you use the **preferredDuringSchedulingIgnoredDuringExecution** rule type, the VM is still scheduled if the constraint is not met, provided that all required constraints are met.

7.13.2.2.4. VM node placement with tolerations

Use tolerations to let a virtual machine (VM) run on nodes that are reserved by a taint, such as nodes set aside for virtualization workloads.

In this example, nodes that are reserved for virtual machines are already tainted with **key=virtualization:NoSchedule**. Because this virtual machine has matching **tolerations**, it can schedule onto the tainted nodes.



NOTE

A virtual machine that tolerates a taint is not required to schedule onto a node with that taint.

Example 7.4. Example VM manifest

```
metadata:
  name: example-vm-tolerations
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  tolerations:
  - key: "key"
    operator: "Equal"
    value: "virtualization"
    effect: "NoSchedule"
# ...
```

7.13.2.3. Additional resources

- Specifying nodes for virtualization components
- Placing pods on specific nodes using node selectors
- Controlling pod placement on nodes using node affinity rules

7.13.3. Using UEFI mode for virtual machines

You can boot a virtual machine (VM) in Unified Extensible Firmware Interface (UEFI) mode for faster boot times, the ability to boot to larger disks, and added security features.

7.13.3.1. About UEFI mode for virtual machines

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

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

7.13.3.2. Boot virtual machines in UEFI mode

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

Prerequisites

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

Procedure

1. To boot a virtual machine (VM) in UEFI mode with secure boot active, edit or create a **VirtualMachine** manifest file. Use the **spec.firmware.bootloader** stanza to configure UEFI mode:

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

   - You must set **spec.template.spec.domain.features.ssm.enabled** to have a value of **true**.
   - If **spec.template.spec.domain.firmware.bootloader.efi.secureBoot** is set to **true**, then UEFI mode is required. However, you can enable UEFI mode without using Secure Boot.

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

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

7.13.3.3. Configure VMs with persistent EFI

You can configure a VM to have EFI persistence enabled by editing its manifest file.

Procedure

- Edit the VM manifest file and save to apply settings.

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm
spec:
  template:
    spec:
      domain:
        firmware:
          bootloader:
            efi:
              persistent: true
# ...
```

7.13.4. Configuring PXE booting for virtual machines

PXE booting, or 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.

7.13.4.1. PXE boot a VM with a specified MAC address

You can boot a virtual machine (VM) over the network by creating a **NetworkAttachmentDefinition** object for your PXE network. Then, reference the **NetworkAttachmentDefinition** in your VM instance configuration file, **VirtualMachine**, before you start the VM instance.

You can also specify a MAC address in the **VirtualMachine** configuration, if required by the PXE server.

Prerequisites

- A Linux bridge must be connected.
- The PXE server must be connected to the same VLAN as the bridge.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Configure a PXE network on the cluster:
   a. Create the network attachment definition file for PXE network **pxe-net-conf**:

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: pxe-net-conf
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "pxe-net-conf",
      "type": "bridge",
      "bridge": "bridge-interface",
```

```
    "macspoofchk": false,
    "vlan": 100,
    "disableContainerInterface": true,
    "preserveDefaultVlan": false
  }
```

- **metadata.name** specifies the name for the **NetworkAttachmentDefinition** object.
- **spec.config.name** specifies the name for the configuration. It is recommended to match the configuration name to the **name** value of the network attachment definition.
- **spec.config.type** specifies the actual name of the Container Network Interface (CNI) plugin that provides the network for this network attachment definition. This example uses a Linux bridge CNI plugin. You can also use an OVN-Kubernetes localnet or an SR-IOV CNI plugin.
- **spec.config.bridge** specifies the name of the Linux bridge configured on the node.
- **spec.config.macspoofchk** is an optional flag to enable the MAC spoof check. When set to **true**, you cannot change the MAC address of the pod or guest interface. This attribute allows only a single MAC address to exit the pod, which provides security against a MAC spoofing attack.
- **spec.config.vlan** is an optional VLAN tag. No additional VLAN configuration is required on the node network configuration policy.
- **spec.config.preserveDefaultVlan** is an optional flag that indicates whether the VM connects to the bridge through the default VLAN. The default value is **true**.

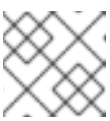
2. Create the network attachment definition by using the file you created in the previous step:

```
$ oc create -f pxe-net-conf.yaml
```

3. Edit the virtual machine instance configuration file to include the details of the interface and network.

   a. Specify the network and MAC address, if required by the PXE server. If the MAC address is not specified, a value is assigned automatically.
   Ensure that **bootOrder** is set to **1** so that the interface boots first. In this example, the interface is connected to a network called **<pxe-net>**:

```
interfaces:
- masquerade: {}
  name: default
- bridge: {}
  name: pxe-net
  macAddress: de:00:00:00:00:de
  bootOrder: 1
```



NOTE

Boot order is global for interfaces and disks.

b. Assign a boot device number to the disk to ensure proper booting after operating system provisioning.
Set the disk **bootOrder** value to **2**:

```
devices:
  disks:
  - disk:
      bus: virtio
    name: containerdisk
    bootOrder: 2
```

c. Specify that the network is connected to the previously created network attachment definition. In this scenario, **<pxe-net>** is connected to the network attachment definition called **<pxe-net-conf>**:

```
networks:
- name: default
  pod: {}
- name: pxe-net
  multus:
    networkName: pxe-net-conf
```

4. Create the virtual machine instance:

```
$ oc create -f vmi-pxe-boot.yaml
```

Example output:

```
virtualmachineinstance.kubevirt.io "vmi-pxe-boot" created
```

5. Wait for the virtual machine instance to run:

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

6. View the virtual machine instance using VNC:

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

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

8. Log in to the virtual machine instance:

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

Verification

1. Verify the interfaces and MAC address on the virtual machine and that the interface connected to the bridge has the specified MAC address. In this case, we used **eth1** for the PXE boot, without an IP address. The other interface, **eth0**, got an IP address from OpenShift Dedicated.

```
$ 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
```

7.13.4.2. OpenShift Virtualization networking glossary

Definitions of key OpenShift Virtualization networking terms and technologies.

Container Network Interface (CNI)

A Cloud Native Computing Foundation project, focused on container network connectivity. OpenShift Virtualization uses CNI plugins to build upon the basic Kubernetes networking functionality.

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.

UserDefinedNetwork

A namespace-scoped CRD introduced by the user-defined network (UDN) API that can be used to create a tenant network that isolates the tenant namespace from other namespaces.

ClusterUserDefinedNetwork

A cluster-scoped CRD introduced by the user-defined network API that cluster administrators can use to create a shared network across multiple namespaces.

7.13.5. Using huge pages with virtual machines

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

7.13.5.1. What huge pages do

To optimize memory mapping efficiency, understand the function of huge pages. Unlike standard 4Ki blocks, huge pages are larger memory segments that reduce the tracking load on the translation lookaside buffer (TLB) hardware cache.

Memory is managed in blocks known as pages. On most systems, a page is 4Ki; 1Mi of memory is equal to 256 pages; 1Gi of memory is 256,000 pages, and so on. CPUs have a built-in memory management unit that manages a list of these pages in hardware. The translation lookaside buffer (TLB) is a small hardware cache of virtual-to-physical page mappings. If the virtual address passed in a hardware instruction can be found in the TLB, the mapping can be determined quickly. If not, a TLB miss occurs, and the system falls back to slower, software-based address translation, resulting in performance issues. Since the size of the TLB is fixed, the only way to reduce the chance of a TLB miss is to increase the page size.

A huge page is a memory page that is larger than 4Ki. On x86_64 architectures, there are two common huge page sizes: 2Mi and 1Gi. Sizes vary on other architectures. To use huge pages, code must be written so that applications are aware of them. Transparent huge pages (THP) attempt to automate the management of huge pages without application knowledge, but they have limitations. In particular, they

are limited to 2Mi page sizes. THP can lead to performance degradation on nodes with high memory utilization or fragmentation because of defragmenting efforts of THP, which can lock memory pages. For this reason, some applications might 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.

7.13.5.2. Map VM memory more efficiently with huge pages

Configure a virtual machine (VM) to use pre-allocated huge pages so that the VM's memory is mapped more efficiently than with standard 4Ki pages.

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 operating system are unrelated. Huge pages requested in the virtual machine manifest apply to QEMU. Huge pages inside the guest can only be configured based on the amount of available memory of the virtual machine instance.

If you edit a running virtual machine, the virtual machine must be rebooted for the changes to take effect.

Prerequisites

- Nodes must have pre-allocated huge pages configured.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. In your virtual machine configuration, add the **resources.requests.memory** and **memory.hugepages.pageSize** parameters to the **spec.domain**. The following configuration snippet is for a virtual machine that requests a total of **4Gi** memory with a page size of **1Gi**:

```
kind: VirtualMachine
# ...
spec:
  domain:
    resources:
      requests:
        memory: "4Gi"
    memory:
      hugepages:
        pageSize: "1Gi"
# ...
```

   - **memory** defines the total amount of memory requested for the virtual machine. This value must be divisible by the page size.
   - **pageSize** defines 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
```

7.13.6. About high availability for virtual machines

You can enable high availability for virtual machines (VMs) by configuring remediating nodes.

You can configure remediating nodes by installing the Self Node Remediation Operator or the Fence Agents Remediation Operator from the software catalog and enabling machine health checks or node remediation checks.

For more information on remediation, fencing, and maintaining nodes, see the "Workload Availability for Red Hat OpenShift" documentation.

7.13.6.1. Additional resources

- Workload Availability for Red Hat OpenShift

7.13.7. Tune the control plane

You can configure how the control plane handles concurrency when you create or migrate virtual machines (VMs). For example, set the **QPS** or **burst** rates to batch create virtual machines (VMs) in a batch, or tune migration settings in the **HyperConverged** custom resource (CR).

7.13.7.1. Configure a highBurst profile

Use the **highBurst** profile to create and maintain a large number of virtual machines (VMs) in one cluster.

Prerequisites

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

Procedure

- Apply the following patch to enable the **highBurst** tuning policy profile:

```
$ oc patch hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv \
  --type=json -p='[{"op": "add", "path": "/spec/tuningPolicy", \
  "value": "highBurst"}]'
```

Verification

- Run the following command to verify the **highBurst** tuning policy profile is enabled:

```
$ oc get kubevirt.kubevirt.io/kubevirt-kubevirt-hyperconverged \
  -n openshift-cnv -o go-template --template='{{range $config, \
  $value := .spec.configuration}} {{if eq $config "apiConfiguration" \
  "webhookConfiguration" "controllerConfiguration" "handlerConfiguration"}} \
  {{"\n"}} {{$config}} = {{$value}} {{end}} {{end}} {{"\n"}}
```

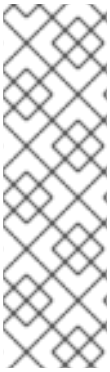
7.14. VM DISKS

7.14.1. Hot-plug virtual disks to running VMs

You can hot-plug or hot-unplug virtual disks from running VMs to dynamically adjust storage without downtime.

Only data volumes and persistent volume claims (PVCs) can be hot plugged and hot-unplugged. You cannot hot plug or hot-unplug container disks.

A hot plugged disk remains attached to the VM even after reboot. You must unplug the disk to remove it from the VM.



NOTE

Each VM has a **virtio-scsi** controller so that hot plugged disks can use the SCSI bus. The **virtio-scsi** controller overcomes the limitations of VirtIO while retaining its performance advantages. It is highly scalable and supports hot plugging over 4 million disks.

When you hot plug disks to the VirtIO (**virtio-blk**) bus, each disk uses a PCI Express (PCIe) slot in the VM. The number of PCIe slots is limited and pre-set automatically at the VM creation as specified in the "Available VirtIO Ports" table. Therefore, you can use **virtio-blk** for a small number of disks that does not exceed the number of available slots.

7.14.1.1. Hot plug and unplug disks by using the web console

You can hot plug a disk by attaching it to a virtual machine (VM) while the VM is running by using the OpenShift Dedicated web console.

The hot plugged disk remains attached to the VM until you unplug it.

Prerequisites

- You must have a data volume or persistent volume claim (PVC) available for hot plugging.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Select a running VM to view its details.
3. On the **VirtualMachine details** page, click **Configuration → Storage**.
4. Add a hot plugged disk:
   a. Click **Add**.
   b. In the **Add disk (hot plugged)** window, select the disk from the **Source** list and click **Save**.
5. Optional: Select the type of the interface bus. The options are **VirtIO** and **SCSI**. The default bus type is **VirtIO**.
6. Optional: Change the type of the interface bus of an existing hot plugged disk:

a. Click the Options menu  beside the disk and select the **Edit** option.

b. In the **Interface** field, select the desired option.

7. Optional: Unplug a hot plugged disk:

a. Click the Options menu  beside the disk and select **Detach**.

b. Click **Detach**.

7.14.1.2. Hot plug and hot unplug a disk by using the CLI

You can hot plug and hot unplug a disk while a virtual machine (VM) is running by using the command line.

The hot plugged disk remains attached to the VM until you unplug it.

Prerequisites

- You must have at least one data volume or persistent volume claim (PVC) available for hot plugging.

Procedure

- Hot plug a disk by running the following command:

```
$ virtctl addvolume <virtual-machine|virtual-machine-instance> \
  --volume-name=<datavolume|PVC> \
  [--bus <bus_type>] [--persist] [--serial=<label_name>]
```

where:

--bus <bus_type>

Optional: Specifies the bus type of the added disk. Supported values are **virtio** and **scsi**. The default bus type is **virtio**.

--persist

Optional: Specifies that the virtual disk is permanently mounted on a virtual machine. This flag does not apply to virtual machine instances.

--serial=<label_name>

Optional: Specifies an alphanumeric string label of your choice to identify the hot plugged disk in a guest virtual machine. If you do not specify this option, the label defaults to the name of the hot plugged data volume or PVC.

- Hot unplug a disk by running the following command:

```
$ virtctl removevolume <virtual-machine|virtual-machine-instance> \
  --volume-name=<datavolume|PVC>
```

7.14.1.3. Additional resources

- Available VirtIO Ports

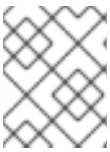
7.14.2. Expand virtual machine disks

Expand the persistent volume claim (PVC) of your virtual machine disk to accommodate growing data requirements. If your storage provider does not support volume expansion, you can expand the available virtual storage of a VM by adding blank data volumes.

You cannot reduce the size of a VM disk.

7.14.2.1. Expand a VM disk PVC by using the web console

You can increase the size of a virtual machine (VM) disk by expanding the persistent volume claim (PVC) of the disk. To specify the increased PVC volume, you can use the **VirtualMachines** page in the web console, with the VM running.

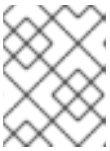


NOTE

If the PVC uses the file system volume mode, the disk image file expands to the available size while reserving some space for file system overhead.

Procedure

1. In the **Administrator** or **Virtualization** perspective, open the **VirtualMachines** page.
2. Select the running VM to open its **Details** page.
3. Select the **Configuration** tab and click **Storage**.
4. Click the options menu  next to the disk you want to expand. Select the **Edit** option. The **Edit disk** dialog opens.
5. In the **PersistentVolumeClaim size** field, enter the desired size.



NOTE

You can enter any value greater than the current one. However, if the new value exceeds the available size, an error is displayed.

6. Click **Save**.

7.14.2.2. Expand a VM disk PVC by using the CLI

You can increase the size of a virtual machine (VM) disk by expanding the persistent volume claim (PVC) of the disk. To specify the increased PVC volume, you can edit the **PersistentVolumeClaim** manifest by using the OpenShift CLI (**oc**).



NOTE

If the PVC uses the file system volume mode, the disk image file expands to the available size while reserving some space for file system overhead.

Prerequisites

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

Procedure

1. Edit the **PersistentVolumeClaim** manifest of the VM disk that you want to expand:

```
$ oc edit pvc <pvc_name>
```

2. Update the disk size:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: vm-disk-expand
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
# ...
```

- **spec.resources.requests.storage** specifies the new disk size.

7.14.2.3. Expand VM storage by adding blank data volumes

You can expand the available storage of a virtual machine (VM) by adding blank data volumes.

Prerequisites

- You must have at least one persistent volume.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **DataVolume** manifest as shown in the following example:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: blank-image-datavolume
spec:
  source:
    blank: {}
  storage:
    resources:
      requests:
        storage: <2Gi>
  storageClassName: "<storage_class>"
```

- **spec.storage.resources.requests.storage** specifies the amount of available space requested for the data volume.
- **spec.storageClassName** is an optional field that specifies a storage class. If you do not specify a storage class, the default storage class is used.

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

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

7.14.2.4. Additional resources

- Extending a basic volume in Windows
- Extending an existing file system partition without destroying data in Red Hat Enterprise Linux
- Extending a logical volume and its file system online in Red Hat Enterprise Linux
- Configuring preallocation mode for data volumes
- Managing data volume annotations

7.15. CPU MODELS

7.15.1. Choose a CPU model

You can choose an appropriate CPU model for your virtual machines based on your environment. The host-model CPU provides maximum performance and feature access in homogeneous clusters. CPU models ensure compatibility and migration flexibility in heterogeneous clusters with different CPU types.

7.15.1.1. Host-model CPU for homogeneous clusters

By default, a virtual machine instance (VMI) uses the **host-model** CPU configuration, which selects the CPU model closest to the node's physical CPU. This gives the VMI all CPU features that the node provides, and suits homogeneous clusters.

The CPU model chosen by **host-model** is often not a standard named model supported by libvirt, which can include additional CPU features. Libvirt creates a CPU definition that only includes the available features on the initial node. This approach ensures:

- The VMI can start successfully on the initial node.
- Future live migrations only require the subset of CPU features used in the host-model, even if target nodes have different or additional features.

7.15.1.1.1. Homogeneous clusters

In a homogeneous cluster, all nodes have similar CPU features. The **host-model** CPU model is recommended for homogeneous clusters, and provides the following benefits:

- Provides VMs with the closest match to the physical CPU.
- VMs can use all CPU features available on the nodes.

- Live migration works smoothly because all nodes share the same CPU features.

7.15.1.1.2. Live migration with host-model

Since the CPU model and features are determined by the initial node, all the host model required CPU features exposed to the VMI must also exist on any future live migration target nodes.

OpenShift Virtualization ensures compatibility by:

1. Detecting all **host-model-required-features.node.kubevirt.io/<cpuFeature>** labels on the initial node.
2. Storing these labels per VMI.
3. Using these same required CPU features to set node selectors for all future live migrations.

This approach guarantees that VM migrations only target nodes that provide the CPU features actually needed by the VMI.



IMPORTANT

Target nodes might support more CPU features than required, but they must support at minimum all features used by the VMI.

7.15.1.2. Named CPU models for heterogeneous clusters

You can use a named CPU model to select a specific CPU type for a virtual machine (VM), either for one VM or as a cluster-wide default. In a heterogeneous cluster, a common named model lets a VM migrate to more nodes.

A node supports a named CPU model only if it includes all required CPU features defined by libvirt for that model. Nodes that support a specific CPU model are labeled with **cpu-model.node.kubevirt.io/<cpuModel>**. OpenShift Virtualization uses these labels to schedule VMs only to compatible nodes. When you configure a named CPU model, the VM receives a node selector that ensures it only runs on nodes labeled with that specific CPU model.

7.15.1.2.1. Heterogeneous clusters

In a heterogeneous cluster, nodes have different CPU generations or feature sets.

A virtual machine instance (VMI) inherits the host model CPU feature of the node where it is first scheduled. If the initial node has CPU features unsupported by other nodes, it can prevent the VM from being migrated to them.

Using a common named CPU model (for example, **EPYC-IBPB** or **Haswell**) is recommended for CPU models in heterogeneous clusters, and provides the following benefits:

- Ensures that all nodes selected for migration support the required CPU features.
- Maximizes the number of nodes with different CPU types that a VM can migrate to.
- Prevents scheduling VMs on nodes that cannot support live migration targets.



IMPORTANT

In heterogeneous clusters, using **host-model** can limit live migration options if the initial scheduling node has host model required CPU features that are not available on all other nodes in the cluster.

7.15.2. Configure CPU models

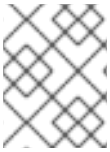
You can configure CPU models for your virtual machines at both the cluster level and individual VM level. Set a cluster-wide default CPU model to automatically apply to all new VMs, or configure a specific CPU model for individual VMs to override the cluster default.

7.15.2.1. Configure the default CPU model

Define a cluster-wide default CPU model. Every new virtual machine (VM) that does not specify its own CPU model uses the default, and runs only on nodes that support it.

When you set a cluster-wide default CPU model:

- Every new virtual machine (VM) that does not have an explicit CPU model defined receives a node selector for the chosen CPU model.
- Only nodes labeled with **cpu-model.node.kubevirt.io/<cpuModel>** are eligible to run VMs using the default model.
- Nodes that do not support the selected CPU model are not considered during VM scheduling.



NOTE

The **defaultCPUModel** is case sensitive and must match a CPU model supported by nodes in your cluster.

A CPU model configured at the VM level always takes precedence over the cluster-wide default CPU model.

Prerequisites

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

Procedure

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

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

2. Add the **defaultCPUModel** field to the CR and set the value to the name of a CPU model that exists in the cluster:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
 name: kubevirt-hyperconverged
```

```
 namespace: openshift-cnv
spec:
 defaultCPUModel: "EPYC-IBPB"
```

where:

EPYC-IBPB

Specifies a CPU model that is supported by nodes in your cluster.

3. Apply the YAML file to your cluster.

7.15.2.2. Configure a CPU model for a single VM

Set a named CPU model on a single virtual machine (VM) when that VM needs a different CPU type from the cluster-wide default. A per-VM model always takes precedence over the default.

The VM CPU model depends on the availability of CPU models within the VM and the cluster.

- If the VM does not have a defined CPU model, the **defaultCPUModel** is automatically set using the CPU model defined at the cluster-wide level.
- If both the VM and the cluster have a defined CPU model, the VM's CPU model takes precedence.
- If neither the VM nor the cluster have a defined CPU model, the host-model is automatically set using the CPU model defined at the host level.

Prerequisites

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

Procedure

1. To configure a named CPU model for a specific VM, edit the **VirtualMachine** custom resource (CR):

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

- **spec.template.spec.domain.cpu.model** defines the named CPU model for the VM.

2. Apply the CR to your cluster.

7.15.3. Schedule virtual machines

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

7.15.3.1. Set a VM to inherit the node CPU model

Set the CPU model of a virtual machine (VM) to **host-model** so that the VM inherits the CPU model of the node where it is scheduled, and can use the CPU features that the node provides.

Procedure

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

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

  - **spec.template.spec.domain.cpu.model** defines the VM that inherits the CPU model of the node where it is scheduled.

7.15.3.2. Set a VM to run only on nodes with a named CPU model

Set a named CPU model on a virtual machine (VM) so that the scheduler places the VM only on nodes that support that model.

Procedure

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

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

  - **spec.template.spec.domain.cpu.model** defines the CPU model for the VM.

7.15.3.3. CPU feature policy attributes

A policy attribute controls how strictly the scheduler enforces a CPU feature that you set on a virtual machine (VM). Use a policy attribute to require a feature, treat it as optional, or forbid it.

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

7.15.3.4. Set a policy attribute and CPU feature

Set a CPU feature and a policy attribute on a virtual machine (VM) so that the VM runs only where the feature is available. The feature must be supported by the host CPU or emulated by the hypervisor.

Procedure

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

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

  - **spec.template.spec.domain.cpu.features.name** defines the name of the CPU feature for the VM.
  - **spec.template.spec.domain.cpu.features.policy** defines the policy attribute for the VM.

7.15.3.5. Schedule VMs with a custom scheduler

You can run a virtual machine (VM) under a custom scheduler instead of the default scheduler, so that your own scheduling logic decides which node runs the VM.

Prerequisites

- A secondary scheduler is configured for your cluster.
- You have installed the OpenShift CLI (**oc**).

Procedure

- Add the custom scheduler to the VM configuration by editing the **VirtualMachine** manifest. For example:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-fedora
spec:
  runStrategy: Always
  template:
    spec:
      schedulerName: my-scheduler
      domain:
        devices:
          disks:
            - name: containerdisk
              disk:
                bus: virtio
# ...
```

schedulerName

The name of the custom scheduler. If the **schedulerName** value does not match an existing scheduler, the **virt-launcher** pod stays in a **Pending** state until the specified scheduler is found.

Verification

- Verify that the VM is using the custom scheduler specified in the **VirtualMachine** manifest by checking the **virt-launcher** pod events:

  a. View the list of pods in your cluster by entering the following command:

  ```
  $ oc get pods
  ```

  Example output:

  ```
  NAME                            READY   STATUS    RESTARTS   AGE
  virt-launcher-vm-fedora-dpc87   2/2     Running   0          24m
  ```

  b. Run the following command to display the pod events:

  ```
  $ oc describe pod virt-launcher-vm-fedora-dpc87
  ```

The value of the **From** field in the output verifies that the scheduler name matches the custom scheduler specified in the **VirtualMachine** manifest:

Example output:

```
[...]
Events:
  Type    Reason     Age   From            Message
  ----    ------     ----  ----            -------
  Normal  Scheduled  21m   my-scheduler  Successfully assigned default/virt-launcher-vm-fedora-dpc87 to node01
[...]
```

CHAPTER 8. NETWORKING

8.1. NETWORKING OVERVIEW

To connect virtual machines (VMs) to cluster networks, configure default and user-defined networking options in OpenShift Virtualization.

8.1.1. OpenShift Virtualization networking glossary

Definitions of key OpenShift Virtualization networking terms and technologies.

Container Network Interface (CNI)

A Cloud Native Computing Foundation project, focused on container network connectivity. OpenShift Virtualization uses CNI plugins to build upon the basic Kubernetes networking functionality.

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.

UserDefinedNetwork

A namespace-scoped CRD introduced by the user-defined network (UDN) API that can be used to create a tenant network that isolates the tenant namespace from other namespaces.

ClusterUserDefinedNetwork

A cluster-scoped CRD introduced by the user-defined network API that cluster administrators can use to create a shared network across multiple namespaces.

8.1.2. Overlay networks

To ensure your virtual machines (VMs) connect reliably by using the standard OpenShift Dedicated networking model, configure the default pod network for cluster-wide connectivity.

Overlay networks provide a flexible, software-defined layer of connectivity on top of a physical network, enabling services like network segmentation, custom routing, and simplified management without altering the underlying hardware.

Connect a VM to the default pod network

Each VM is connected by default to the default internal pod network. You can add or remove network interfaces by editing the VM specification.
You can access a virtual machine (VM) that is connected to the default internal pod network on a stable fully qualified domain name (FQDN) by using headless services.

Connect a VM to a custom primary overlay network

Configure a primary user-defined network (UDN) that supports multi-namespace connectivity to provide isolated and flexible traffic paths for your workloads.
Cluster administrators can configure a primary **UserDefinedNetwork** CRD to create a tenant network that isolates the tenant namespace from other namespaces without requiring network policies. Additionally, cluster administrators can use the **ClusterUserDefinedNetwork** CRD to create a shared OVN layer 2 network across multiple namespaces.

User-defined networks with the layer 2 overlay topology are useful for VM workloads, and a good alternative to secondary networks in environments where physical network access is limited, such as the public cloud. The layer 2 topology enables seamless migration of VMs without the need for Network Address Translation (NAT), and also provides persistent IP addresses that are preserved between reboots and during live migration.

Connect a VM to a custom secondary overlay network

Configure a secondary UDN with layer 2 topology to create a private isolated communication channel between a group of VMs across different nodes. A layer 2 topology connects workloads by a cluster-wide logical switch. The OVN-Kubernetes CNI plugin uses the Geneve (Generic Network Virtualization Encapsulation) protocol to create an overlay network between nodes. You can use this overlay network to connect VMs on different nodes, without having to configure any additional physical networking infrastructure.

Configure external ingress by exposing a VM as a service

You can expose a VM within the cluster or outside the cluster by creating a **Service** object.

8.1.3. VM physical network access methods

To give virtual machines (VMs) access to the internet or other physical devices, you configure the node network, define the secondary network, and attach the VM to the secondary network.

Connect a VM to the physical network by using an Open vSwitch bridge

You can connect a VM to the physical network infrastructure by configuring an OVN-Kubernetes secondary user-defined network (UDN) with the localnet topology.
A localnet topology connects the secondary network to the physical underlay. This enables both east-west cluster traffic and access to services running outside the cluster, but it requires additional configuration of the underlying Open vSwitch (OVS) bridge on cluster nodes.

Cluster administrators can use the following steps to configure the localnet UDN:

1. Install the Kubernetes NMState Operator which provides a state-driven network configuration across cluster nodes.
2. Use the **NodeNetworkConfigurationPolicy** custom resource (CR) to configure OVS bridges and add the appropriate bridge mappings on the nodes.
3. Use the **ClusterUserDefinedNetwork** CR from the UDN API to attach their workload to the underlay network through the OVS bridges configured in the previous step.

8.1.4. VM network interface management options

Manage virtual machine (VM) network configuration to scale connectivity without incurring application downtime, troubleshoot network latency, define and automate management of MAC address pools, configure IP addresses, and isolate live migration traffic.

Hot plug secondary network interfaces

You can add or remove secondary network interfaces without stopping your VM. OpenShift Virtualization supports hot plugging and hot unplugging for secondary interfaces that use bridge binding and the OVN-Kubernetes **layer2** topology.

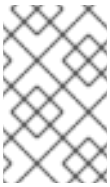
Access a VM by using its external FQDN

You can access a virtual machine (VM) that is attached to a secondary network interface from outside the cluster by using its fully qualified domain name (FQDN). To connect to a VM by using its

external FQDN, you must configure the DNS server, retrieve the cluster FQDN, and then connect to the VM by using the **ssh** command.

Manage the link state of a VM network interface

You can manage the link state of a primary or secondary VM network interface by using the OpenShift Dedicated web console or the command line. By specifying the link state, you can logically connect or disconnect the virtual network interface controller (vNIC) from a network.



NOTE

OpenShift Virtualization does not support link state management for Single Root I/O Virtualization (SR-IOV) secondary network interfaces and their link states are not reported.

Configure and view VM IP address

You can configure the IP address of a secondary network interface when you create a VM. The IP address is provisioned with cloud-init. You can view the IP address of a VM by using the OpenShift Dedicated web console or the command line. The network information is collected by the QEMU guest agent.

Manage MAC address pools for VM network interfaces

The KubeMacPool component allocates MAC addresses for VM network interfaces from a shared MAC address pool. This ensures that each network interface is assigned a unique MAC address. A virtual machine instance created from that VM retains the assigned MAC address across reboots.

View network details in the web console

On the **VirtualMachine details** page, the **Configuration → Network** tab lists the VM's network interfaces. Each network name is a link that opens the details page for the associated network resource, such as a network attachment definition (NAD), user-defined network (UDN), or cluster user-defined network (CUDN). Networks that use the default pod network display "Pod networking" as plain text because there is no corresponding network resource to go to.

8.1.5. VM SSH access

You can use SSH to securely access your virtual machines (VMs) from the command line.

To set up your SSH configuration, use one of the following methods:

Use the virtctl ssh command

You create an SSH key pair, add the public key to a VM, and connect to the VM by running the **virtctl ssh** command with the private key.

You can add public SSH keys to Red Hat Enterprise Linux (RHEL) 9 VMs at runtime or at first boot to VMs with guest operating systems that can be configured by using a cloud-init data source.

Use the virtctl port-forward command

You add the **virtctl port-foward** command to your **.ssh/config** file and connect to the VM by using OpenSSH.

Service

You create a service, associate the service with the VM, and connect to the IP address and port exposed by the service.

Secondary network

You configure a secondary network, attach a VM to the secondary network interface, and connect to its allocated IP address.

8.1.6. Additional resources

- Connect a virtual machine to the default pod network
- Connect a virtual machine to a custom primary overlay network
- Configure external ingress by exposing a VM as a service
- Add a VM to a Service Mesh

8.2. CONNECTING A VIRTUAL MACHINE TO THE DEFAULT POD NETWORK

You can connect a virtual machine to the default internal pod network by configuring its network interface to use the **masquerade** binding mode.



NOTE

Traffic passing through network interfaces to the default pod network is interrupted during live migration.

8.2.1. Configure masquerade mode by using the CLI

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

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

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- The virtual machine must be configured to use DHCP to acquire IPv4 addresses.

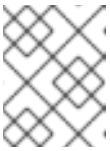
Procedure

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

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - name: default
```

```
            masquerade: {}
            ports:
              - port: 80
# ...
      networks:
      - name: default
        pod: {}
```

- **spec.template.spec.domain.devices.interfaces.masquerade** connects the virtual machine using masquerade mode.
- **spec.template.spec.domain.devices.interfaces.ports** is optional and defines the list of ports that you want to expose from the virtual machine, each specified by the **port** field. The **port** value must be a number between 0 and 65536. When the **ports** array is not used, all ports in the valid range are open to incoming traffic. In this example, incoming traffic is allowed on port **80**.



NOTE

Ports 49152 and 49153 are reserved for use by the libvirt platform and all other incoming traffic to these ports is dropped.

2. Create the virtual machine:

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

8.2.2. Jumbo frames support

When using the OVN-Kubernetes CNI plugin, you can send unfragmented jumbo frame packets between two virtual machines (VMs) that are connected on the default pod network. Jumbo frames have a maximum transmission unit (MTU) value greater than 1500 bytes.

The VM automatically gets the MTU value of the cluster network, set by the cluster administrator, in one of the following ways:

- **libvirt**: If the guest OS has the latest version of the VirtIO driver that can interpret incoming data via a Peripheral Component Interconnect (PCI) config register in the emulated device.
- DHCP: If the guest DHCP client can read the MTU value from the DHCP server response.



NOTE

For Windows VMs that do not have a VirtIO driver, you must set the MTU manually by using **netsh** or a similar tool. This is because the Windows DHCP client does not read the MTU value.

8.3. CONNECTING A VIRTUAL MACHINE TO A PRIMARY USER-DEFINED NETWORK

You can connect a virtual machine (VM) to a user-defined network (UDN) on the VM's primary interface. The primary UDN replaces the default pod network in your specified namespace. You can define the primary UDN per project, where each project can use its specific subnet and topology.

OpenShift Virtualization supports the namespace-scoped **UserDefinedNetwork** and the cluster-scoped **ClusterUserDefinedNetwork** custom resource definitions (CRD).



NOTE

You must add the **k8s.ovn.org/primary-user-defined-network** label when you create a namespace that is to be used with user-defined networks.

Cluster administrators can configure a primary **UserDefinedNetwork** CRD to create a tenant network that isolates the tenant namespace from other namespaces without requiring network policies. Additionally, cluster administrators can use the **ClusterUserDefinedNetwork** CRD to create a shared OVN network across multiple namespaces.

With the layer 2 topology, OVN-Kubernetes creates an overlay network between nodes. You can use this overlay network to connect VMs on different nodes without having to configure any additional physical networking infrastructure.

The layer 2 topology enables seamless migration of VMs without the need for Network Address Translation (NAT) because persistent IP addresses are preserved across cluster nodes during live migration.



NOTE

You can use the Ethernet Virtual Private Network's (EVPN) Border Gateway Protocol (BGP) control plane to extend layer 2 connectivity between VMs running on different OpenShift Dedicated clusters. BGP EVPN support for primary cluster-scoped UDNs enables deeper integration with data center networks for your virtualized workloads. For more information, see "About BGP EVPN for primary cluster user-defined networks" in the Additional resources section.

You must consider the following limitations before implementing a primary UDN:

- You cannot use the **virtctl ssh** command to configure SSH access to a VM.
- You cannot use the **oc port-forward** command to forward ports to a VM.
- You cannot use headless services to access a VM.

8.3.1. Primary overlay network creation in the web console

You can use the OpenShift Dedicated web console to create a primary namespace-scoped **UserDefinedNetwork** (UDN) or a cluster-scoped **ClusterUserDefinedNetwork** custom resource definition (CRD).

The UDN serves as the default primary network for pods and VMs that you create in namespaces associated with the network.

After you define the custom primary overlay network, you can create namespaces that are associated with the cluster-scoped UDN.

8.3.1.1. Create a namespace for overlay networks by using the web console

You can create a namespace to be used with primary user-defined networks (UDNs) by using the OpenShift Dedicated web console.

Prerequisites

- Log in to the OpenShift Dedicated web console as a user with **cluster-admin** permissions.

Procedure

1. From the **Administrator** perspective, click **Administration → Namespaces**.
2. Click **Create Namespace**.
3. In the **Name** field, specify a name for the namespace. The name must consist of lower case alphanumeric characters or '-', and must start and end with an alphanumeric character.
4. In the **Labels** field, add the **k8s.ovn.org/primary-user-defined-network** label.
5. Optional: If the namespace is to be used with an existing cluster-scoped UDN, add the appropriate labels as defined in the **spec.namespaceSelector** field in the **ClusterUserDefinedNetwork** custom resource.
6. Optional: Specify a default network policy.
7. Click **Create** to create the namespace.

8.3.1.2. Create a namespace-scoped primary overlay network in the web console

You can create an isolated primary network in your project namespace by creating a **UserDefinedNetwork** custom resource in the OpenShift Dedicated web console.

Prerequisites

- You have access to the OpenShift Dedicated web console as a user with **cluster-admin** permissions.
- You have created a namespace and applied the **k8s.ovn.org/primary-user-defined-network** label. For more information, see "Creating a namespace for user-defined networks by using the web console".

Procedure

1. From the **Administrator** perspective, click **Networking → UserDefinedNetworks**.
2. Click **Create UserDefinedNetwork**.
3. From the **Project name** list, select the namespace that you previously created.
4. Specify a value in the **Subnet** field.
5. Click **Create**. The user-defined network serves as the default primary network for pods and virtual machines that you create in this namespace.

8.3.1.3. Creating a cluster-scoped network to connect pods directly to an external network

You can connect one or more projects to a physical network for direct layer 2 access to data center resources through a **ClusterUserDefinedNetwork** custom resource in the OpenShift Dedicated web console.

Prerequisites

- You have access to the OpenShift Dedicated web console as a user with **cluster-admin** permissions.

Procedure

1. In the OpenShift Dedicated web console, go to **Virtualization → Networking**.
2. Click **Virtual machine networks**in the navigation pane.
3. Click **Create**. The **Create virtual machine network**wizard is displayed.
4. Give details about the network on the **Network definition** page:
   a. Enter a name for the network in the **Name** field.
   b. Select a physical network through an **OpenvSwitch** bridge from the **Select physical network** list.
   c. Enter the maximum transmission unit (MTU).



NOTE

An MTU, measured in bytes, is the largest allowable size of a data packet. Ensure that all underlying physical network equipment supports this MTU, or higher.

   d. Optional: Select the **VLAN ID** checkbox to enter VLAN tagging information. If you tag traffic with a VLAN ID, you must configure your physical switch with a VLAN trunk that includes the VLAN ID that you choose.
5. Click **Next**.
6. Select the projects that the network should be made available to on the **Project mapping** page. By default, all projects have access to the network.
7. Click **Create**.

Verification

1. Navigate to the **Virtualization → Virtual machine networks**page.
2. Click the **OVN localnet** tab.
3. Verify that your new network is displayed in the list.

8.3.1.4. Create a cluster-scoped primary overlay network in the web console

You can connect multiple namespaces to the same primary user-defined network (UDN) by creating a **ClusterUserDefinedNetwork** custom resource in the OpenShift Dedicated web console.

Prerequisites

- You have access to the OpenShift Dedicated web console as a user with **cluster-admin** permissions.

Procedure

1. From the **Administrator** perspective, click **Networking → UserDefinedNetworks**.
2. From the **Create** list, select **ClusterUserDefinedNetwork**.
3. In the **Name** field, specify a name for the cluster-scoped UDN.
4. Specify a value in the **Subnet** field.
5. In the **Project(s) Match Labels** field, add the appropriate labels to select namespaces that the cluster UDN applies to.
6. Click **Create**. The cluster-scoped UDN serves as the default primary network for pods and virtual machines located in namespaces that contain the labels that you specified in step 5.

8.3.2. Primary overlay network creation by using the CLI

You can create a primary **UserDefinedNetwork** or **ClusterUserDefinedNetwork** custom resource definition (CRD) by using the OpenShift CLI (**oc**). After you define the custom primary overlay network, you can create namespaces that are associated with the cluster-scoped UDN.

8.3.2.1. Create a namespace for overlay networks by using the CLI

You can create a namespace to be used with primary user-defined networks (UDNs) by using the OpenShift CLI (**oc**).

Prerequisites

- You have access to the cluster as a user with **cluster-admin** permissions.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **Namespace** object as a YAML file similar to the following example:

```
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace
  labels:
    k8s.ovn.org/primary-user-defined-network: ""
# ...
```

The **k8s.ovn.org/primary-user-defined-network** label is required for the namespace to be associated with a UDN. If the namespace is to be used with an existing cluster UDN, you must also add the appropriate labels that are defined in the **spec.namespaceSelector** field of the **ClusterUserDefinedNetwork** custom resource.

2. Apply the **Namespace** manifest by running the following command:

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

8.3.2.2. Create a namespace-scoped primary overlay network by using the CLI

You can create an isolated primary network in your project namespace by using the CLI. You must use the OVN-Kubernetes layer 2 topology and enable persistent IP address allocation in the user-defined network (UDN) configuration to ensure VM live migration support.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have created a namespace and applied the **k8s.ovn.org/primary-user-defined-network** label.

Procedure

1. Create a **UserDefinedNetwork** object to specify the custom network configuration.
   Example **UserDefinedNetwork** manifest:

```
apiVersion: k8s.ovn.org/v1
kind: UserDefinedNetwork
metadata:
  name: udn-l2-net
  namespace: my-namespace
spec:
  topology: Layer2
  layer2:
    role: Primary
    subnets:
      - "10.0.0.0/24"
      - "2001:db8::/60"
    ipam:
      lifecycle: Persistent
```

   - **metadata.name** specifies the name of the **UserDefinedNetwork** custom resource.
   - **metadata.namespace** specifies the namespace in which the VM is located. The namespace must have the **k8s.ovn.org/primary-user-defined-network** label. The namespace must not be **default**, an **openshift-*** namespace, or match any global namespaces that are defined by the Cluster Network Operator (CNO).
   - **spec.topology** specifies the topological configuration of the network. The required value is **Layer2**. A **Layer2** topology creates a logical switch that is shared by all nodes.
   - **spec.layer2.role** specifies whether the UDN is primary or secondary. The **Primary** role means that the UDN acts as the primary network for the VM and all default traffic passes through this network.
   - **spec.layer2.ipam.lifecycle** specifies that virtual workloads have consistent IP addresses across reboots and migration. The **spec.layer2.subnets** field is required when **ipam.lifecycle: Persistent** is specified.

2. Apply the **UserDefinedNetwork** manifest by running the following command:

```
$ oc apply -f --validate=true <filename>.yaml
```

8.3.2.3. Create a cluster-scoped primary overlay network by using the CLI

You can connect multiple namespaces to the same primary user-defined network (UDN) to achieve native tenant isolation by using the CLI.

Prerequisites

- You have access to the cluster as a user with **cluster-admin** privileges.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **ClusterUserDefinedNetwork** object to specify the custom network configuration. Example **ClusterUserDefinedNetwork** manifest:

```
apiVersion: k8s.ovn.org/v1
kind: ClusterUserDefinedNetwork
metadata:
  name: cudn-l2-net
spec:
  namespaceSelector:
    matchExpressions:
    - key: kubernetes.io/metadata.name
      operator: In
      values: ["red-namespace", "blue-namespace"]
  network:
    topology: Layer2
    layer2:
      role: Primary
      ipam:
        lifecycle: Persistent
      subnets:
        - 203.203.0.0/16
```

- **metadata.name** specifies the name of the **ClusterUserDefinedNetwork** custom resource.
- **spec.namespaceSelector** specifies the set of namespaces that the cluster UDN applies to. The namespace selector must not point to **default**, an **openshift-*** namespace, or any global namespaces that are defined by the Cluster Network Operator (CNO).
- **spec.namespaceSelector.matchExpressions** specifies the type of selector. In this example, the **matchExpressions** selector selects objects that have the label **kubernetes.io/metadata.name** with the value **red-namespace** or **blue-namespace**.
- **spec.namespaceSelector.matchExpressions.operator** specifies the type of operator. Possible values are **In**, **NotIn**, and **Exists**.
- **spec.network.topology** specifies the topological configuration of the network. The required value is **Layer2**. A **Layer2** topology creates a logical switch that is shared by all nodes.

- **spec.network.layer2.role** specifies whether the UDN is primary or secondary. The **Primary** role means that the UDN acts as the primary network for the VM and all default traffic passes through this network.

2. Apply the **ClusterUserDefinedNetwork** manifest by running the following command:

```
$ oc apply -f --validate=true <filename>.yaml
```

8.3.3. VM attachment to the primary overlay network

You can connect a virtual machine (VM) to the primary user-defined network (UDN) by requesting the pod network attachment and configuring the interface binding.

OpenShift Virtualization supports the following network binding plugins to connect the network interface to the VM:

Layer 2 bridge

The Layer 2 bridge binding creates a direct Layer 2 connection between the VM's virtual interface and the virtual switch of the UDN.

Passt

The Plug a Simple Socket Transport (passt) binding provides a user-space networking solution that integrates seamlessly with the pod network, providing better integration with the OpenShift Dedicated networking ecosystem.
Passt binding has the following benefits:

- You can define readiness and liveness HTTP probes to configure VM health checks.
- You can use Red Hat Advanced Cluster Security to monitor TCP traffic within the cluster with detailed insights.



IMPORTANT

Using the passt binding plugin to attach a VM to the primary UDN is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

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

8.3.3.1. Create a VM with primary overlay access in the web console

You can create a virtual machine (VM) in a namespace that is configured to use a primary user-defined network (UDN) by using the OpenShift Dedicated web console.

By default, the VM automatically connects to the primary UDN using the Layer 2 bridge (**l2bridge**) network binding plugin.



NOTE

When creating the virtual machine, verify that the default network interface type is automatically set to **L2 bridge** and the network attachment is configured for the **pod** network.

Prerequisites

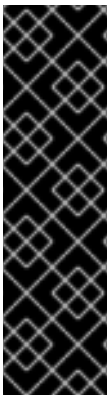
- You have a namespace configured and labeled for a primary user-defined network.
- You are logged in to the OpenShift Dedicated web console.

Procedure

1. Click **Virtualization → VirtualMachines**.
2. Select the UDN-configured namespace from the **Project** drop-down list.
3. Click **Create → With Wizard**.
4. Configure the VM specifications in the wizard and click **Create VirtualMachine**.

8.3.3.2. Attach a VM to a primary overlay network

You can connect a virtual machine (VM) to the primary user-defined network (UDN) by using the OpenShift CLI (**oc**).



IMPORTANT

Using the passt binding plugin to attach a VM to the primary UDN is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

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

Prerequisites

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

Procedure

1. Edit the **VirtualMachine** manifest to add the UDN interface details, as in the following example:
   Example **VirtualMachine** manifest:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: my-namespace
spec:
  template:
```

```
    spec:
      domain:
        devices:
          interfaces:
            - name: default
              binding:
                name: l2bridge
# ...
      networks:
      - name: default
        pod: {}
# ...
```

- **metadata.namespace** specifies the namespace in which the VM is located. This value must match the namespace in which the UDN is defined.
- **spec.template.spec.domain.devices.interfaces.name** specifies the name of the user-defined network interface.
- **spec.template.spec.domain.devices.interfaces.binding.name** specifies the name of the binding plugin that is used to connect the interface to the VM. The possible values are **l2bridge** and **passt**. The default value is **l2bridge**.
- **spec.template.spec.networks.name** specifies the name of the network. This must match the value of the **spec.template.spec.domain.devices.interfaces.name** field.

2. Optional: If you are using the Plug a Simple Socket Transport (passt) network binding plugin, set the **hco.kubevirt.io/deployPasstNetworkBinding** annotation to **true** in the **HyperConverged** custom resource (CR) by running the following command:

```
$ oc annotate hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n
openshift-cnv hco.kubevirt.io/deployPasstNetworkBinding=true --overwrite
```

3. Apply the **VirtualMachine** manifest by running the following command:

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

8.4. EXPOSING A VIRTUAL MACHINE BY USING A SERVICE

You can expose a virtual machine (VM) within or outside the cluster by exposing a VM as a Kubernetes service. You can leverage native load balancing and observability tools that provide unified traffic management, consistent SSL termination, and centralized security policies across hybrid workloads.

8.4.1. Service types for VM network access

A Kubernetes service exposes an application running on a set of pods to network clients. Services provide abstraction and load balancing, and the **NodePort** and **LoadBalancer** types also expose the application outside the cluster.

ClusterIP

Exposes the service on an internal IP address and as a DNS name to other applications within the cluster. A single service can map to multiple virtual machines. When a client tries to connect to the service, the client's request is load balanced among available backends. **ClusterIP** is the default

service type.

NodePort

Exposes the service on the same port of each selected node in the cluster. **NodePort** makes a port accessible from outside the cluster, provided that the node itself is externally accessible to the client.

LoadBalancer

Creates an external load balancer in the current cloud (if supported) and assigns a fixed, external IP address to the service.

8.4.2. Expose a VM as a service by using the CLI

You can use the command line to create a service and associate it with a virtual machine (VM). You must create a label on the VM that matches the selector in the **Service** manifest.

Prerequisites

- You configured the cluster network to support the service.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **VirtualMachine** manifest to add the label for service creation. Add **special: key** to the **spec.template.metadata.labels** stanza:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  runStrategy: Halted
  template:
    metadata:
      labels:
        special: key
# ...
```



NOTE

Labels on a virtual machine pass through to the pod. The **special: key** label must match the label in the **spec.selector** attribute of the **Service** manifest.

2. Save the **VirtualMachine** manifest file to apply your changes.
3. Create a **Service** manifest to expose the VM:

```
apiVersion: v1
kind: Service
metadata:
  name: example-service
  namespace: example-namespace
spec:
```

```
# ...
  selector:
    special: key
  type: NodePort
  ports:
    protocol: TCP
    port: 80
    targetPort: 9376
    nodePort: 30000
```

- **spec.selector** defines the label that you added to the **spec.template.metadata.labels** stanza of the **VirtualMachine** manifest.
- **spec.type** defines the type of service by the way it is exposed. Choose one of **ClusterIP**, **NodePort**, or **LoadBalancer**.
- **spec.ports** defines a collection of network ports and protocols to expose from the virtual machine.

4. Save the **Service** manifest file.
5. Create the service by running the following command:

```
$ oc create -f example-service.yaml
```

6. Restart the VM to apply the changes.

Verification

- Query the **Service** object to verify that it is available:

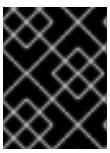
```
$ oc get service -n example-namespace
```

8.5. CONNECTING A VIRTUAL MACHINE TO A SERVICE MESH

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

8.5.1. Connect a virtual machine to a service mesh

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



IMPORTANT

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

Prerequisites

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

- You have installed the Service Mesh Operator.

Procedure

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

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm-istio
  name: vm-istio
spec:
  runStrategy: Always
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm-istio
        app: vm-istio
      annotations:
        sidecar.istio.io/inject: "true"
    spec:
      domain:
        devices:
          interfaces:
          - name: default
            masquerade: {}
          disks:
          - disk:
              bus: virtio
            name: containerdisk
          - disk:
              bus: virtio
            name: cloudinitdisk
        resources:
          requests:
            memory: 1024M
      networks:
      - name: default
        pod: {}
      terminationGracePeriodSeconds: 180
      volumes:
      - containerDisk:
          image: registry:5000/kubevirt/fedora-cloud-container-disk-demo:devel
        name: containerdisk
```

- **spec.template.metadata.labels.app** specifies the key/value pair (label) that must be matched to the service selector attribute.
- **spec.template.metadata.annotations.sidecar.istio.io/inject** is the annotation to enable automatic sidecar injection.
- **spec.template.spec.domain.devices.interfaces.masquerade** is the binding method (masquerade mode) for use with the default pod network.

2. Run the following command to apply the VM configuration:

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

where:

<vm_name>

Specifies the name of the virtual machine YAML file.

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

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

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

4. Run the following command to create the service:

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

where:

<service_name>

Specifies the name of the service YAML file.

8.6. CONFIGURING AND VIEWING IP ADDRESSES

You can configure an IP address when you create a virtual machine (VM). The IP address is provisioned with cloud-init. View the IP address of a VM by using the OpenShift Dedicated web console or the command line. The network information is collected by the QEMU guest agent.

8.6.1. Configure a VM IP address by using the web console

You can configure a static IP address when you create a virtual machine (VM) by using the web console. The IP address is provisioned with cloud-init.



NOTE

If the VM is connected to the pod network, the pod network interface is the default route unless you update it.

Prerequisites

- The virtual machine is connected to a secondary network.

Procedure

1. Navigate to **Virtualization → Catalog** in the web console.
2. Click a template tile.
3. Click **Customize VirtualMachine**.
4. Click **Next**.
5. On the **Scripts** tab, click the edit icon beside **Cloud-init**.
6. Select the **Add network data** checkbox.
7. Enter the ethernet name, one or more IP addresses separated by commas, and the gateway address.
8. Click **Apply**.
9. Click **Create VirtualMachine**.

8.6.2. Configure a VM IP address by using the CLI

You can configure a static or dynamic IP address when you create a virtual machine (VM). The IP address is provisioned with cloud-init.



NOTE

If the VM is connected to the pod network, the pod network interface is the default route unless you update it.

Prerequisites

- The virtual machine is connected to a secondary network.
- You have a DHCP server available on the secondary network to configure a dynamic IP for the virtual machine.

Procedure

- Edit the **spec.template.spec.volumes.cloudInitNoCloud.networkData** stanza of the virtual machine configuration:
  - To configure a dynamic IP address, specify the interface name and enable DHCP:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: <interface_name>
  kind: VirtualMachine
spec:
# ...
```

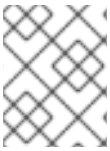
```
template:
# ...
  spec:
    volumes:
    - cloudInitNoCloud:
        networkData: |
          version: 2
          ethernets:
            eth1:
              dhcp4: true
  # ...
```

- To configure a static IP, specify the interface name and the IP address:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: <interface_name>
  kind: VirtualMachine
spec:
# ...
  template:
  # ...
    spec:
      volumes:
      - cloudInitNoCloud:
          networkData: |
            version: 2
            ethernets:
              eth1:
                addresses:
                - 10.10.10.14/24
    # ...
```

8.6.3. View a VM IP address by using the web console

You can view the IP address of a virtual machine (VM) by using the OpenShift Dedicated web console.



NOTE

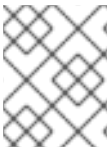
You must install the QEMU guest agent on a VM to view the IP address of a secondary network interface. A pod network interface does not require the QEMU guest agent.

Procedure

1. In the OpenShift Dedicated console, click **Virtualization → VirtualMachines** from the side menu.
2. Select a VM to open the **VirtualMachine details** page.
3. Click the **Details** tab to view the IP address.

8.6.4. View a VM IP address by using the CLI

You can view the IP address of a virtual machine (VM) by using the command line.



NOTE

You must install the QEMU guest agent on a VM to view the IP address of a secondary network interface. A pod network interface does not require the QEMU guest agent.

Prerequisites

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

Procedure

- Obtain the virtual machine instance configuration by running the following command:

```
$ oc describe vmi <vmi_name>
```

Example output:

```
# ...
Interfaces:
  Interface Name:  eth0
  Ip Address:      10.244.0.37/24
  Ip Addresses:
    10.244.0.37/24
    fe80::858:aff:fef4:25/64
  Mac:             0a:58:0a:f4:00:25
  Name:            default
  Interface Name:  v2
  Ip Address:      1.1.1.7/24
  Ip Addresses:
    1.1.1.7/24
    fe80::f4d9:70ff:fe13:9089/64
  Mac:             f6:d9:70:13:90:89
  Interface Name:  v1
  Ip Address:      1.1.1.1/24
  Ip Addresses:
    1.1.1.1/24
    1.1.1.2/24
    1.1.1.4/24
    2001:de7:0:f101::1/64
    2001:db8:0:f101::1/64
    fe80::1420:84ff:fe10:17aa/64
  Mac:             16:20:84:10:17:aa
```

8.6.5. Additional resources

- Installing the QEMU guest agent

CHAPTER 9. STORAGE

9.1. STORAGE CONFIGURATION OVERVIEW

You can configure a default storage class, storage profiles, Containerized Data Importer (CDI), data volumes (DVs), and automatic boot source updates.

On OpenShift Dedicated, specific configurations for Hyperdisk are required to ensure performance and support features like live migration. For more information, see "Storage configuration for OpenShift Virtualization 4.21.x on Google Cloud" in the "Additional resources" section.

9.1.1. Storage

The following storage configuration tasks are mandatory:

Configure storage profiles

You must configure storage profiles if your storage provider is not recognized by CDI. A storage profile provides recommended storage settings based on the associated storage class.

The following storage configuration tasks are optional:

Reserve additional PVC space for file system overhead

By default, 5.5% of a file system PVC is reserved for overhead, reducing the space available for VM disks by that amount. You can configure a different overhead value.

Configure local storage by using the hostpath provisioner

You can configure local storage for virtual machines by using the hostpath provisioner (HPP). When you install the OpenShift Virtualization Operator, the HPP Operator is automatically installed.

Configure user permissions to clone data volumes between namespaces

You can configure RBAC roles to enable users to clone data volumes between namespaces.

9.1.2. Containerized Data Importer

You can perform the following Containerized Data Importer (CDI) configuration tasks:

Override the resource request limits of a namespace

You can configure CDI to import, upload, and clone VM disks into namespaces that are subject to CPU and memory resource restrictions.

Configure CDI scratch space

CDI requires scratch space (temporary storage) to complete some operations, such as importing and uploading VM images. During this process, CDI provisions a scratch space PVC equal to the size of the PVC backing the destination data volume (DV).

9.1.3. Data volumes

You can perform the following data volume configuration tasks:

Enable preallocation for data volumes

CDI can preallocate disk space to improve write performance when creating data volumes. You can enable preallocation for specific data volumes.

Manage data volume annotations

Data volume annotations allow you to manage pod behavior. You can add one or more annotations to a data volume, which then propagates to the created importer pods.

9.1.4. Boot source updates

You can perform the following boot source update configuration task:

Manage automatic boot source updates

Boot sources can make virtual machine (VM) creation more accessible and efficient for users. If automatic boot source updates are enabled, CDI imports, polls, and updates the images so that they are ready to be cloned for new VMs. By default, CDI automatically updates Red Hat boot sources. You can enable automatic updates for custom boot sources.

9.1.5. Additional resources

- Configure storage profiles
- Reserve additional PVC space for file system overhead
- Configure local storage by using the hostpath provisioner
- Configure user permissions to clone data volumes between namespaces
- Override the resource request limits of a namespace
- Configure CDI scratch space
- Enable preallocation for data volumes
- Manage data volume annotations
- Manage automatic boot source updates
- Storage configuration for OpenShift Virtualization 4.22.x on Google Cloud

9.2. CONFIGURING STORAGE PROFILES

A storage profile provides recommended storage settings based on the associated storage class and is allocated for each storage class.

The Containerized Data Importer (CDI) recognizes a storage provider if it has been configured to identify and interact with the storage provider's capabilities.

For recognized storage types, the CDI provides values that optimize the creation of PVCs. You can also configure automatic settings for the storage class by customizing the storage profile. If the CDI does not recognize your storage provider, you must configure storage profiles.

9.2.1. Customizing the storage profile

You can specify default parameters by editing the **StorageProfile** object for the storage class of the provisioner. These default parameters only apply to the persistent volume claim (PVC) if they are not configured in the **DataVolume** object.

You cannot modify storage class parameters. To make changes, delete and re-create the storage class. You must then reapply any customizations that were previously made to the storage profile.

An empty **status** section in a storage profile indicates that a storage provisioner is not recognized by the Containerized Data Importer (CDI). Customizing a storage profile is necessary if you have a storage provisioner that is not recognized by CDI. In this case, the administrator sets appropriate values in the storage profile to ensure successful allocations.

If you are creating a snapshot of a VM, a warning is displayed if the storage class of the disk has more than one **VolumeSnapshotClass** associated with it. In this case, you must specify one volume snapshot class. Otherwise, any disk that has more than one volume snapshot class is excluded from the snapshots list.



WARNING

If you create a data volume and omit YAML attributes and these attributes are not defined in the storage profile, then the requested storage will not be allocated and the underlying persistent volume claim (PVC) will not be created.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- Ensure that your planned configuration is supported by the storage class and its provider. Specifying an incompatible configuration in a storage profile causes volume provisioning to fail.

Procedure

1. Edit the storage profile. In this example, the provisioner is not recognized by CDI.

```
$ oc edit storageprofile <storage_class>
```

2. Specify the **accessModes** and **volumeMode** values you want to configure for the storage profile. For example:

Example storage profile

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

- **spec.claimPropertySets.accessModes** defines how the volume can be mounted. For example, **ReadWriteOnce**

- **spec.claimPropertySets.accessModes.volumeMode** defines whether the volume uses a file system or raw block storage. For example, **volumeMode**.

9.2.1.1. Specifying a volume snapshot class by using the web console

If you are creating a snapshot of a VM, you must specify only one volume snapshot class. Any disk that has more than one volume snapshot class is excluded from the snapshots list. A warning is displayed if the storage class of the disk has more than one volume snapshot class associated with it.

You can specify the default volume snapshot class in the OpenShift Dedicated web console.

Procedure

1. From the **Virtualization** focused view, select **Storage**.
2. Click **VolumeSnapshotClasses**.
3. Select a volume snapshot class from the list.
4. Click the **Annotations** pencil icon.
5. Enter the following **Key**: **snapshot.storage.kubernetes.io/is-default-class**.
6. Enter the following **Value**: **true**.
7. Click **Save**.

9.2.1.2. Specifying a volume snapshot class by using the CLI

If you are creating a snapshot of a VM, you must specify only one volume snapshot class. Any disk that has more than one volume snapshot class is excluded from the snapshots list. A warning is displayed if the storage class of the disk has more than one volume snapshot class associated with it.

You can select which volume snapshot class to use by either:

- Setting the **spec.snapshotClass** for the storage profile.
- Setting a default volume snapshot class.

Prerequisites

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

Procedure

- Set the **VolumeSnapshotClass** you want to use. For example:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: ocs-storagecluster-ceph-rbd-virtualization
spec:
  snapshotClass: ocs-storagecluster-rbdplugin-snapclass
```

- Alternatively, set the default volume snapshot class by running the following command:

```
# oc patch VolumeSnapshotClass ocs-storagecluster-cephfsplugin-snapclass --type=merge -
p '{"metadata":{"annotations":{"snapshot.storage.kubernetes.io/is-default-class":"true"}}}'
```

9.2.1.3. Viewing automatically created storage profiles

The system creates storage profiles for each storage class automatically. You can view these storage class profiles by using the **oc** command.

Prerequisites

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

Procedure

1. To view the list of storage profiles, run the following command:

```
$ oc get storageprofile
```

2. To fetch the details of a particular storage profile, run the following command:

```
$ oc describe storageprofile <name>
```

Example storage profile details:

```
Name:         ocs-storagecluster-ceph-rbd-virtualization
Namespace:
Labels:       app=containerized-data-importer
              app.kubernetes.io/component=storage
              app.kubernetes.io/managed-by=cdi-controller
              app.kubernetes.io/part-of=hyperconverged-cluster
              app.kubernetes.io/version=4.17.2
              cdi.kubevirt.io=
Annotations:  <none>
API Version:  cdi.kubevirt.io/v1beta1
Kind:         StorageProfile
Metadata:
  Creation Timestamp:  2023-11-13T07:58:02Z
  Generation:          2
  Owner References:
    API Version:           cdi.kubevirt.io/v1beta1
    Block Owner Deletion:  true
    Controller:            true
    Kind:                  CDI
    Name:                  cdi-kubevirt-hyperconverged
    UID:                   2d6f169a-382c-4caf-b614-a640f2ef8abb
  Resource Version:        4186799537
  UID:                     14aef804-6688-4f2e-986b-0297fd3aaa68
Spec:
Status:
  Claim Property Sets:
    accessModes:
      ReadWriteOnce
    volumeMode:  Block
    accessModes:
```

```
    ReadWriteOnce
  volumeMode:              Filesystem
Clone Strategy:            csi-clone
Data Import Cron Source Format:  snapshot
Provisioner:               openshift-storage.rbd.csi.ceph.com
Snapshot Class:              ocs-storagecluster-rbdplugin-snapclass
Storage Class:             ocs-storagecluster-ceph-rbd-virtualization
Events:                    <none>
```

status.claimPropertySets

Claim Property Sets is an ordered list of **AccessMode**/**VolumeMode** pairs, which describe the PVC modes that are used to provision VM disks.

status.cloneStrategy

The **Clone Strategy** line indicates the clone strategy to be used.

status.dataImportCronSourceFormat

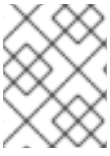
Data Import Cron Source Format indicates whether boot source images on this storage are stored as PVCs or volume snapshots.

9.2.1.4. Setting a default cloning strategy by using a storage profile

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

Cloning strategies are specified by setting the **cloneStrategy** attribute in a storage profile to one of the following values:

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



NOTE

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

Procedure

1. Create or edit a **StorageProfile** object to define the cloning strategy. In the **spec** section, set the **cloneStrategy** field and define the required **claimPropertySets** values, as shown in the following example.

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
```

```
metadata:
  name: <provisioner_class>
# ...
spec:
  claimPropertySets:
  - accessModes:
    - ReadWriteOnce
    volumeMode: Filesystem
  cloneStrategy: csi-clone
status:
  provisioner: <provisioner>
  storageClass: <provisioner_class>
```

- **accessModes** and **volumeMode** define the claim properties.
- **cloneStrategy** sets the default cloning method.

2. Apply the configuration:

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

9.3. MANAGING AUTOMATIC BOOT SOURCE UPDATES

You can manage automatic updates for boot sources used to create virtual machines. This includes configuring update behavior for Red Hat and custom boot sources.

9.3.1. Managing automatic updates for all system-defined boot sources

Disabling automatic boot source imports and updates can lower resource usage. In disconnected environments, disabling automatic boot source updates prevents **CDIDataImportCronOutdated** alerts from filling up logs.

To disable automatic updates for all system-defined boot sources, set the **enableCommonBootImageImport** field value to **false**. Disabling automatic updates deletes the associated **DataImportCron** objects. Setting this value to **true** turns automatic updates back on.



NOTE

Custom boot sources are not affected by this setting.

Prerequisites

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

Procedure

- Enable or disable automatic boot source updates by editing the **HyperConverged** custom resource (CR).
  - To disable automatic boot source updates, set the **spec.enableCommonBootImageImport** field value in the **HyperConverged** CR to **false**. For example:

```
$ oc patch hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n
```

```
openshift-cnv \
  --type json -p '[{"op": "replace", "path": \
  "/spec/enableCommonBootImageImport", \
  "value": false}]'
```

- To re-enable automatic boot source updates, set the **spec.enableCommonBootImageImport** field value in the **HyperConverged** CR to **true**. For example:

```
$ oc patch hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n
openshift-cnv \
  --type json -p '[{"op": "replace", "path": \
  "/spec/enableCommonBootImageImport", \
  "value": true}]'
```

9.3.2. Configuring the default and virt-default storage classes

A storage class determines how persistent storage is provisioned for workloads. In OpenShift Virtualization, the virt-default storage class takes precedence over the cluster default storage class and is used specifically for virtualization workloads.

Only one storage class should be set as virt-default or cluster default at a time. If multiple storage classes are marked as default, the virt-default storage class overrides the cluster default. To ensure consistent behavior, configure only one storage class as the default for virtualization workloads.



IMPORTANT

Boot sources are created using the default storage class. When the default storage class changes, old boot sources are automatically updated using the new default storage class. If your cluster does not have a default storage class, you must define one.

If boot source images were stored as volume snapshots and both the cluster default and virt-default storage class have been unset, the volume snapshots are cleaned up and new data volumes will be created. However the newly created data volumes will not start importing until a default storage class is set.

Prerequisites

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

Procedure

1. Patch the current virt-default or a cluster default storage class to false:

   a. Identify all storage classes currently marked as virt-default by running the following command:

```
$ oc get sc -o json| jq '.items[].metadata|select(.annotations."storageclass.kubevirt.io/is-
default-virt-class"=="true")|.name'
```

   b. For each storage class returned, remove the virt-default annotation by running the following command:

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

c. Identify all storage classes currently marked as cluster default by running the following command:

```
$ oc get sc -o json| jq
'.items[].metadata|select(.annotations."storageclass.kubernetes.io/is-default-
class"=="true")|.name'
```

d. For each storage class returned, remove the cluster default annotation by running the following command:

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

2. Set a new default storage class:

a. Assign the virt-default role to a storage class by running the following command:

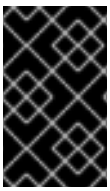
```
$ oc patch storageclass <storage_class_name> -p '{"metadata": {"annotations":
{"storageclass.kubevirt.io/is-default-virt-class": "true"}}}'
```

b. Alternatively, assign the cluster default role to a storage class by running the following command:

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

9.3.3. Configuring a storage class for boot source images

You can configure a specific storage class in the **HyperConverged** resource.



IMPORTANT

To ensure stable behavior and avoid unnecessary re-importing, you can specify the **storageClassName** in the **dataImportCronTemplates** section of the **HyperConverged** resource.

Prerequisites

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

Procedure

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

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

2. Add the **dataImportCronTemplate** to the spec section of the **HyperConverged** resource and set the **storageClassName**:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      name: rhel9-image-cron
    spec:
      template:
        spec:
          storage:
            storageClassName: <storage_class>
      schedule: "0 */12 * * *"
      managedDataSource: <data_source>
# ...
```

- **spec.dataImportCronTemplates.spec.template.spec.storage.storageClassName** specifies the storage class.
- **spec.dataImportCronTemplates.spec.schedule** is a required field that specifies the schedule for the job in cron format.
- **spec.dataImportCronTemplates.spec.managedDataSource** is a required field that specifies the data source to use.



NOTE

For the custom image to be detected as an available boot source, the value of the **spec.dataVolumeTemplates.spec.sourceRef.name** parameter in the VM template must match this value.

3. Wait for the HyperConverged Operator (HCO) and Scheduling, Scale, and Performance (SSP) resources to complete reconciliation.
4. Delete any outdated **DataVolume** and **VolumeSnapshot** objects from the **openshift-virtualization-os-images** namespace by running the following command.

```
$ oc delete DataVolume,VolumeSnapshot -n openshift-virtualization-os-images --
selector=cdi.kubevirt.io/dataImportCron
```

5. Wait for all **DataSource** objects to reach a "Ready - True" status. Data sources can reference either a PersistentVolumeClaim (PVC) or a VolumeSnapshot. To check the expected source format, run the following command:

```
$ oc get storageprofile <storage_class_name> -o json | jq
.status.dataImportCronSourceFormat
```

9.3.4. Enabling automatic updates for custom boot sources

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

Prerequisites

- The cluster has a default storage class.
- You have installed the OpenShift CLI (**oc**).

Procedure

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

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

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

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      name: centos-stream9-image-cron
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true"
    spec:
      schedule: "0 */12 * * *"
      template:
        spec:
          source:
            registry:
              url: docker://quay.io/containerdisks/centos-stream:9
          storage:
            resources:
              requests:
                storage: 30Gi
      garbageCollect: Outdated
      managedDataSource: centos-stream9
```

- **spec.dataImportCronTemplates.metadata.annotations** specifies a required annotation for storage classes with **volumeBindingMode** set to **WaitForFirstConsumer**.
- **spec.dataImportCronTemplates.spec.schedule** specifies the schedule for the job, specified in cron format.
- **spec.dataImportCronTemplates.spec.template.spec.source.registry** specifies the registry source to use to create a data volume. Use the default **pod pullMethod** and not **node pullMethod**, which is based on the **node** docker cache. The **node** docker cache is useful when a registry image is available via **Container.Image**, but the CDI importer is not authorized to access it.
- **spec.dataImportCronTemplates.spec.managedDataSource** specifies the name of the managed data source. For the custom image to be detected as an available boot source, the name of the image's **managedDataSource** must match the name of the template's

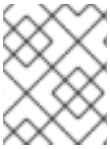
DataSource, which is found under **spec.dataVolumeTemplates.spec.sourceRef.name** in the VM template YAML file.

3. Save the file.

9.3.5. Enabling volume snapshot boot sources

You can enable volume snapshot boot sources by setting the parameter in the **StorageProfile** associated with the storage class that stores operating system base images.

Although **DataImportCron** was originally designed to maintain only PVC sources, **VolumeSnapshot** sources scale better than PVC sources for certain storage types.



NOTE

Use volume snapshots on a storage profile that is proven to scale better when cloning from a single snapshot.

Prerequisites

- You must have access to a volume snapshot with the operating system image.
- The storage must support snapshotting.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Open the storage profile object that corresponds to the storage class used to provision boot sources by running the following command:

```
$ oc edit storageprofile <storage_class>
```

2. Review the **dataImportCronSourceFormat** specification of the **StorageProfile** to confirm whether or not the VM is using PVC or volume snapshot by default.

3. Edit the storage profile, if needed, by updating the **dataImportCronSourceFormat** specification to **snapshot**.
   Example storage profile:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
# ...
spec:
  dataImportCronSourceFormat: snapshot
```

Verification

1. Open the storage profile object that corresponds to the storage class used to provision boot sources.

```
$ oc get storageprofile <storage_class>  -oyaml
```

2. Confirm that the **dataImportCronSourceFormat** specification of the **StorageProfile** is set to 'snapshot', and that any **DataSource** objects that the **DataImportCron** points to now reference volume snapshots.

You can now use these boot sources to create virtual machines.

9.3.6. Disabling automatic updates for a single boot source

You can disable automatic updates for an individual boot source, whether it is custom or system-defined, by editing the **HyperConverged** custom resource (CR).

Prerequisites

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

Procedure

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

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

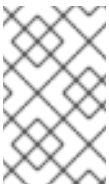
2. Disable automatic updates for an individual boot source by editing the **spec.dataImportCronTemplates** field.

   Custom boot source

   - Remove the boot source from the **spec.dataImportCronTemplates** field. Automatic updates are disabled for custom boot sources by default.

   System-defined boot source

   a. Add the boot source to **spec.dataImportCronTemplates**.

   NOTE

   Automatic updates are enabled by default for system-defined boot sources, but these boot sources are not listed in the CR unless you add them.

   b. Set the value of the **dataimportcrontemplate.kubevirt.io/enable** annotation to **'false'**. For example:

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

3. Save the file.

9.3.7. Increasing boot source disk image retention

You can configure image retention settings to increase the number of older operating system image versions preserved on the cluster.

When automatic boot source updates are enabled, the Containerized Data Importer (CDI) tracks and downloads the latest versions of operating system images. By default, the system aggressively minimizes the retention of older versions to conserve disk space. However, if you require a safety mechanism that allows you to roll back if a newly imported version introduces issues, you can increase the retention count.



NOTE

Manually deleting older **PersistentVolumeClaim** or **DataVolume** objects associated with historic boot source imports does not impact cluster stability or future updates.

Procedure

1. Open the **HyperConverged** custom resource (CR) in your default editor:

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

2. Edit the **spec.dataImportCronTemplates** field to adjust the **importsToKeep** parameter to your preferred retention threshold:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
    - metadata:
        name: rhel9-image-cron
      spec:
        garbageCollect: Outdated (1)
        importsToKeep: 3
        schedule: "0 */12 * * *"
        managedDataSource: rhel9
```

9.3.8. Verifying the status of a boot source

You can determine if a boot source is system-defined or custom by viewing the **HyperConverged** custom resource (CR).

Prerequisites

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

Procedure

1. View the contents of the **HyperConverged** CR by running the following command:

```
$ oc get hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
-o yaml
```

Example output:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
# ...
status:
# ...
  dataImportCronTemplates:
  - metadata:
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true"
      name: centos-9-image-cron
    spec:
      garbageCollect: Outdated
      managedDataSource: centos-stream9
      schedule: 55 8/12 * * *
      template:
        metadata: {}
        spec:
          source:
            registry:
              url: docker://quay.io/containerdisks/centos-stream:9
          storage:
            resources:
              requests:
                storage: 30Gi
        status: {}
    status:
      commonTemplate: true
# ...
  - metadata:
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true"
      name: user-defined-dic
    spec:
      garbageCollect: Outdated
      managedDataSource: user-defined-centos-stream9
      schedule: 55 8/12 * * *
      template:
        metadata: {}
        spec:
          source:
            registry:
              pullMethod: node
              url: docker://quay.io/containerdisks/centos-stream:9
          storage:
            resources:
              requests:
                storage: 30Gi
        status: {}
```

```
    status: {}
# ...
```

- **status.dataImportCronTemplates.status.commonTemplate** specifies a system-defined boot source.
- **status.dataImportCronTemplates.status** specifies a custom boot source.

2. Verify the status of the boot source by reviewing the **status.dataImportCronTemplates.status** field.
   - If the field contains **commonTemplate: true**, it is a system-defined boot source.
   - If the **status.dataImportCronTemplates.status** field has the value **{}**, it is a custom boot source.

9.3.9. Additional resources

- All Red Hat boot sources
- All custom boot sources
- Individual Red Hat or custom boot sources

9.4. RESERVING PVC SPACE FOR FILE SYSTEM OVERHEAD

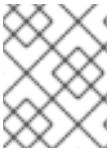
When you create a **DataVolume** custom resource (CR) for a virtual machine (VM) by setting the **spec.storage.volumeMode** attribute to **Filesystem**, OpenShift Virtualization automatically accounts for file system overhead.

If you specify the storage type by using the **spec.pvc** attribute in the **DataVolume** CR, OpenShift Virtualization does not add any file system overhead and the requested size is passed directly to Kubernetes.

The default file system overhead value is 6%. For example, if you request a 10 GiB disk and the **spec.storage.volumeMode** attribute is set to **FileSystem**, Kubernetes provisions a PVC of approximately 10.6 GiB so that the VM has the full 10 GiB of space available.

Table 9.1. Example file system overhead for data volumes

Requested virtual disk size	Calculated overhead (6%)	Total PVC space provisioned
10 GiB	0.6 GiB	10.6 GiB
100 GiB	6 GiB	106 GiB



NOTE

You can change the default file system overhead value by editing the **HyperConverged** CR.

9.4.1. Overriding the default file system overhead value

Change the amount of persistent volume claim (PVC) space that the OpenShift Virtualization reserves for file system overhead by editing the **spec.filesystemOverhead** attribute of the **HCO** object.

Prerequisites

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

Procedure

1. Open the **HCO** object for editing by running the following command:

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

2. Edit the **spec.filesystemOverhead** fields, populating them with your chosen values:

```
# ...
spec:
  filesystemOverhead:
    global: "<new_global_value>"
    storageClass:
      <storage_class_name>: "<new_value_for_this_storage_class>"
```

   - **spec.filesystemOverhead.global** specifies the default file system overhead percentage used for any storage classes that do not already have a set value. For example, **global: "0.07"** reserves 7% of the PVC for file system overhead.

   - **spec.filesystemOverhead.storageClass** specifies the file system overhead percentage for the specified storage class. For example, **mystorageclass: "0.04"** changes the default overhead value for PVCs in the **mystorageclass** storage class to 4%.

3. Save and exit the editor to update the **HCO** object.

Verification

- View the **CDIConfig** status and verify your changes by running one of the following commands: To generally verify changes to **CDIConfig**:

```
$ oc get cdiconfig -o yaml
```

  To view your specific changes to **CDIConfig**:

```
$ oc get cdiconfig -o jsonpath='{.items..status.filesystemOverhead}'
```

9.5. CONFIGURING LOCAL STORAGE BY USING THE HOSTPATH PROVISIONER

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

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

9.5.1. Creating a hostpath provisioner with a basic storage pool

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



IMPORTANT

Do not create storage pools in the same partition as the operating system. Otherwise, the operating system partition might become filled to capacity, which will impact performance or cause the node to become unstable or unusable.

Prerequisites

- The directories specified in **spec.storagePools.path** must have read/write access.
- You have installed the OpenShift CLI (**oc**).

Procedure

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

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

   - **spec.storagePools.name** defines the name to identify the source to use. It must be the same as the **storagePools** name in the **StorageClass.yaml**.
   - **spec.storagePools.path** defines the storage pool directories under this node path. Ensure that the path **/var/myvolumes** value specifies a directory that exists on each worker node.

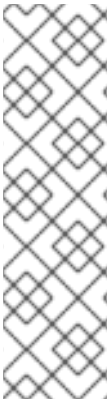
2. Save the file and exit.
3. Create the HPP by running the following command:

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

9.5.1.1. About creating storage classes

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

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



NOTE

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

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

9.5.1.2. Create a storage class for the CSI driver with the storagePools stanza

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



NOTE

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

Prerequisites

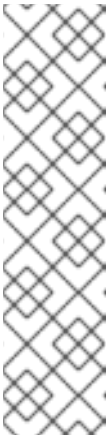
- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.

Procedure

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

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

   - **reclaimPolicy** defines whether the underlying storage is deleted or retained when a user deletes a PVC. The two possible **reclaimPolicy** values are **Delete** and **Retain**. If you do not specify a value, the default value is **Delete**.
   - **volumeBindingMode** defines the timing of PV creation. In this example, the **WaitForFirstConsumer** configuration delays PV creation until the scheduler assigns a pod to a specific node.



NOTE

Virtual machines use data volumes based on local PVs, which reside on specific nodes. When the system prepares a disk image for the virtual machine, the scheduler might not place the virtual machine on the node where it pinned the local storage PV.

+ To solve this problem, use the Kubernetes pod scheduler to bind the persistent volume claim (PVC) to a PV on the correct node. Setting the **volumeBindingMode** parameter of the **StorageClass** to **WaitForFirstConsumer** delays PV binding and provisioning until you create a pod that uses the PVC.

- **parameters.storagePool** defines the name of the storage pool defined in the HPP custom resource (CR).

2. Save the file and exit.

3. Create the **StorageClass** object by running the following command:

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

9.5.2. About storage pools created with PVC templates

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

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

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

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

The **spec.volumeMode** value is only required for block volume mode PVs.

You define a storage pool by using a **pvcTemplate** specification in the HPP CR. The Operator creates a PVC from the **pvcTemplate** specification for each node containing the HPP CSI driver. The PVC created from the PVC template consumes the single large PV, allowing the HPP to create smaller dynamic volumes.

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

9.5.2.1. Creating a storage pool with a PVC template

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



IMPORTANT

Do not create storage pools in the same partition as the operating system. Otherwise, the operating system partition might become filled to capacity, which will impact performance or cause the node to become unstable or unusable.

Prerequisites

- The directories specified in **spec.storagePools.path** must have read/write access.
- You have installed the OpenShift CLI (**oc**).

Procedure

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

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

- **spec.storagePools** defines an array that can contain both basic and PVC template storage pools.
- **spec.storagePools.path** defines the storage pool directories under this node path.
- **spec.storagePools.pvcTemplate.volumeMode** is an optional parameter, which can be either **Block** or **Filesystem** if it matches the provisioned volume format. If no value is specified, the default is **Filesystem**. If the **volumeMode** is **Block**, the mounting pod creates an XFS file system on the block volume before mounting it.

- **spec.storagePools.pvcTemplate.storageClassName** specifies if the **storageClassName** parameter is omitted, the default storage class is used to create PVCs. If you omit **storageClassName**, ensure that the HPP storage class is not the default storage class.
- **spec.storagePools.pvcTemplate.resources.requests.storage** defines the storage request for statically or dynamically provisioned storage. Ensure the requested storage size is appropriate for the volume you want to virtually divide or the PVC cannot be bound to the large PV. If the storage class uses dynamically provisioned storage, pick an allocation size that matches the size of a typical request.

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

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

9.6. ENABLING USER PERMISSIONS TO CLONE DATA VOLUMES ACROSS NAMESPACES

By default, users cannot clone resources between namespaces. To enable cloning, a user with the **cluster-admin** role must create and bind a cluster role that grants the required permissions.

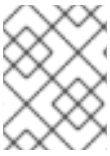
To enable a user to clone a virtual machine to another namespace, a user with the **cluster-admin** role must create a new cluster role. Bind this cluster role to a user to enable them to clone virtual machines to the destination namespace.

9.6.1. Creating RBAC resources for cloning data volumes

You can create a new cluster role that enables permissions for all actions for the **datavolumes** resource.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You must have cluster admin privileges.



NOTE

If you are a non-admin user that is an administrator for both the source and target namespaces, you can create a **Role** instead of a **ClusterRole** where appropriate.

Procedure

1. Create a **ClusterRole** manifest:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: <datavolume_cloner>
rules:
- apiGroups: ["cdi.kubevirt.io"]
  resources: ["datavolumes/source"]
  verbs: ["*"]
# ...
```

where:

<datavolume_cloner>

Specifies a unique name for the cluster role.

2. Create the cluster role in the cluster:

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

where:

<datavolume_cloner.yaml>

Specifies the file name of the **ClusterRole** manifest created in the previous step.

3. Create a **RoleBinding** manifest that applies to both the source and destination namespaces and references the cluster role created in the previous step.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: <allow_clone_to_user>
  namespace: <source_namespace>
subjects:
- kind: ServiceAccount
  name: default
  namespace: <destination_namespace>
roleRef:
  kind: ClusterRole
  name: datavolume-cloner
  apiGroup: rbac.authorization.k8s.io
```

- **metadata.name** specifies a unique name for the role binding.
- **metadata.namespace** specifies the namespace for the source data volume.
- **subjects.namespace** specifies the namespace to which the data volume is cloned.
- **roleRef.name** specifies the name of the cluster role created in the previous step.

4. Create the role binding in the cluster:

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

where:

<datavolume_cloner.yaml>

Specifies the file name of the **RoleBinding** manifest created in the previous step.

9.7. CONFIGURING CDI TO OVERRIDE CPU AND MEMORY QUOTAS

You can configure the Containerized Data Importer (CDI) to import, upload, and clone virtual machine disks into namespaces that are subject to CPU and memory resource restrictions.

9.7.1. About CPU and memory quotas in a namespace

A *resource quota*, defined by the **ResourceQuota** object, imposes restrictions on a namespace that limit the total amount of compute resources that can be consumed by resources within that namespace.

The **HyperConverged** custom resource (CR) defines the user configuration for the Containerized Data Importer (CDI). The CPU and memory request and limit values are set to a default value of **0**. This ensures that pods created by CDI that do not specify compute resource requirements are given the default values and are allowed to run in a namespace that is restricted with a quota.

9.7.2. Overriding CPU and memory defaults

Modify the default settings for CPU and memory requests and limits for your use case by adding the **spec.resourceRequirements.storageWorkloads** stanza to the **HyperConverged** custom resource (CR).

Prerequisites

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

Procedure

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

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

2. Add the **spec.resourceRequirements.storageWorkloads** stanza to the CR, setting the values based on your use case. For example:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  resourceRequirements:
    storageWorkloads:
      limits:
        cpu: "500m"
        memory: "2Gi"
      requests:
        cpu: "250m"
        memory: "1Gi"
```

3. Save and exit the editor to update the **HyperConverged** CR.

9.7.3. Additional resources

- Resource quotas per project

9.8. PREPARING CDI SCRATCH SPACE

To support image import and processing, configure the Containerized Data Importer (CDI) scratch space and the required storage class so that CDI can temporarily store and convert virtual machine (VM) images.

9.8.1. About scratch space

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

The scratch space PVC is deleted after the operation completes or aborts.

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

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



NOTE

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

9.8.1.1. Manual provisioning

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

9.8.2. CDI operations that require scratch space

To import and process virtual machine (VM) images, the Containerized Data Importer (CDI) uses scratch space as temporary storage during specific operations such as registry imports and image uploads.

Type	Reason
Registry imports	CDI must download the image to a scratch space and extract the layers to find the image file. The image file is then passed to QEMU-IMG for conversion to a raw disk.
Upload image	QEMU-IMG does not accept input from STDIN. Instead, the image to upload is saved in scratch space before it can be passed to QEMU-IMG for conversion.

Type	Reason
HTTP imports of archived images	QEMU-IMG does not know how to handle the archive formats CDI supports. Instead, the image is unarchived and saved into scratch space before it is passed to QEMU-IMG.
HTTP imports of authenticated images	QEMU-IMG inadequately handles authentication. Instead, the image is saved to scratch space and authenticated before it is passed to QEMU-IMG.
HTTP imports of custom certificates	QEMU-IMG inadequately handles custom certificates of HTTPS endpoints. Instead, CDI downloads the image to scratch space before passing the file to QEMU-IMG.

9.8.3. Defining a storage class

You can define the storage class that the Containerized Data Importer (CDI) uses when allocating scratch space by adding the **spec.scratchSpaceStorageClass** field to the **HyperConverged** custom resource (CR).

Prerequisites

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

Procedure

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

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

2. Add the **spec.scratchSpaceStorageClass** field to the CR and set the value to the name of a storage class that exists in the cluster. If you do not specify a storage class, CDI uses the storage class of the persistent volume claim that is being populated.

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  scratchSpaceStorageClass: "<storage_class>"
```

3. Save and exit your default editor to update the **HyperConverged** CR.

9.8.4. CDI supported operations matrix

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

Content types	HTTP	HTTPS	Basic HTTP authentication	Registry	Upload
KubeVirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* ☐ GZ ☐ XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (raw)	✓ raw ✓ GZ ✓ XZ	✓ raw ✓ GZ ✓ XZ	✓ raw ✓ GZ ✓ XZ	✓ raw* ☐ GZ ☐ XZ	✓ raw* ✓ GZ* ✓ XZ*

✓	Supported operation
☐	Unsupported operation
*	Requires scratch space
**	Requires scratch space if a custom certificate authority is required

9.8.5. Additional resources

- Dynamic provisioning

9.9. USING PREALLOCATION FOR DATA VOLUMES

The Containerized Data Importer (CDI) can preallocate disk space to improve write performance when creating data volumes. You can enable preallocation for specific data volumes.

9.9.1. About preallocation

The Containerized Data Importer (CDI) can use the QEMU preallocate mode for data volumes to improve write performance. You can use preallocation mode for importing and uploading operations and when creating blank data volumes.

If preallocation is enabled, CDI uses the better preallocation method depending on the underlying file system and device type:

fallocate

If the file system supports it, CDI uses the operating system's **fallocate** call to preallocate space by using the **posix_fallocate** function, which allocates blocks and marks them as uninitialized.

full

If **fallocate** mode cannot be used, **full** mode allocates space for the image by writing data to the underlying storage. Depending on the storage location, all the empty allocated space might be zeroed.

9.9.2. Enabling preallocation for a data volume

You can enable preallocation for specific data volumes by including the **spec.preallocation** field in the data volume manifest. You can enable preallocation mode in either the web console or by using the OpenShift CLI (**oc**).

Preallocation mode is supported for all CDI source types. However, preallocation is ignored for cloning operations.

Procedure

- Specify the **spec.preallocation** field in the data volume manifest:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: preallocated-datavolume
spec:
  source:
    registry:
      url: <image_url>
  storage:
    resources:
      requests:
        storage: 1Gi
  preallocation: true
# ...
```

where:

<image_url>

Specifies the URL of the data source in your registry.

9.10. MANAGING DATA VOLUME ANNOTATIONS

Data volume (DV) annotations allow you to manage pod behavior. You can add one or more annotations to a data volume, which then propagates to the created importer pods.

9.10.1. Example: Data volume annotations

You can configure data volume (DV) annotations to control which network the importer pod uses. The **v1.multus-cni.io/default-network: bridge-network** annotation causes the pod to use the Multus network named **bridge-network** as its default network.

If you want the importer pod to use both the default network from the cluster and the secondary Multus network, use the **k8s.v1.cni.cncf.io/networks: <network_name>** annotation.

Multus network annotation example:

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: datavolume-example
```

```
  annotations:
    v1.multus-cni.io/default-network: bridge-network
# ...
```

The **v1.multus-cni.io/default-network** annotation specifies the Multus network name.

CHAPTER 10. LIVE MIGRATION

10.1. ABOUT LIVE MIGRATION

Live migration moves a running virtual machine to another node without interrupting the workload. It enables smooth transitions during cluster upgrades and node maintenance. Traffic is encrypted using TLS by default.

10.1.1. Live migration requirements

Live migration requires shared storage, sufficient resources, and compatible CPUs across nodes.

Live migration requirements

- Shared storage that supports live migration.
- Sufficient RAM and network bandwidth.



NOTE

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

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

The default number of migrations that can run in parallel in the cluster is 5. For more information, see "Configuring live migration" in the Additional resources section.

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

10.1.2. Live migration permissions

In OpenShift Virtualization 4.19 and later, live migration operations are restricted to users who are explicitly granted the **kubevirt.io:migrate** cluster role. Users with this role can create, delete, and update virtual machine (VM) live migration requests.

The live migration requests are represented by **VirtualMachineInstanceMigration** (VMIM) custom resources. Cluster administrators can bind the **kubevirt.io:migrate** role to trusted users or groups at either the namespace or cluster level.

Before OpenShift Virtualization 4.19, namespace administrators had live migration permissions by default. This behavior changed in version 4.19 to prevent unintended or malicious disruptions to infrastructure-critical migration operations.

As a cluster administrator, you can preserve the old behavior by creating a temporary cluster role before updating. After assigning the new role to users, delete the temporary role to enforce the more restrictive permissions. If you have already updated, you can still revert to the old behavior by aggregating the **kubevirt.io:migrate** role into the **admin** cluster role.

10.1.3. Preserve pre-4.19 live migration permissions during update

Before you update to OpenShift Virtualization 4.22, you can create a temporary cluster role to preserve the previous live migration permissions until you are ready for the more restrictive default permissions to take effect.

Prerequisites

- The OpenShift CLI (**oc**) is installed.
- You have cluster administrator permissions.

Procedure

1. Before updating to OpenShift Virtualization 4.22, create a temporary **ClusterRole** object. For example:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    rbac.authorization.k8s.io/aggregate-to-admin=true
  name: kubevirt.io:upgrademigrate
rules:
- apiGroups:
  - subresources.kubevirt.io
  resources:
  - virtualmachines/migrate
  verbs:
  - update
- apiGroups:
  - kubevirt.io
  resources:
  - virtualmachineinstancemigrations
  verbs:
  - get
  - delete
  - create
  - update
  - patch
  - list
  - watch
  - deletecollection
```

   This cluster role is aggregated into the **admin** role before you update OpenShift Virtualization. The update process does not modify it, ensuring the previous behavior is maintained.

2. Add the cluster role manifest to the cluster by running the following command:

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

3. Update OpenShift Virtualization to version 4.22.

4. Bind the **kubevirt.io:migrate** cluster role to trusted users or groups by running one of the following commands, replacing **<namespace>**, **<first_user>**, **<second_user>**, and **<group_name>** with your own values.

   - To bind the role at the namespace level, run the following command:

     ```
     $ oc create -n <namespace> rolebinding kvmigrate --clusterrole=kubevirt.io:migrate --user=<first_user> --user=<second_user> --group=<group_name>
     ```

   - To bind the role at the cluster level, run the following command:

     ```
     $ oc create clusterrolebinding kvmigrate --clusterrole=kubevirt.io:migrate --user=<first_user> --user=<second_user> --group=<group_name>
     ```

5. When you have bound the **kubevirt.io:migrate** role to all necessary users, delete the temporary **ClusterRole** object by running the following command:

   ```
   $ oc delete clusterrole kubevirt.io:upgrademigrate
   ```

   After you delete the temporary cluster role, only users with the **kubevirt.io:migrate** role can create, delete, and update live migration requests.

10.1.4. Grant live migration permissions

You can grant trusted users or groups the ability to create, delete, and update live migration instances.

Prerequisites

- The OpenShift CLI (**oc**) is installed.
- You have cluster administrator permissions.

Procedure

- (Optional) To change the default behavior so that namespace administrators always have permission to create, delete, and update live migrations, aggregate the **kubevirt.io:migrate** role into the **admin** cluster role by running the following command:

  ```
  $ oc label --overwrite clusterrole kubevirt.io:migrate rbac.authorization.k8s.io/aggregate-to-admin=true
  ```

- Bind the **kubevirt.io:migrate** cluster role to trusted users or groups by running one of the following commands, replacing **<namespace>**, **<first_user>**, **<second_user>**, and **<group_name>** with your own values.

  - To bind the role at the namespace level, run the following command:

    ```
    $ oc create -n <namespace> rolebinding kvmigrate --clusterrole=kubevirt.io:migrate --user=<first_user> --user=<second_user> --group=<group_name>
    ```

  - To bind the role at the cluster level, run the following command:

```
$ oc create clusterrolebinding kvmigrate --clusterrole=kubevirt.io:migrate --user=
<first_user> --user=<second_user> --group=<group_name>
```

10.1.5. VM migration tuning

You can adjust your cluster-wide live migration settings based on the type of workload and migration scenario.

This enables you to control how many VMs migrate at the same time, the network bandwidth you want to use for each migration, and how long OpenShift Virtualization attempts to complete the migration before canceling the process. Configure these settings in the **HyperConverged** custom resource (CR).

If you are migrating multiple VMs per node at the same time, set a **bandwidthPerMigration** limit to prevent a large or busy VM from using a large portion of the node's network bandwidth. By default, the **bandwidthPerMigration** value is **0**, which means unlimited.

A large VM running a heavy workload (for example, database processing), with higher memory dirty rates, requires a higher bandwidth to complete the migration.



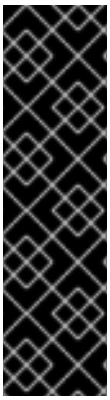
NOTE

Post copy mode, when enabled, triggers if the initial pre-copy phase does not complete within the defined timeout. During post copy, the VM CPUs pause on the source host while transferring the minimum required memory pages. Then the VM CPUs activate on the destination host, and the remaining memory pages transfer into the destination node at runtime. This can impact performance during the transfer.

Post copy mode should not be used for critical data, or with unstable networks.

10.1.6. VM migration support for RHCOS 10.x

{VirtProductName} 4.22 and later supports live migration to, from, and between {op-system} 10.x worker nodes as a Technology Preview feature. This includes clusters that run a mix of {op-system} 9.x and 10.x worker nodes.



IMPORTANT

Live migration support for RHCOS 10.x is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

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

For information on configuring your cluster to use RHCOS 10.x, refer to the OpenShift Dedicated documentation.

When performing live migration on a cluster using RHCOS 10.x, the migration does not complete successfully when the migration policy uses the attribute **allowPostCopy: true**. This is a known limitation.

Live migration is supported across both RHCOS 9.x and 10.x worker nodes when both versions are present in a cluster. Any VM live migration from RHCOS 10.x to RHCOS 9.x and from RHCOS 9.x to RHCOS 10.x worker nodes, is a Technology Preview feature in OpenShift Dedicated 4.22.

10.1.7. Additional resources

- Default cluster roles for OpenShift Virtualization
- Prometheus queries for live migration
- Configure eviction and run strategies

10.2. CONFIGURING LIVE MIGRATION

You can configure live migration settings to ensure that the migration processes do not overwhelm the cluster. You can configure live migration policies to apply different migration configurations to groups of virtual machines (VMs).

10.2.1. Configure live migration limits and timeouts

Configure live migration limits and timeouts for the cluster by updating the **HyperConverged** custom resource (CR), which is located in the **openshift-cnv** namespace.

Prerequisites

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

Procedure

- Edit the **HyperConverged** CR and add the necessary live migration parameters:

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

Example configuration file:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  liveMigrationConfig:
    bandwidthPerMigration: 2048Mi
    completionTimeoutPerGiB: 800
    parallelMigrationsPerCluster: 5
    parallelOutboundMigrationsPerNode: 2
    progressTimeout: 150
    allowPostCopy: false
```

where:

bandwidthPerMigration

Specifies the bandwidth of each migration in bytes per second. For example, a value of 2048Mi means 2048 MiB/s. Default: 0, which is unlimited.

completionTimeoutPerGiB

Specifies the length of time, in seconds per GiB of memory, at which the migration is canceled if it has not completed. For example, a VM with 6GiB memory times out if it has not completed migration in 4800 seconds. If the **Migration Method** is **BlockMigration**, the size of the migrating disks is included in the calculation.

parallelMigrationsPerCluster

Specifies the number of migrations running in parallel in the cluster. Default: **5**.

parallelOutboundMigrationsPerNode

Specifies the maximum number of outbound migrations per node. Default: **2**.

progressTimeout

Specifies the length of time, in seconds, at which the migration is canceled if memory copy fails to make progress. Default: **150**.

allowPostCopy

Specifies whether the post copy mode is enabled. You can enable post copy mode to allow the migration of one node to another to converge, even if a VM is running a heavy workload and the memory dirty rate is too high. By default, **allowPostCopy** is set to **false**.



NOTE

You can restore the default value for any **spec.liveMigrationConfig** field by deleting that key/value pair and saving the file. For example, delete **progressTimeout: <value>** to restore the default **progressTimeout: 150**.

10.2.2. Configure live migration for heavy workloads

When migrating a VM running a heavy workload, such as database processing, higher memory dirty rates can prevent the migration from completing. To address this, you can enable post copy mode, which allows the migration to complete when the pre-copy phase cannot converge.

Configure live migration for heavy workloads by updating the **HyperConverged** custom resource (CR) in the **openshift-cnv** namespace.

Prerequisites

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

Procedure

1. Edit the **HyperConverged** CR and add the necessary parameters for migrating heavy workloads:

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

Example configuration file:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
```

```
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  liveMigrationConfig:
    bandwidthPerMigration: 0Mi
    completionTimeoutPerGiB: 150
    parallelMigrationsPerCluster: 5
    parallelOutboundMigrationsPerNode: 1
    progressTimeout: 150
    allowPostCopy: true
```

where:

bandwidthPerMigration

Specifies the bandwidth of each migration in bytes per second. The default is **0**, which is unlimited.

completionTimeoutPerGiB

Specifies the length of time, in seconds per GiB of memory, at which the migration is canceled if it has not completed and post copy mode is triggered, if enabled. You can lower **completionTimeoutPerGiB** to trigger post copy mode earlier in the migration process, or raise the **completionTimeoutPerGiB** to trigger post copy mode later in the migration process.

parallelMigrationsPerCluster

Specifies the number of migrations running in parallel in the cluster. The default is **5**. Keeping the **parallelMigrationsPerCluster** setting low is better when migrating heavy workloads.

parallelOutboundMigrationsPerNode

Specifies the maximum number of outbound migrations per node. Configure a single VM per node for heavy workloads.

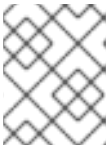
progressTimeout

Specifies the length of time, in seconds, at which the migration is canceled if memory copy fails to make progress. Increase this parameter for large memory sizes running heavy workloads.

allowPostCopy

Specifies whether the post copy mode is enabled. You can enable post copy mode to allow the migration of one node to another to converge, even if a VM is running a heavy workload and the memory dirty rate is too high. Set allowPostCopy to true to enable post copy mode.

2. Optional: If your main network is too busy for the migration, configure a secondary, dedicated migration network.



NOTE

Post copy mode can impact performance during the transfer, and should not be used for critical data, or with unstable networks.

10.2.3. Live migration policies

You can create live migration policies to apply different migration configurations to groups of VMs that are defined by VM or project labels.

TIP

You can create live migration policies by using the OpenShift Dedicated web console.

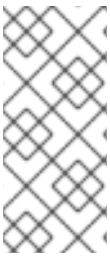
10.2.4. Create a live migration policy by using the CLI

You can create a live migration policy by using the command line.

KubeVirt applies the live migration policy to selected virtual machines (VMs) by using any combination of labels:

- VM labels such as **size**, **os**, or **gpu**
- Project labels such as **priority**, **bandwidth**, or **hpc-workload**

For the policy to apply to a specific group of VMs, all labels on the group of VMs must match the labels of the policy.



NOTE

If multiple live migration policies apply to a VM, the policy with the greatest number of matching labels takes precedence.

If multiple policies meet this criteria, the policies are sorted by alphabetical order of the matching label keys, and the first one in that order takes precedence.

Prerequisites

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

Procedure

1. Edit the VM object to which you want to apply a live migration policy, and add the corresponding VM labels.
   a. Open the YAML configuration of the resource:

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

   b. Adjust the required label values in the **.spec.template.metadata.labels** section of the configuration. For example, to mark the VM as a **production** VM for the purposes of migration policies, add the **kubevirt.io/environment: production** line:

```
apiVersion: migrations.kubevirt.io/v1alpha1
kind: VirtualMachine
metadata:
  name: <vm_name>
  namespace: default
  labels:
    app: my-app
    environment: production
spec:
  template:
    metadata:
      labels:
```

```
        kubevirt.io/domain: <vm_name>
        kubevirt.io/size: large
        kubevirt.io/environment: production
# ...
```

c. Save and exit the configuration.

2. Configure a **MigrationPolicy** object with the corresponding labels. The following example configures a policy that applies to all VMs that are labeled as **production**:

```
apiVersion: migrations.kubevirt.io/v1alpha1
kind: MigrationPolicy
metadata:
  name: <migration_policy>
spec:
  selectors:
    namespaceSelector:
      hpc-workloads: "True"
      xyz-workloads-type: ""
    virtualMachineInstanceSelector:
      kubevirt.io/environment: "production"
```

where:

namespaceSelector

Specifies the project labels.

virtualMachineInstanceSelector

Specifies the VM labels.

3. Create the migration policy by running the following command:

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

10.3. INITIATING AND CANCELING LIVE MIGRATION

To move a running virtual machine (VM) to a different node without interrupting the workload, you can initiate a live migration. You can also cancel an ongoing migration to keep the VM on its original node.

You can initiate the live migration of a virtual machine (VM) to another node by using the OpenShift Dedicated web console or the command line.

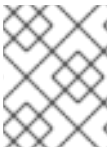
You can cancel a live migration by using the web console or the command line. The VM remains on its original node.

TIP

You can also initiate and cancel live migration by using the **virtctl migrate <vm_name>** and **virtctl migrate-cancel <vm_name>** commands.

10.3.1. Start live migration by using the web console

You can live migrate a running virtual machine (VM) to a different node in the cluster by using the OpenShift Dedicated web console.



NOTE

The **Migrate** action is visible to all users but only cluster administrators can initiate a live migration.

Prerequisites

- You have the **kubevirt.io:migrate** RBAC role or you are a cluster administrator.
- The VM is able to be migrated.
- If the VM is configured with a host model CPU, the cluster has an available node that supports the CPU model.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Take either of the following steps:
   - Click the Options menu  beside the VM you want to migrate, hover over the **Migrate** option, and select **Compute**.
   - Open the **VM details** page of the VM you want to migrate, click the **Actions** menu, hover over the **Migrate** option, and select **Compute**.
3. In the **Migrate Virtual Machine to a different Node** dialog box, select either **Automatically Selected Node** or **Specific Node**.
   a. If you selected the **Specific Node** option, choose a node from the list.
4. Click **Migrate Virtual Machine**

10.3.2. Start live migration by using the CLI

You can initiate the live migration of a running virtual machine (VM) by using the command line to create a **VirtualMachineInstanceMigration** object for the VM.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have the **kubevirt.io:migrate** RBAC role or you are a cluster administrator.

Procedure

1. Create a **VirtualMachineInstanceMigration** manifest for the VM that you want to migrate:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstanceMigration
metadata:
  name: <migration_name>
spec:
  vmiName: <vm_name>
```

2. Create the object by running the following command:

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

The **VirtualMachineInstanceMigration** object triggers a live migration of the VM. This object exists in the cluster only while the virtual machine instance is running, unless manually deleted.

Verification

- Obtain the VM status by running the following command:

```
$ oc describe vmi <vm_name> -n <namespace>
```

Example output:

```
# ...
Status:
  Conditions:
    Last Probe Time:       <nil>
    Last Transition Time:  <nil>
    Status:                True
    Type:                  LiveMigratable
  Migration Method:  LiveMigration
  Migration State:
    Completed:                true
    End Timestamp:            2018-12-24T06:19:42Z
    Migration UID:            d78c8962-0743-11e9-a540-fa163e0c69f1
    Source Node:              node2.example.com
    Start Timestamp:          2018-12-24T06:19:35Z
    Target Node:              node1.example.com
    Target Node Address:      10.9.0.18:43891
    Target Node Domain Detected:  true
```

10.3.3. Cancel live migration by using the web console

You can cancel the live migration of a virtual machine (VM) by using the OpenShift Dedicated web console.

Prerequisites

- You have the **kubevirt.io:migrate** RBAC role or you are a cluster administrator.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Select **Cancel Migration** on the Options menu beside a VM.

10.3.4. Cancel live migration by using the CLI

Cancel the live migration of a virtual machine by deleting the **VirtualMachineInstanceMigration** object associated with the migration.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have the **kubevirt.io:migrate** RBAC role or you are a cluster administrator.

Procedure

- Delete the **VirtualMachineInstanceMigration** object that triggered the live migration, **migration-job** in this example:

```
$ oc delete vmim migration-job
```

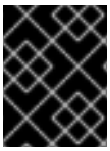
10.3.5. Additional resources

- About live migration permissions
- Initiating live migration by using the web console
- Initiating live migration by using the CLI
- Canceling live migration by using the web console
- Canceling live migration by using the CLI

CHAPTER 11. NODES

11.1. NODE MAINTENANCE MODE

Placing a node into maintenance mode marks the node as unschedulable, and removes all the VMs and pods from it. Nodes can be placed into maintenance mode by using the **oc adm** utility or **NodeMaintenance** custom resources (CRs).



IMPORTANT

Virtual machines (VMs) must use persistent volume claims (PVCs) with storage that supports live migration to be live migrated.

11.1.1. Maintaining bare-metal nodes

When you deploy OpenShift Dedicated on bare metal infrastructure, there are additional considerations that must be taken into account compared to deploying on cloud infrastructure.

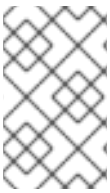
Unlike in cloud environments where the cluster nodes are considered ephemeral, re-provisioning a bare-metal node requires significantly more time and effort for maintenance tasks.

When a bare-metal node fails, for example, if a an unrecoverable kernel error happens or a NIC card hardware failure occurs, workloads on the failed node need to be restarted elsewhere else on the cluster while the problem node is repaired or replaced. Node maintenance mode allows cluster administrators to gracefully power down nodes, moving workloads to other parts of the cluster and ensuring workloads do not get interrupted. Detailed progress and node status details are provided during maintenance.

11.1.2. About the Node Maintenance Operator

The Node Maintenance Operator watches for new or deleted **NodeMaintenance** custom resources (CRs). When a new **NodeMaintenance** CR is detected, no new workloads are scheduled, and the node is cordoned off from the rest of the cluster. All pods that can be evicted are evicted from the node. When a **NodeMaintenance** CR is deleted, the node that is referenced in the CR is made available for new workloads.

Using a **NodeMaintenance** CR for node maintenance tasks achieves the same results as the **oc adm cordon** and **oc adm drain** commands using standard OpenShift Dedicated custom resource processing.



NOTE

The **node-maintenance-operator** (NMO) is no longer shipped with OpenShift Virtualization. It is deployed as a standalone Operator from the software catalog in the OpenShift Dedicated web console or by using the OpenShift CLI (**oc**).

11.1.3. Additional resources

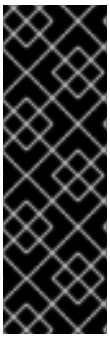
- About live migration
- About node remediation, fencing, and maintenance

11.2. CONFIGURE EVICTION AND RUN STRATEGIES

You can configure eviction strategies for virtual machines (VMs) or for the cluster. The default eviction strategy is **LiveMigrate**, which ensures that a virtual machine instance (VMI) is not interrupted if the node is placed into maintenance or drained.

11.2.1. Configure VM eviction strategy

You can configure an eviction strategy for a virtual machine (VM) by using the command line.



IMPORTANT

The default eviction strategy is **LiveMigrate**. A non-migratable VM with a **LiveMigrate** eviction strategy might prevent nodes from draining or block an infrastructure upgrade because the VM is not evicted from the node. This situation causes a migration to remain in a **Pending** or **Scheduling** state unless you shut down the VM manually.

You must set the eviction strategy of non-migratable VMs to **LiveMigrateIfPossible**, which does not block an upgrade, or to **None**, for VMs that should not be migrated.

Prerequisites

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

Procedure

1. Edit the **VirtualMachine** resource by running the following command:

```
$ oc edit vm <vm_name> -n <namespace>
```

   Example eviction strategy:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: <vm_name>
spec:
  template:
    spec:
      evictionStrategy: LiveMigrateIfPossible
# ...
```

   - **spec.template.spec.evictionStrategy** defines the eviction strategy. The default value is **LiveMigrate**.

2. Restart the VM to apply the changes:

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

11.2.2. Run strategies

The **spec.runStrategy** key determines how a VM behaves under certain conditions. This key has four possible values: **Always**, **RerunOnFailure**, **Manual**, and **Halted**.

Always

The virtual machine instance (VMI) is always present when a virtual machine (VM) is created on another node. A new VMI is created if the original stops for any reason.

RerunOnFailure

The VMI is re-created on another node if the previous instance fails. The instance is not re-created if the VM stops successfully, such as when it is shut down.



IMPORTANT

Setting **spec.runStrategy: RerunOnFailure** is an explicit command to start the VM. If the VM is intentionally stopped and you do not want it to start, do not change the run strategy value to **RerunOnFailure**.

Manual

You control the VMI state manually with the **start**, **stop**, and **restart** virtctl client commands. The VM is not automatically restarted.

Halted

No VMI is present when a VM is created.

Different combinations of the **virtctl start**, **stop** and **restart** commands affect the run strategy.

The following table describes a VM's transition between states. The first column shows the VM's initial run strategy. The remaining columns show a virtctl command and the new run strategy after that command is run.

Table 11.1. Run strategy before and after virtctl commands

Initial run strategy	Start	Stop	Restart
Always	-	Halted	Always
RerunOnFailure	RerunOnFailure	RerunOnFailure	RerunOnFailure
Manual	Manual	Manual	Manual
Halted	Always	-	-



NOTE

If a node in a cluster installed by using installer-provisioned infrastructure fails the machine health check and is unavailable, VMs with **runStrategy: Always** or **runStrategy: RerunOnFailure** are rescheduled on a new node.

11.2.3. Configure VM run strategy

You can configure a run strategy for a virtual machine (VM) by using the command line. The run strategy controls whether a VM automatically restarts after disruptions such as node failures or maintenance events.

Prerequisites

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

Procedure

- Edit the **VirtualMachine** resource by running the following command:

```
$ oc edit vm <vm_name> -n <namespace>
```

Example run strategy:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  runStrategy: Always
# ...
```

11.2.4. Additional resources

- Live migration policies
- About listing all the nodes in a cluster
- OpenShift Virtualization - Fencing and VM High Availability Guide
- How to destroy all the data from server for decommission?
- Listing all virtual machine instances using the CLI

11.3. MANAGING NODE LABELING FOR OBSOLETE CPU MODELS

You can schedule a virtual machine (VM) on a node if the VM CPU model and policy are supported by the node.

11.3.1. Node labeling for obsolete CPU models

The OpenShift Virtualization Operator uses a predefined list of obsolete CPU models to ensure that a node supports only valid CPU models for scheduled VMs.

By default, the following CPU models are eliminated from the list of labels generated for the node:

Example 11.1. Obsolete CPU models

```
"486"
Conroe
athlon
core2duo
coreduo
kvm32
kvm64
n270
pentium
pentium2
pentium3
pentiumpro
```

```
phenom
qemu32
qemu64
```

This predefined list is not visible in the **HyperConverged** CR. You cannot *remove* CPU models from this list, but you can add to the list by editing the **spec.obsoleteCPUs.cpuModels** field of the **HyperConverged** CR.

11.3.2. Configure obsolete CPU models

You can configure a list of obsolete CPU models by editing the **HyperConverged** custom resource (CR).

Procedure

- Edit the **HyperConverged** custom resource, specifying the obsolete CPU models in the **obsoleteCPUs** array. For example:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  obsoleteCPUs:
    cpuModels:
      - "<obsolete_cpu_1>"
      - "<obsolete_cpu_2>"
```

  Replace the example values in the **cpuModels** array with obsolete CPU models. Any value that you specify is added to a predefined list of obsolete CPU models. The predefined list is not visible in the CR.

11.4. PREVENTING NODE RECONCILIATION

Use **skip-node** annotation to prevent the **node-labeller** from reconciling a node.

11.4.1. Skip node labeling reconciliation

If you want the **node-labeller** to skip a node, annotate that node by using the OpenShift CLI (**oc**).

Prerequisites

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

Procedure

- Annotate the node that you want to skip by running the following command:

```
$ oc annotate node <node_name> node-labeller.kubevirt.io/skip-node=true
```

Replace **<node_name>** with the name of the relevant node to skip.

Reconciliation resumes on the next cycle after the node annotation is removed or set to false.

11.4.2. Additional resources

- Managing node labeling for obsolete CPU models

CHAPTER 12. MONITORING

12.1. MONITORING OVERVIEW

Monitor the health of your cluster and virtual machines (VMs) to have a unified operational view of your environment. This ensures high availability and optimal resource performance.

You can monitor the health of your cluster and VMs with the following tools:

Monitoring OpenShift Virtualization VM health status

View the overall health of your OpenShift Virtualization environment in the web console by navigating to the **Home → Overview** page in the OpenShift Dedicated web console. The **Status** card displays the overall health of OpenShift Virtualization based on the alerts and conditions.

Prometheus queries for virtual resources

Query vCPU, network, storage, and guest memory swapping usage and live migration progress.

VM custom metrics

Configure the **node-exporter** service to expose internal VM metrics and processes.

VM health checks

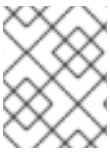
Configure readiness, liveness, and guest agent ping probes and a watchdog for VMs.

Runbooks

Diagnose and resolve issues that trigger OpenShift Virtualization alerts in the OpenShift Dedicated web console.

12.2. PROMETHEUS QUERIES FOR VIRTUAL RESOURCES

Monitor the consumption of cluster infrastructure resources by using the metrics provided by OpenShift Virtualization. These metrics are also used to query live migration status.



NOTE

- For guest memory swapping queries to return data, enable memory swapping on the virtual guests.

12.2.1. Query metrics for all projects by using the OpenShift Dedicated web console

Monitor the state of a cluster and any user-defined workloads by using the OpenShift Dedicated metrics query browser. The query browser uses Prometheus Query Language (PromQL) queries to examine metrics visualized on a plot.

As a **dedicated-admin** or as a user with view permissions for all projects, you can access metrics for all default OpenShift Dedicated and user-defined projects in the Metrics UI.



NOTE

Only dedicated administrators have access to the third-party UIs provided with OpenShift Dedicated monitoring.

Prerequisites

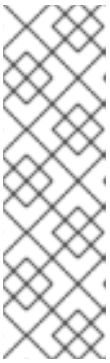
- You have access to the cluster as a user with the **dedicated-admin** role or with view permissions for all projects.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. In the OpenShift Dedicated web console, click **Observe → Metrics**.
2. To add one or more queries, perform any of the following actions:

Option	Description
Select an existing query.	From the **Select query** drop-down list, select an existing query.
Create a custom query.	Add your Prometheus Query Language (PromQL) query to the **Expression** field. As you type a PromQL expression, autocomplete suggestions appear in a drop-down list. These suggestions include functions, metrics, labels, and time tokens. Use the keyboard arrows to select one of these suggested items and then press Enter to add the item to your expression. Move your mouse pointer over a suggested item to view a brief description of that item.
Add multiple queries.	Click **Add query**.
Duplicate an existing query.	Click the options menu next to the query, then choose **Duplicate query**.
Disable a query from being run.	Click the options menu next to the query and choose **Disable query**.

3. To run queries that you created, click **Run queries**. The metrics from the queries are visualized on the plot. If a query is invalid, the UI shows an error message.



NOTE

- When drawing time series graphs, queries that operate on large amounts of data might time out or overload the browser. To avoid this, click **Hide graph** and calibrate your query by using only the metrics table. Then, after finding a feasible query, enable the plot to draw the graphs.
- By default, the query table shows an expanded view that lists every metric and its current value. Click the ˅ down arrowhead to minimize the expanded view for a query.

4. Optional: Save the page URL to use this set of queries again in the future.

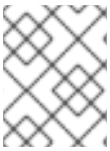
5. Explore the visualized metrics. Initially, all metrics from all enabled queries are shown on the plot. Select which metrics are shown by performing any of the following actions:

Option	Description
Hide all metrics from a query.	Click the options menu  for the query and click **Hide all series**.
Hide a specific metric.	Go to the query table and click the colored square near the metric name.
Zoom into the plot and change the time range.	Perform one of the following actions: • Visually select the time range by clicking and dragging on the plot horizontally. • Use the menu to select the time range.
Reset the time range.	Click **Reset zoom**.
Display outputs for all queries at a specific point in time.	Hover over the plot at the point you are interested in. The query outputs appear in a pop-up box.
Hide the plot.	Click **Hide graph**.

12.2.2. Query metrics for user-defined projects by using the OpenShift Dedicated web console

Monitor user-defined workloads by using the OpenShift Dedicated metrics query browser. The query browser uses Prometheus Query Language (PromQL) queries to examine metrics visualized on a plot.

To query metrics in the **Developer** perspective, you must specify a project name that represents the namespace. You must have the required privileges to view metrics for the selected project.



NOTE

Developers cannot access the third-party UIs provided with OpenShift Dedicated monitoring.

Prerequisites

- You have access to the cluster as a developer or as a user with view permissions for the project that you are viewing metrics for.
- You have enabled monitoring for user-defined projects.
- You have deployed a service in a user-defined project.

- You have created a **ServiceMonitor** custom resource definition (CRD) for the service to define how the service is monitored.

Procedure

1. In the OpenShift Dedicated web console, click **Observe → Metrics**.
2. To add one or more queries, perform any of the following actions:

Option	Description
Select an existing query.	From the **Select query** drop-down list, select an existing query.
Create a custom query.	Add your Prometheus Query Language (PromQL) query to the **Expression** field. As you type a PromQL expression, autocomplete suggestions appear in a drop-down list. These suggestions include functions, metrics, labels, and time tokens. Use the keyboard arrows to select one of these suggested items and then press Enter to add the item to your expression. Move your mouse pointer over a suggested item to view a brief description of that item.
Add multiple queries.	Click **Add query**.
Duplicate an existing query.	Click the options menu ⋮ next to the query, then choose **Duplicate query**.
Disable a query from being run.	Click the options menu ⋮ next to the query and choose **Disable query**.

3. To run queries that you created, click **Run queries**. The metrics from the queries are visualized on the plot. If a query is invalid, the UI shows an error message.



NOTE

- When drawing time series graphs, queries that operate on large amounts of data might time out or overload the browser. To avoid this, click **Hide graph** and calibrate your query by using only the metrics table. Then, after finding a feasible query, enable the plot to draw the graphs.
- By default, the query table shows an expanded view that lists every metric and its current value. Click the ˅ down arrowhead to minimize the expanded view for a query.

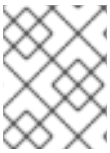
4. Optional: Save the page URL to use this set of queries again in the future.

5. Explore the visualized metrics. Initially, all metrics from all enabled queries are shown on the plot. Select which metrics are shown by performing any of the following actions:

Option	Description
Hide all metrics from a query.	Click the options menu for the query and click **Hide all series**.
Hide a specific metric.	Go to the query table and click the colored square near the metric name.
Zoom into the plot and change the time range.	Perform one of the following actions: ● Visually select the time range by clicking and dragging on the plot horizontally. ● Use the menu to select the time range.
Reset the time range.	Click **Reset zoom**.
Display outputs for all queries at a specific point in time.	Hover over the plot at the point you are interested in. The query outputs appear in a pop-up box.
Hide the plot.	Click **Hide graph**.

12.2.3. Virtualization metrics

Metric descriptions including example Prometheus Query Language (PromQL) queries. These metrics are not an API and might change between versions.



NOTE

The following examples use **topk** queries that specify a time period. If virtual machines (VMs) are deleted during that time period, they can still appear in the query output.

12.2.3.1. Network metrics

The following queries can identify virtual machines that are saturating the network:

kubevirt_vmi_network_receive_bytes_total

Returns the total amount of traffic received (in bytes) on the virtual machine's network. Type: Counter.

kubevirt_vmi_network_transmit_bytes_total

Returns the total amount of traffic transmitted (in bytes) on the virtual machine's network. Type: Counter.

Example network traffic query:

The following query returns the top 3 VMs transmitting the most network traffic at every given moment over a six-minute time period:

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_network_receive_bytes_total[6m])) + sum by (name, namespace) (rate(kubevirt_vmi_network_transmit_bytes_total[6m]))) > 0
```

12.2.3.2. Storage metrics

You can monitor virtual machine storage traffic and identify high-traffic VMs by using Prometheus queries.

The following queries can identify VMs that are writing large amounts of data:

kubevirt_vmi_storage_read_traffic_bytes_total

Returns the total amount (in bytes) of the virtual machine's storage-related traffic. Type: Counter.

kubevirt_vmi_storage_write_traffic_bytes_total

Returns the total amount of storage writes (in bytes) of the virtual machine's storage-related traffic. Type: Counter.

Example storage-related traffic queries:

- The following query returns the top 3 VMs performing the most storage traffic at every given moment over a six-minute time period:

```
topk(3, sum by (name, namespace)
(rate(kubevirt_vmi_storage_read_traffic_bytes_total[6m])) + sum by (name, namespace)
(rate(kubevirt_vmi_storage_write_traffic_bytes_total[6m]))) > 0
```

- The following query returns the top 3 VMs with the highest average read latency at every given moment over a six-minute time period:

```
topk(3, sum by (name, namespace)
(rate(kubevirt_vmi_storage_read_times_seconds_total{name='${name}',namespace='${namespace}'${clusterFilter}}[6m]) /
rate(kubevirt_vmi_storage_iops_read_total{name='${name}',namespace='${namespace}'${clusterFilter}}[6m]) > 0)) > 0
```

The following queries can track data restored from storage snapshots:

vm:kubevirt_vmsnapshot_disks_restored:sum

Returns the total number of virtual machine disks restored from the source virtual machine. Type: Gauge.

vm:kubevirt_vmsnapshot_restored_bytes:sum

Returns the amount of space in bytes restored from the source virtual machine. Type: Gauge.

Examples of storage snapshot data queries:

- The following query returns the total number of virtual machine disks restored from the source virtual machine:

```
vm:kubevirt_vmsnapshot_disks_restored:sum{vm_name="simple-vm",
vm_namespace="default"}
```

- The following query returns the amount of space in bytes restored from the source virtual machine:

```
vm:kubevirt_vmsnapshot_restored_bytes:sum{vm_name="simple-vm",
vm_namespace="default"}
```

The following queries can determine the I/O performance of storage devices:

kubevirt_vmi_storage_iops_read_total

Returns the amount of read I/O operations the virtual machine is performing per second. Type: Counter.

kubevirt_vmi_storage_iops_write_total

Returns the amount of write I/O operations the virtual machine is performing per second. Type: Counter.

Example I/O performance query:

The following query returns the top 3 VMs performing the most I/O operations per second at every given moment over a six-minute time period:

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_storage_iops_read_total[6m])) + sum by
(name, namespace) (rate(kubevirt_vmi_storage_iops_write_total[6m]))) > 0
```

12.2.3.3. Guest memory swapping metrics

The following queries can identify which swap-enabled guests are performing the most memory swapping:

kubevirt_vmi_memory_swap_in_traffic_bytes

Returns the total amount (in bytes) of memory the virtual guest is swapping in. Type: Gauge.

kubevirt_vmi_memory_swap_out_traffic_bytes

Returns the total amount (in bytes) of memory the virtual guest is swapping out. Type: Gauge.

Example memory swapping query:

The following query returns the top 3 VMs where the guest is performing the most memory swapping at every given moment over a six-minute time period:

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_memory_swap_in_traffic_bytes[6m])) + sum
by (name, namespace) (rate(kubevirt_vmi_memory_swap_out_traffic_bytes[6m]))) > 0
```



NOTE

Memory swapping indicates that the virtual machine is under memory pressure. Increasing the memory allocation of the virtual machine can mitigate this issue.

12.2.3.4. Monitoring AAQ operator metrics

The following metrics are exposed by the Application Aware Quota (AAQ) controller for monitoring resource quotas:

kube_application_aware_resourcequota

Returns the current quota usage and the CPU and memory limits enforced by the AAQ Operator resources. Type: Gauge.

kube_application_aware_resourcequota_creation_timestamp

Returns the time, in UNIX timestamp format, when the AAQ Operator resource is created. Type: Gauge.

12.2.3.5. VM label metrics

kubevirt_vm_labels

Returns virtual machine labels as Prometheus labels. Type: Gauge.
You can expose and ignore specific labels by editing the **kubevirt-vm-labels-config** config map. After you apply the config map to your cluster, the configuration is loaded dynamically.

Example config map:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kubevirt-vm-labels-config
  namespace: openshift-cnv
data:
  allowlist: "*"
  ignorelist: ""
```

- **data.allowlist** specifies labels to expose.
  - If **data.allowlist** has a value of **"*"**, all labels are included.
  - If **data.allowlist** has a value of **""**, the metric does not return any labels.
  - If **data.allowlist** contains a list of label keys, only the explicitly named labels are exposed. For example: **allowlist: "example.io/name,example.io/version"**.
- **data.ignorelist** specifies labels to ignore. The ignore list overrides the allow list.
  - The **data.ignorelist** field does not support wildcard patterns. It can be empty or include a list of specific labels to ignore.
  - If **data.ignorelist** has a value of **""**, no labels are ignored.

12.2.3.6. Live migration metrics

You can query metrics to show live migration status.

kubevirt_vmi_migration_data_processed_bytes

The amount of guest operating system data that has migrated to the new virtual machine (VM). Type: Gauge.

kubevirt_vmi_migration_data_remaining_bytes

The amount of guest operating system data that remains to be migrated. Type: Gauge.

kubevirt_vmi_migration_memory_transfer_rate_bytes

The rate at which memory is becoming dirty in the guest operating system. Dirty memory is data that has been changed but not yet written to disk. Type: Gauge.

kubevirt_vmi_migrations_in_pending_phase

The number of pending migrations. Type: Gauge.

kubevirt_vmi_migrations_in_scheduling_phase

The number of scheduling migrations. Type: Gauge.

kubevirt_vmi_migrations_in_running_phase

The number of running migrations. Type: Gauge.

kubevirt_vmi_migration_succeeded

The number of successfully completed migrations. Type: Gauge.

kubevirt_vmi_migration_failed

The number of failed migrations. Type: Gauge.

12.2.4. Node Memory Overcommit dashboard

The Node Memory Overcommit dashboard displays physical and virtual memory utilization across the cluster, focusing on how virtual machines (VMs) affect node memory.

Use this dashboard to monitor memory capacity, detect memory pressure, identify overcommitment risks, and validate that system processes do not exceed their reserved memory.

You can access this dashboard from the web console in **Observe → Dashboards (Perses)**.

12.2.4.1. Dashboard filters

The dashboard provides two filter variables in the top toolbar:

Filter	Description	Default
Node	Filters all panels to display information about one or more nodes in the cluster.	**All**
role	Filters all panels to display information about nodes with a specific Kubernetes role, populated from **kube_node_role**.	**worker**

You can also change the time range displayed in the different panels in the top toolbar of the dashboard.

12.2.4.2. Summary section

The **Summary** section provides a high-level overview of cluster memory health. The panels in this section provide general memory utilization data for the cluster and the nodes. Most of the panels in this section appear in other sections of the dashboard as well. However, the following panels are unique to the **Summary** section:

Panel	Type	Description
Node Utilization - max	Gauge	Displays the highest node utilization in the cluster. Use this panel together with the **Node Utilization - min** panel to assess scheduling balance. A large gap between minimum and maximum node utilization indicates an imbalanced workload distribution.
Cluster Utilization	Time series	Displays the trend over time of several memory metrics. Use this panel to compare actual utilization and virtual memory plans against total capacity.

12.2.4.3. Cluster section

The **Cluster** section provides a more detailed view of cluster-wide memory behavior over a range of time. This section includes the following panels:

Panel	Type	Description
Physical Memory Utilization & Requests	Time series	Displays the total node memory capacity alongside actual memory utilization split into virtualization and non-virtualization workloads. This panel also shows the memory request plan (system-reserved requests and pod requests) so that you can compare actual usage to the scheduler expectations.
Cluster Utilization	Gauge	Displays the aggregated memory utilization of the nodes in the cluster. This panel also appears in the **Summary** section.

Panel	Type	Description
Virtual Memory Assignment	Time series	This panel displays the worst-case scenario if all virtual memory is used at present. It shows total node capacity, utilization without virtualization, and non-virtualization utilization combined with the total assigned VM memory. If the **VM assigned virtual memory** line nears or exceeds **Node capacity**, the cluster risks out-of-memory conditions under full VM memory pressure.
Cluster Virtual Committed	Gauge	Displays the percentage of committed virtual memory out of all of the allocatable physical memory. This panel also appears in the **Summary** section.
Cluster - Memory Pressure	Time series	Shows cluster-level memory PSI rates for **Waiting** (processes delayed by memory) and **Stalled** (processes completely blocked).
Cluster - Aggregated Swap	Time series	Shows total swap capacity and usage across all the nodes that you select. Rising swap usage indicates memory pressure that has not yet caused PSI stalls.

12.2.4.4. Nodes section

The **Nodes** section breaks down memory information per node to help identify imbalances. This section includes the following panels:

Panel	Type	Description
Utilization - Actual Overcommit Level	Time series	Displays the per-node allocatable memory utilization minus the system-reserved memory.
Node Utilization - min	Gauge	Displays the lowest node utilization in the cluster. This panel also appears in the **Summary** section.

Panel	Type	Description
Plan - Pod Requests per Node	Time series	Shows the memory request fill level per node: the sum of all pod memory requests on a node divided by the node's allocatable memory.
Node Requests - min/max	Stat	Displays the minimum and maximum pod request ratios per node. Use this panel to quickly assess the cluster's remaining capacity to host new workloads.
Plan - Virtual Memory Commit Level	Time series	Displays the virtual memory commit ratio per node. The panel compares total active and assigned VM memory against the node's available memory.
Node Virtual - min/max	Stat	Displays the minimum and maximum virtual commit levels per node.
Node - Pressure	Time series	Displays the memory PSI waiting rate per node. Each line represents one node to help you identify which nodes experience memory contention.
Node PSI - max	Gauge	Displays the highest PSI value across all nodes. PSI shows the amount of time applications are stalled or delayed waiting for memory resources. This panel also appears in the **Summary** section.

12.2.4.5. System Reserved section

The **System Reserved** section monitors whether the system (hypervisor) processes stay within their reserved memory budget. This section includes the following panels:

Panel	Type	Description
Utilization - Reserved System Memory	Time series	Displays the top 5 nodes by system-reserved memory utilization. This metric compares active system process memory against the reserved memory budget.

Panel	Type	Description
Utilization - min/max	Time series	Displays the minimum and maximum system-reserved memory utilization across all nodes over time.
System Exceeding Reservation	Stat	Displays the percentage or number of nodes out of all monitored nodes that are currently triggering the **SystemMemoryExceedsReservation** alert. This panel also appears in the **Summary** section.

12.2.4.6. Workloads section

The **Workloads** section, which is collapsed by default, focuses on individual VM memory behavior.

Panel	Type	Description
VM Overcommit Ratio	Time series	Displays the ratio of assigned virtual memory to pod memory requests for each VM.
VM Virtual Committed	Gauge	Displays the average VM overcommit ratio, which includes launcher overhead. This panel also appears in the **Summary** section.
VM Virtual Memory Utilization vs Host VM Utilization	Time series	Displays the 10 VMs with the highest ratio between guest-reported memory usage and host-side container memory usage. Use this panel to identify VMs where guest-reported usage differs significantly from host-side accounting. This difference indicates balloon driver effectiveness or memory accounting discrepancies.
Number of Running VMs	Time series	Displays the total number of running virtual machine instances (VMIs) on the cluster to provide context for the other workload panels.

12.2.4.7. Interpreting the dashboard

Monitor the dashboard indicators in the **Summary** section to identify early warning signs of memory pressure and prevent critical out-of-memory events.

Healthy state

The **Cluster Utilization** panel gauge is green (below 70%), the **Cluster Virtual Committed** panel gauge is below 120%, the **Node PSI - max** values are near zero, and no nodes exceed their system reservation.

Warning signs

Utilization gauges turn amber (80% to 90%), the virtual commit approaches 150%, or individual nodes diverge significantly from the cluster average, which suggests imbalanced scheduling.

Critical state

Utilization gauges turn red (above 90%), PSI values exceed 0.5, system reservation is exceeded on any node, or virtual commit ratios per node exceed 200%. These conditions indicate that the cluster is at risk of out-of-memory events and VM eviction.

12.2.5. Additional resources

- KubeVirt components metrics
- About OpenShift Dedicated monitoring
- Querying Prometheus
- Prometheus query examples

12.3. EXPOSING CUSTOM METRICS FOR VIRTUAL MACHINES

Monitor core platform components by using the OpenShift Dedicated monitoring stack based on the Prometheus monitoring system. Additionally, enable monitoring for user-defined projects by using the CLI and query custom metrics exposed for virtual machines through the **node-exporter** service.

12.3.1. Configure the node exporter service

The node-exporter agent is deployed on every virtual machine in the cluster from which you want to collect metrics. Configure the node-exporter agent as a service to expose internal metrics and processes that are associated with virtual machines.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in to the cluster as a user with **cluster-admin** privileges.
- Create the **cluster-monitoring-config ConfigMap** object in the **openshift-monitoring** project.
- Configure the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project by setting **enableUserWorkload** to **true**.

Procedure

1. Create the **Service** YAML file. In the following example, the file is called **node-exporter-service.yaml**.

```
kind: Service
apiVersion: v1
metadata:
  name: node-exporter-service
  namespace: dynamation
  labels:
    servicetype: metrics
spec:
  ports:
    - name: exmet
      protocol: TCP
      port: 9100
      targetPort: 9100
  type: ClusterIP
  selector:
    monitor: metrics
```

- **metadata.name** defines the node-exporter service that exposes the metrics from the virtual machines.
- **metadata.namespace** defines the namespace where the service is created.
- **metadata.labels.servicetype** defines the label for the service. The **ServiceMonitor** uses this label to match this service.
- **spec.ports.name** defines the name given to the port that exposes metrics on port 9100 for the **ClusterIP** service.
- **spec.ports.port** defines the target port used by **node-exporter-service** to listen for requests.
- **spec.ports.targetPort** defines the TCP port number of the virtual machine that is configured with the **monitor** label.
- **spec.selector.monitor** defines the label used to match the virtual machine's pods. In this example, any virtual machine's pod with the label **monitor** and a value of **metrics** will be matched.

2. Create the node-exporter service:

```
$ oc create -f node-exporter-service.yaml
```

12.3.2. Configure a virtual machine with the node exporter service

Download the **node-exporter** file on to the virtual machine. Then, create a **systemd** service that runs the node-exporter service when the virtual machine boots.

Prerequisites

- The pods for the component are running in the **openshift-user-workload-monitoring** project.
- Grant the **monitoring-edit** role to users who need to monitor this user-defined project.

Procedure

1. Log on to the virtual machine.
2. Download the **node-exporter** file on to the virtual machine by using the directory path that applies to the version of **node-exporter** file.

```
$ wget
https://github.com/prometheus/node_exporter/releases/download/<version>/node_exporter-
<version>.linux-<architecture>.tar.gz
```

3. Extract the executable and place it in the **/usr/bin** directory.

```
$ sudo tar xvf node_exporter-<version>.linux-<architecture>.tar.gz \
    --directory /usr/bin --strip 1 "*/node_exporter"
```

4. Create a **node_exporter.service** file in this directory path: **/etc/systemd/system**. This **systemd** service file runs the node-exporter service when the virtual machine reboots.

```
[Unit]
Description=Prometheus Metrics Exporter
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=always
RestartSec=1
User=root
ExecStart=/usr/bin/node_exporter

[Install]
WantedBy=multi-user.target
```

5. Enable and start the **systemd** service.

```
$ sudo systemctl enable node_exporter.service
```

```
$ sudo systemctl start node_exporter.service
```

Verification

- Verify that the node-exporter agent is reporting metrics from the virtual machine.

```
$ curl http://localhost:9100/metrics
```

Example output:

```
go_gc_duration_seconds{quantile="0"} 1.5244e-05
go_gc_duration_seconds{quantile="0.25"} 3.0449e-05
go_gc_duration_seconds{quantile="0.5"} 3.7913e-05
```

12.3.3. Create a custom monitoring label for virtual machines

To enable queries to multiple virtual machines from a single service, you can add a custom label in the virtual machine's YAML file.

Prerequisites

- Install the OpenShift CLI (**oc**).
- Log in as a user with **cluster-admin** privileges.
- Access to the web console for stop and restart a virtual machine.

Procedure

1. Edit the **template** spec of your virtual machine configuration file. In this example, the label **monitor** has the value **metrics**.

```
spec:
  template:
    metadata:
      labels:
        monitor: metrics
```

2. Stop and restart the virtual machine to create a new pod with the label name given to the **monitor** label.

12.3.3.1. Query the node-exporter service for metrics

Metrics are exposed for virtual machines through an HTTP service endpoint under the /**metrics** canonical name. When you query for metrics, Prometheus directly scrapes the metrics from the metrics endpoint exposed by the virtual machines and presents these metrics for viewing.

Prerequisites

- You have access to the cluster as a user with **cluster-admin** privileges or the **monitoring-edit** role.
- You have enabled monitoring for the user-defined project by configuring the node-exporter service.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Obtain the HTTP service endpoint by specifying the namespace for the service:

```
$ oc get service -n <namespace> <node-exporter-service>
```

2. To list all available metrics for the node-exporter service, query the **metrics** resource.

```
$ curl http://<172.30.226.162:9100>/metrics | grep -vE "^#|^$"
```

Example output:

```
node_arp_entries{device="eth0"} 1
node_boot_time_seconds 1.643153218e+09
node_context_switches_total 4.4938158e+07
node_cooling_device_cur_state{name="0",type="Processor"} 0
node_cooling_device_max_state{name="0",type="Processor"} 0
node_cpu_guest_seconds_total{cpu="0",mode="nice"} 0
node_cpu_guest_seconds_total{cpu="0",mode="user"} 0
node_cpu_seconds_total{cpu="0",mode="idle"} 1.10586485e+06
node_cpu_seconds_total{cpu="0",mode="iowait"} 37.61
node_cpu_seconds_total{cpu="0",mode="irq"} 233.91
node_cpu_seconds_total{cpu="0",mode="nice"} 551.47
node_cpu_seconds_total{cpu="0",mode="softirq"} 87.3
node_cpu_seconds_total{cpu="0",mode="steal"} 86.12
node_cpu_seconds_total{cpu="0",mode="system"} 464.15
node_cpu_seconds_total{cpu="0",mode="user"} 1075.2
node_disk_discard_time_seconds_total{device="vda"} 0
node_disk_discard_time_seconds_total{device="vdb"} 0
node_disk_discarded_sectors_total{device="vda"} 0
node_disk_discarded_sectors_total{device="vdb"} 0
node_disk_discards_completed_total{device="vda"} 0
node_disk_discards_completed_total{device="vdb"} 0
node_disk_discards_merged_total{device="vda"} 0
node_disk_discards_merged_total{device="vdb"} 0
node_disk_info{device="vda",major="252",minor="0"} 1
node_disk_info{device="vdb",major="252",minor="16"} 1
node_disk_io_now{device="vda"} 0
node_disk_io_now{device="vdb"} 0
node_disk_io_time_seconds_total{device="vda"} 174
node_disk_io_time_seconds_total{device="vdb"} 0.054
node_disk_io_time_weighted_seconds_total{device="vda"} 259.79200000000003
node_disk_io_time_weighted_seconds_total{device="vdb"} 0.039
node_disk_read_bytes_total{device="vda"} 3.71867136e+08
node_disk_read_bytes_total{device="vdb"} 366592
node_disk_read_time_seconds_total{device="vda"} 19.128
node_disk_read_time_seconds_total{device="vdb"} 0.039
node_disk_reads_completed_total{device="vda"} 5619
node_disk_reads_completed_total{device="vdb"} 96
node_disk_reads_merged_total{device="vda"} 5
node_disk_reads_merged_total{device="vdb"} 0
node_disk_write_time_seconds_total{device="vda"} 240.66400000000002
node_disk_write_time_seconds_total{device="vdb"} 0
node_disk_writes_completed_total{device="vda"} 71584
node_disk_writes_completed_total{device="vdb"} 0
node_disk_writes_merged_total{device="vda"} 19761
node_disk_writes_merged_total{device="vdb"} 0
node_disk_written_bytes_total{device="vda"} 2.007924224e+09
node_disk_written_bytes_total{device="vdb"} 0
```

12.3.4. Create a ServiceMonitor resource for the node exporter service

You can use a Prometheus client library and scrape metrics from the /**metrics** endpoint to access and view the metrics exposed by the node-exporter service. Use a **ServiceMonitor** custom resource definition (CRD) to monitor the node exporter service.

Prerequisites

- You have access to the cluster as a user with **cluster-admin** privileges or the **monitoring-edit** role.
- You have enabled monitoring for the user-defined project by configuring the node-exporter service.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a YAML file for the **ServiceMonitor** resource configuration. In this example, the service monitor matches any service with the label **metrics** and queries the **exmet** port every 30 seconds.

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    k8s-app: node-exporter-metrics-monitor
  name: node-exporter-metrics-monitor
  namespace: dynamation
spec:
  endpoints:
  - interval: 30s
    port: exmet
    scheme: http
  selector:
    matchLabels:
      servicetype: metrics
```

   - **metadata.name** defines the name of the **ServiceMonitor**.
   - **metadata.namespace** defines the namespace where the **ServiceMonitor** is created.
   - **spec.endpoints.interval** defines the interval at which the port will be queried.
   - **spec.endpoints.port** defines the name of the port that is queried every 30 seconds

2. Create the **ServiceMonitor** configuration for the node-exporter service.

```
$ oc create -f node-exporter-metrics-monitor.yaml
```

12.3.4.1. Access the node exporter service outside the cluster

You can access the node-exporter service outside the cluster and view the exposed metrics.

Prerequisites

- You have access to the cluster as a user with **cluster-admin** privileges or the **monitoring-edit** role.
- You have enabled monitoring for the user-defined project by configuring the node-exporter service.

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

Procedure

1. Expose the node-exporter service.

```
$ oc expose service -n <namespace> <node_exporter_service_name>
```

2. Obtain the FQDN (Fully Qualified Domain Name) for the route.

```
$ oc get route -o=custom-columns=NAME:.metadata.name,DNS:.spec.host
```

Example output:

```
NAME                    DNS
node-exporter-service   node-exporter-service-dynamation.apps.cluster.example.org
```

3. Use the **curl** command to display metrics for the node-exporter service.

```
$ curl -s http://node-exporter-service-dynamation.apps.cluster.example.org/metrics
```

Example output:

```
go_gc_duration_seconds{quantile="0"} 1.5382e-05
go_gc_duration_seconds{quantile="0.25"} 3.1163e-05
go_gc_duration_seconds{quantile="0.5"} 3.8546e-05
go_gc_duration_seconds{quantile="0.75"} 4.9139e-05
go_gc_duration_seconds{quantile="1"} 0.000189423
```

12.3.5. Additional resources

- Creating and using config maps
- Controlling virtual machine states

12.4. VIRTUAL MACHINE HEALTH CHECKS

Define probes and watchdogs in the **VirtualMachine** resource to configure virtual machine (VM) health checks. Health checks monitor and report the internal state of a VM.

You can configure VM health checks by defining readiness and liveness probes in the **VirtualMachine** resource.

12.4.1. Readiness and liveness probes for VM health

Use readiness and liveness probes to detect and handle unhealthy virtual machines (VMs). You can include one or more probes in the specification of the VM to ensure that traffic does not reach a VM that is not ready for it and that a new VM is created when a VM becomes unresponsive.

A *readiness probe* determines whether a VM is ready to accept service requests. If the probe fails, the VM is removed from the list of available endpoints until the VM is ready.

A *liveness probe* determines whether a VM is responsive. If the probe fails, the VM is deleted and a new VM is created to restore responsiveness.

You can configure readiness and liveness probes by setting the **spec.readinessProbe** and the **spec.livenessProbe** fields of the **VirtualMachine** object. These fields support the following tests:

HTTP GET

The probe determines the health of the VM by using a web hook. The test is successful if the HTTP response code is between 200 and 399. You can use an HTTP GET test with applications that return HTTP status codes when they are completely initialized.

TCP socket

The probe attempts to open a socket to the VM. The VM is only considered healthy if the probe can establish a connection. You can use a TCP socket test with applications that do not start listening until initialization is complete.

Guest agent ping

The probe uses the **guest-ping** command to determine if the QEMU guest agent is running on the virtual machine.

12.4.1.1. Define an HTTP readiness probe

You can define an HTTP readiness probe by setting the **spec.readinessProbe.httpGet** field of the virtual machine (VM) configuration.

Prerequisites

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

Procedure

1. Include details of the readiness probe in the VM configuration file.
   Sample readiness probe with an HTTP GET test:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      readinessProbe:
        httpGet:
          port: 1500
          path: /healthz
          httpHeaders:
          - name: Custom-Header
            value: Awesome
        initialDelaySeconds: 120
        periodSeconds: 20
        timeoutSeconds: 10
```

```
      failureThreshold: 3
      successThreshold: 3
# ...
```

- **spec.template.spec.readinessProbe.httpGet** defines the HTTP GET request to perform to connect to the VM.
- **spec.template.spec.readinessProbe.httpGet.port** defines the port of the VM that the probe queries. In the above example, the probe queries port 1500.
- **spec.template.spec.readinessProbe.httpGet.path** defines the path to access on the HTTP server. In the above example, if the handler for the server's /healthz path returns a success code, the VM is considered to be healthy. If the handler returns a failure code, the VM is removed from the list of available endpoints.
- **spec.template.spec.readinessProbe.initialDelaySeconds** defines the time, in seconds, after the VM starts before the readiness probe is initiated.
- **spec.template.spec.readinessProbe.periodSeconds** defines the delay, in seconds, between performing probes. The default delay is 10 seconds. This value must be greater than **timeoutSeconds**.
- **spec.template.spec.readinessProbe.timeoutSeconds** defines the number of seconds of inactivity after which the probe times out and the VM is assumed to have failed. The default value is 1. This value must be lower than **periodSeconds**.
- **spec.template.spec.readinessProbe.failureThreshold** defines the number of times that the probe is allowed to fail. The default is 3. After the specified number of attempts, the pod is marked **Unready**.
- **spec.template.spec.readinessProbe.successThreshold** defines the number of times that the probe must report success, after a failure, to be considered successful. The default is 1.

2. Create the VM by running the following command:

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

12.4.1.2. Define a TCP readiness probe

You can define a TCP readiness probe by setting the **spec.readinessProbe.tcpSocket** field of the virtual machine (VM) configuration.

Prerequisites

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

Procedure

1. Include details of the TCP readiness probe in the VM configuration file.
   Sample readiness probe with a TCP socket test:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
```

```
metadata:
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      readinessProbe:
        initialDelaySeconds: 120
        periodSeconds: 20
        tcpSocket:
          port: 1500
        timeoutSeconds: 10
# ...
```

- **spec.template.spec.readinessProbe.initialDelaySeconds** defines the time, in seconds, after the VM starts before the readiness probe is initiated.
- **spec.template.spec.readinessProbe.periodSeconds`defines the delay, in seconds, between performing probes. The default delay is 10 seconds. This value must be greater than `timeoutSeconds**.
- **spec.template.spec.readinessProbe.tcpSocket** defines the TCP action to perform.
- **spec.template.spec.readinessProbe.tcpSocket.port** defines the port of the VM that the probe queries.
- **spec.template.spec.readinessProbe.timeoutSeconds** defines the number of seconds of inactivity after which the probe times out and the VM is assumed to have failed. The default value is 1. This value must be lower than **periodSeconds**.

2. Create the VM by running the following command:

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

12.4.1.3. Define an HTTP liveness probe

Define an HTTP liveness probe by setting the **spec.livenessProbe.httpGet** field of the virtual machine (VM) configuration. You can define both HTTP and TCP tests for liveness probes in the same way as readiness probes. This procedure configures a sample liveness probe with an HTTP GET test.

Prerequisites

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

Procedure

1. Include details of the HTTP liveness probe in the VM configuration file.
Sample liveness probe with an HTTP GET test:

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

```
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      livenessProbe:
        initialDelaySeconds: 120
        periodSeconds: 20
        httpGet:
          port: 1500
          path: /healthz
          httpHeaders:
          - name: Custom-Header
            value: Awesome
        timeoutSeconds: 10
# ...
```

- **spec.tenmplate.spec.livenessProbe.initialDelaySeconds** defines the time, in seconds, after the VM starts before the liveness probe is initiated.
- **spec.tenmplate.spec.livenessProbe.periodSeconds** defines the delay, in seconds, between performing probes. The default delay is 10 seconds. This value must be greater than **timeoutSeconds**.
- **spec.tenmplate.spec.livenessProbe.httpGet** defines the HTTP GET request to perform to connect to the VM.
- **spec.tenmplate.spec.livenessProbe.httpGet.port** defines the port of the VM that the probe queries. In the above example, the probe queries port 1500. The VM installs and runs a minimal HTTP server on port 1500 via cloud-init.
- **spec.tenmplate.spec.livenessProbe.httpGet.path** defines the path to access on the HTTP server. In the above example, if the handler for the server's **/healthz** path returns a success code, the VM is considered to be healthy. If the handler returns a failure code, the VM is deleted and a new VM is created.
- **spec.tenmplate.spec.livenessProbe.timeoutSeconds** defines the number of seconds of inactivity after which the probe times out and the VM is assumed to have failed. The default value is 1. This value must be lower than **periodSeconds**.

2. Create the VM by running the following command:

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

12.4.2. Watchdog devices and recovery actions

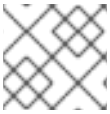
Watchdog devices monitor guest operating system responsiveness and trigger recovery actions when a virtual machine becomes unresponsive.

A watchdog device continuously monitors the agent running on a virtual machine (VM). When the guest operating system becomes unresponsive, the watchdog can trigger one of three recovery actions depending on how it is configured.

The **poweroff** action causes the VM to power down immediately. If **spec.runStrategy** is not set to **manual**, the VM automatically reboots after powering down.

The **reset** action reboots the VM in place without allowing the guest operating system to react. When using the reset action, be aware that the reboot time might cause liveness probes to time out. If cluster-level protections detect a failed liveness probe, the VM might be forcibly rescheduled, which increases the overall reboot time.

The **shutdown** action initiates a graceful shutdown by stopping all services before powering down the VM.



NOTE

Watchdog functionality is not available for Windows VMs.

To implement watchdog functionality, you must configure the watchdog device for the VM and install the watchdog agent on the guest operating system.

12.4.2.1. Configure a watchdog device for a virtual machine

You configure a watchdog device for the virtual machine (VM).

Prerequisites

- For **x86** systems, the VM must use a kernel that works with the **i6300esb** watchdog device. If you use **s390x** architecture, the kernel must be enabled for **diag288**. Red Hat Enterprise Linux (RHEL) images support **i6300esb** and **diag288**.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Create a **YAML** file with the following contents:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: <vm-label>
  name: <vm-name>
spec:
  runStrategy: Halted
  template:
    metadata:
      labels:
        kubevirt.io/vm: <vm-label>
    spec:
      domain:
        devices:
          watchdog:
            name: <watchdog>
            <watchdog-device-model>:
              action: "poweroff"
# ...
```

- **spec.template.spec.domain.devices.watchdog.name.<watchdog-device-model>** defines the watchdog device model to use. For **x86** specify **i6300esb**. For **s390x** specify **diag288**.

- **spec.template.spec.domain.devices.watchdog.name.<watchdog-device-model>.action** defines the watchdog device action. Specify **poweroff**, **reset**, or **shutdown**. The **shutdown** action requires that the guest virtual machine is responsive to ACPI signals. Using **shutdown** is not recommended.
  The example above configures the watchdog device on a VM with the **poweroff** action and exposes the device as /**dev**/**watchdog**.

  This device can now be used by the watchdog binary.

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

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

Verification

1. Run the following command to verify that the VM is connected to the watchdog device:



IMPORTANT

Verification steps are provided for testing watchdog functionality only and must not be run on production machines.

```
$ lspci | grep watchdog -i
```

2. Run one of the following commands to confirm the watchdog is active:

   - Trigger a kernel panic:

```
# echo c > /proc/sysrq-trigger
```

   - Stop the watchdog service:

```
# pkill -9 watchdog
```

12.4.2.2. Install the watchdog agent on the guest

You can install the watchdog agent on the guest and start the **watchdog** service.

Procedure

1. Log in to the virtual machine as root user.

2. Verify that the /**dev**/**watchdog** file path is present in the VM by running the following command:

```
# ls /dev/watchdog
```

3. Install the **watchdog** package and its dependencies:

```
# yum install watchdog
```

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

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

5. Enable the **watchdog** service to start on boot:

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

12.5. OPENSHIFT VIRTUALIZATION RUNBOOKS

To diagnose and resolve OpenShift Virtualization alerts, you can use the OpenShift Virtualization Operator runbooks. These guides help ensure you can effectively troubleshoot cluster issues and restore system health.



NOTE

Runbooks for the OpenShift Virtualization Operator are maintained in the openshift/runbooks Git repository, and you can view them on GitHub.

Table 12.1. Runbooks for OpenShift Virtualization alerts

Alert	GitHub link
CDIDataImportCronOutdated	Runbook
CDIDataVolumeUnusualRestartCount	Runbook
CDIDefaultStorageClassDegraded	Runbook
CDIMultipleDefaultVirtStorageClasses	Runbook
CDINoDefaultStorageClass	Runbook
CDINotReady	Runbook
CDIOperatorDown	Runbook
CDIStorageProfilesIncomplete	Runbook
ClusterVMPanicDetected	Runbook
CnaoDown	Runbook
CnaoNMstateMigration	Runbook

Alert	GitHub link
DeprecatedMachineType	Runbook
GuestFilesystemAlmostOutOfSpace	Runbook
GuestVCPUQueueHighCritical	Runbook
GuestVCPUQueueHighWarning	Runbook
HAControlPlaneDown	Runbook
HCOGoldenImageWithNoArchitectureAnnotation	Runbook
HCOGoldenImageWithNoSupportedArchitecture	Runbook
HCOInstallationIncomplete	Runbook
HCOMisconfiguredDescheduler	Runbook
HCOMultiArchGoldenImagesDisabled	Runbook
HCOOperatorConditionsUnhealthy	Runbook
HighNodeCPUFrequency	Runbook
HPPNotReady	Runbook
HPPOperatorDown	Runbook
HPPSharingPoolPathWithOS	Runbook
HighCPUWorkload	Runbook
KubemacpoolDown	Runbook
KubeVirtCRModified	Runbook
KubeVirtDeprecatedAPIRequested	Runbook
KubeVirtVMGuestMemoryAvailableLow	Runbook
KubeVirtVMGuestMemoryPressure	Runbook

Alert	GitHub link
KubeVirtNoAvailableNodesToRunVMs	Runbook
KubeVirtVMIExcessiveMigrations	Runbook
LowKVMNodesCount	Runbook
LowReadyVirtControllersCount	Runbook
LowReadyVirtOperatorsCount	Runbook
LowVirtAPICount	Runbook
LowVirtControllersCount	Runbook
LowVirtOperatorCount	Runbook
NetworkAddonsConfigNotReady	Runbook
NoLeadingVirtOperator	Runbook
NoReadyVirtController	Runbook
NoReadyVirtOperator	Runbook
NodeNetworkInterfaceDown	Runbook
OrphanedVirtualMachineInstances	Runbook
OutdatedVirtualMachineInstanceWorkloads	Runbook
PersistentVolumeFillingUp	Runbook
SSPCommonTemplatesModificationReverted	Runbook
SSPDown	Runbook
SSPFailingToReconcile	Runbook
SSPHighRateRejectedVms	Runbook
SSPTemplateValidatorDown	Runbook
UnsupportedHCOModification	Runbook

Alert	GitHub link
VirtAPIDown	Runbook
VirtApiRESTErrorsBurst	Runbook
VirtControllerDown	Runbook
VirtControllerRESTErrorsBurst	Runbook
VirtHandlerDaemonSetRolloutFailing	Runbook
VirtHandlerRESTErrorsBurst	Runbook
VirtLauncherPodsStuckFailed	Runbook
VirtOperatorDown	Runbook
VirtOperatorRESTErrorsBurst	Runbook
VirtualMachineInstanceHasEphemeralHotplugVolume	Runbook
VirtualMachineStuckInUnhealthyState	Runbook
VirtualMachineStuckOnNode	Runbook
VMCannotBeEvicted	Runbook
VMNonRecoverableOSPanic	Runbook
VMStorageClassWarning	Runbook

12.5.1. Runbooks for deprecated alerts

Deprecated alerts no longer report actual issues and you can safely ignore them.

Table 12.2. Runbooks for deprecated OpenShift Virtualization alerts

Alert	GitHub link	Notes
DuplicateWaspAgentDSDetected	Runbook	-
KubeMacPoolDuplicateMacsFound	Runbook	-
KubeVirtComponentExceedsRequestedCPU	Runbook	-

Alert	GitHub link	Notes
KubeVirtComponentExceedsRequestedMemory	Runbook	-
KubevirtVmHighMemoryUsage	Runbook	-
OperatorConditionsUnhealthy	Runbook	-
SingleStackIPv6Unsupported	Runbook	-
SSPOperatorDown	Runbook	-
VirtApiRESTErrorsHigh	Runbook	-
VirtControllerRESTErrorsHigh	Runbook	-
VirtHandlerRESTErrorsHigh	Runbook	-
VirtOperatorRESTErrorsHigh	Runbook	-
VirtualMachineCRCErrors	Runbook	Renamed to **VMStorageClassWarning**.

CHAPTER 13. SUPPORT

13.1. SUPPORT OVERVIEW

Accelerate the resolution of cluster and virtual machine (VM) issues by using the integrated diagnostic tools and support provided by OpenShift Virtualization.

To gather debugging information, configure Prometheus and Alertmanager and collect **must-gather** data for OpenShift Dedicated and OpenShift Virtualization.

13.1.1. Opening a support case

Open a support case with Red Hat Support when you encounter an issue that requires immediate assistance.

13.1.1.1. Collecting data for Red Hat Support

Gather information about the issue affecting your environment to submit with your support case. This aids Red Hat Support in effectively diagnosing your issue.

Gather troubleshooting information by using the following tools:

- Configure Prometheus and Alertmanager.

13.1.1.2. Submitting a support case

Submit a support case to resolve a cluster issue that is affecting the ability of OpenShift Virtualization to function properly in your environment.

You can submit a support case to Red Hat Support by using the Customer Support page. Include data that you collected about your issue with your support request.

13.1.2. Report a bug in OpenShift Virtualization

To report a bug, use the Red Hat Issue Router (RHIR), which is available in the Customer Portal Labs.

Procedure

1. Access the RHIR.
2. In the list of all OpenShift Virtualization components, find the component for which you want to report an issue.
3. Click the **Report a bug** link of the component.
4. On the **Create issue** page, fill out the form:
    a. Complete the **Summary** and **Description** fields. In the **Description** field, include a detailed description of the issue.
    b. Submit any collected troubleshooting information:
        i. Add any textual troubleshooting information, such as command outputs, in the **Description** field.

ii. Add troubleshooting files using the **Attachment** field.

5. Click **Create** at the bottom of the page.
6. Review the details of the bug you created.

13.1.3. Report a OpenShift Virtualization documentation issue

To report an error or request an enhancement in the documentation, log in to your Red Hat Jira account and submit an issue. If you do not have a Red Hat Jira account, you are prompted to create an account.

Procedure

1. Create a Jira issue and in the **Component** field, select **CNV Documentation**.
2. Enter a brief description of the issue in the **Summary**.
3. Provide a detailed description of the issue or requested enhancement in the **Description**. Include a URL to where the issue occurs in the documentation.
4. Click **Create**.

13.1.4. Web console monitoring

Monitor cluster and virtual machine (VM) health with the OpenShift Dedicated web console.

The OpenShift Dedicated web console displays resource usage, alerts, events, and trends for your cluster and for OpenShift Virtualization components and resources.

Table 13.1. Web console pages for monitoring and troubleshooting

Page	Description
Virtualization → VirtualMachines → Overview page	Cluster details, status, alerts, inventory, and resource usage
Virtualization → VirtualMachines → Overview → Overview tab	OpenShift Virtualization resources, usage, alerts, and status
Virtualization → Migrations page	Progress of live migrations
Virtualization → VirtualMachines → Virtual machines tab	CPU, memory, and storage usage summary
Virtualization → VirtualMachines → Virtual machines → VirtualMachine details → Metrics tab	VM resource usage, storage, network, and migration
Virtualization → VirtualMachines → Virtual machines → VirtualMachine details → Events tab	List of VM events

Page	Description
Virtualization → VirtualMachines → Virtual machines → VirtualMachine details → Diagnostics tab	VM status conditions and volume snapshot status

13.1.5. Additional resources

- Submitting a support case

13.2. COLLECTING DATA FOR RED HAT SUPPORT

When you submit a support case to Red Hat Support, it is helpful to provide debugging information for OpenShift Dedicated and OpenShift Virtualization.

You can use the following tools to collect debugging information:

Prometheus

Prometheus is a time-series database and a rule evaluation engine for metrics. Prometheus sends alerts to Alertmanager for processing.

Alertmanager

The Alertmanager service handles alerts received from Prometheus. The Alertmanager is also responsible for sending the alerts to external notification systems.

13.2.1. Collecting data about your environment

Collecting data about your environment minimizes the time required to analyze and determine the root cause.

Prerequisites

- You have set the retention time for Prometheus metrics data to a minimum of seven days.
- You have configured the Alertmanager to capture relevant alerts and to send alert notifications to a dedicated mailbox so that they can be viewed and persisted outside the cluster.
- You have recorded the exact number of affected nodes and virtual machines.

Procedure

1. Collect Prometheus metrics for the cluster.

13.2.2. Collecting data about virtual machines

Collecting data about malfunctioning virtual machines (VMs) minimizes the time required to analyze and determine the root cause.

Prerequisites

- For Linux VMs, you have installed the latest QEMU guest agent.

- For Windows VMs, you have:
  - Recorded the Windows patch update details.
  - Installed the latest VirtIO drivers.
  - Installed the latest QEMU guest agent.
  - If Remote Desktop Protocol (RDP) is enabled, you have connected by using the desktop viewer to determine whether there is a problem with the connection software.

Procedure

1. Collect screenshots of VMs that have crashed before you restart them.
2. Collect memory dumps from VMs before remediation attempts.
3. Record factors that the malfunctioning VMs have in common. For example, the VMs have the same host or network.

13.2.3. Generate a VM memory dump

When a virtual machine (VM) terminates unexpectedly, you can run the **virtctl memory-dump** command to output a VM memory dump and save it on a persistent volume claim (PVC). Afterward, you can analyze the memory dump to diagnose and troubleshoot issues on the VM.

Procedure

1. Optional: You have an existing PVC on which you want to save the memory dump.
   - The PVC volume mode must be **FileSystem**.
   - The PVC must be large enough to contain the memory dump.
     The formula for calculating the PVC size is **(VMMemorySize + 100Mi) * (1 + FileSystemOverhead)**, where **100Mi** is the memory dump overhead, and **FileSystemOverhead** is defined in the **HCO** object.
2. Create a memory dump of the required VM:
   - If you have an existing PVC selected on which you want to save the memory dump:

     ```
     $ virtctl memory-dump get <vm_name> --claim-name=<pvc_name>
     ```

   - If you want to create a new PVC for the memory dump:

     ```
     $ virtctl memory-dump get <vm_name> --claim-name=<new_pvc_name> --create-claim
     ```

3. Download the memory dump:

   ```
   $ virtctl memory-dump download <vm_name> --output=<output_file>
   ```

4. Attach the memory dump to a Red Hat Support case.
   Alternatively, you can inspect the memory dump, for example by using the volatility3 tool.
5. Optional: Remove the memory dump:

```
$ virtctl memory-dump remove <vm_name>
```

13.2.4. Additional resources

- VM support overview
- How to provide log files to Red Hat Support (Red Hat Knowledgebase)
- About OpenShift Dedicated monitoring
- Install the QEMU guest agent on a Linux VM
- Install VirtIO drivers from a SATA CD on an existing VM
- Connect to the desktop viewer by using the web console
- Generate a VM memory dump
- Submitting a support case
- Modifying retention time and size for Prometheus metrics data
- Configuring the Alertmanager to capture relevant alerts and to send alert notifications to a dedicated mailbox
- Modifying retention time and size for Prometheus metrics data
- Configuring alerts and notifications
- Downloading log files and diagnostic information
- Querying metrics for all projects with the monitoring dashboard
- Installing the latest VirtIO drivers
- Volatility3 tool
- Customer Support
- Create issue

13.3. TROUBLESHOOTING

To diagnose and resolve issues with virtual machine (VM) and cluster components, you can troubleshoot OpenShift Virtualization by using the web console or the OpenShift CLI (**oc**). These practices help ensure your virtualized infrastructure remains healthy.

13.3.1. View VM and resource events

To monitor and troubleshoot virtual machine (VM), namespace, and resource issues, you can review OpenShift Dedicated events. Tracking this life-cycle information helps ensure you maintain a healthy cluster environment.

Procedure

- To view VM events, go to **VirtualMachine details → Events** in the web console.
- To view namespace events, run the following command:

```
$ oc get events -n <namespace>
```

- To view resource events, run the following command:

```
$ oc describe <resource> <resource_name>
```

13.3.2. OpenShift Virtualization pod logs

To diagnose issues and monitor OpenShift Virtualization pods, you can view logs by using the web console or the CLI.

Pod logs exist only while the pod exists. If a container restarts within a pod, you can still retrieve the logs of the earlier container, but the cluster discards the logs when it replaces the pod itself. For evidence that must survive pod replacement, or that spans several virtual machines (VMs) or nodes, use aggregated logs instead.

13.3.2.1. Configure OpenShift Virtualization pod log verbosity

To gather more detailed diagnostic information for troubleshooting, you can configure the verbosity level of OpenShift Virtualization pod logs. Edit the **HyperConverged** custom resource (CR) to configure this setting.

Prerequisites

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

Procedure

1. To set log verbosity for specific components, open the **HyperConverged** CR in your default text editor by running the following command:

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

2. Set the log level for one or more components by editing the **spec.logVerbosityConfig** stanza. For example:

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  logVerbosityConfig:
    kubevirt:
      virtAPI: 5
      virtController: 4
      virtHandler: 3
      virtLauncher: 2
      virtOperator: 6
```

The log verbosity value must be an integer in the range **1–9**, where a higher number indicates a more detailed log. In this example, the **virtAPI** component logs are exposed if their priority level is **5** or higher.

3. Apply your changes by saving and exiting the editor.

13.3.2.2. View virt-launcher pod logs with the web console

To diagnose and troubleshoot virtual machine issues, you can view the **virt-launcher** pod logs by using the OpenShift Dedicated web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines**.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. On the **General** tile, click the pod name to open the **Pod details** page.
4. Click the **Logs** tab to view the logs.

13.3.2.3. View OpenShift Virtualization pod logs with the CLI

To diagnose issues and monitor OpenShift Virtualization pods, you can view logs by using the OpenShift CLI (**oc**).

Prerequisites

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

Procedure

1. View a list of pods in the OpenShift Virtualization namespace by running the following command:

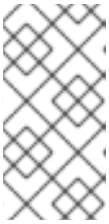
```
$ oc get pods -n openshift-cnv
```

Example output:

```
NAME                               READY   STATUS    RESTARTS   AGE
disks-images-provider-7gqbc        1/1     Running   0          32m
disks-images-provider-vg4kx        1/1     Running   0          32m
virt-api-57fcc4497b-7qfmc          1/1     Running   0          31m
virt-api-57fcc4497b-tx9nc          1/1     Running   0          31m
virt-controller-76c784655f-7fp6m   1/1     Running   0          30m
virt-controller-76c784655f-f4pbd   1/1     Running   0          30m
virt-handler-2m86x                 1/1     Running   0          30m
virt-handler-9qs6z                 1/1     Running   0          30m
virt-operator-7ccfdbf65f-q5snk     1/1     Running   0          32m
virt-operator-7ccfdbf65f-vllz8     1/1     Running   0          32m
```

2. View the pod log by running the following command:

```
$ oc logs -n openshift-cnv <pod_name>
```



NOTE

If a pod fails to start, you can use the **--previous** option to view logs from the last attempt.

To monitor log output in real time, use the **-f** option.

Example output:

{"component":"virt-handler","level":"info","msg":"set verbosity to 2","pos":"virt-handler.go:453","timestamp":"2022-04-17T08:58:37.373695Z"}
{"component":"virt-handler","level":"info","msg":"set verbosity to 2","pos":"virt-handler.go:453","timestamp":"2022-04-17T08:58:37.373726Z"}
{"component":"virt-handler","level":"info","msg":"setting rate limiter to 5 QPS and 10 Burst","pos":"virt-handler.go:462","timestamp":"2022-04-17T08:58:37.373782Z"}
{"component":"virt-handler","level":"info","msg":"CPU features of a minimum baseline CPU model: map[apic:true clflush:true cmov:true cx16:true cx8:true de:true fpu:true fxsr:true lahf_lm:true lm:true mca:true mce:true mmx:true msr:true mtrr:true nx:true pae:true pat:true pge:true pni:true pse:true pse36:true sep:true sse:true sse2:true sse4.1:true ssse3:true syscall:true tsc:true]","pos":"cpu_plugin.go:96","timestamp":"2022-04-17T08:58:37.390221Z"}
{"component":"virt-handler","level":"warning","msg":"host model mode is expected to contain only one model","pos":"cpu_plugin.go:103","timestamp":"2022-04-17T08:58:37.390263Z"}
{"component":"virt-handler","level":"info","msg":"node-labeller is running","pos":"node_labeller.go:94","timestamp":"2022-04-17T08:58:37.391011Z"}

13.3.3. Guest system logs

To diagnose issues with virtual machine (VM) guests, you can configure access to and view their boot logs using the OpenShift Dedicated web console or the OpenShift CLI (**oc**).

If the guest VM has no network, you can access it using its VNC or serial console.

This feature is disabled by default. If a VM does not explicitly have this setting enabled or disabled, it inherits the cluster-wide default setting.



IMPORTANT

If sensitive information such as credentials or other personally identifiable information (PII) is written to the serial console, it is logged with all other visible text. Use SSH to send sensitive data.

13.3.3.1. Enable default access to guest system logs with the web console

To troubleshoot issues more easily, you can enable default access to virtual machine (VM) guest system logs by using the web console.

Procedure

1. From the side menu, click **Virtualization → Settings**.
2. Click **Cluster → Guest management**.
3. Set **Enable guest system log access**to on.

4. Optional: If you want to hide the VM user credentials that were set by using cloud-init, set **Hide guest credentials for non-privileged users** to on.

13.3.3.2. Enable default access to guest system logs with the CLI

To troubleshoot issues more easily, you can enable default access to virtual machine (VM) guest system logs by editing the **HyperConverged** custom resource (CR).

Prerequisites

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

Procedure

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

```
$ oc edit hyperconvergeds.v1beta1.hco.kubevirt.io kubevirt-hyperconverged -n openshift-cnv
```

2. Update the **disableSerialConsoleLog** value. For example:

```
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  virtualMachineOptions:
    disableSerialConsoleLog: true
#...
```

   Set the value of **disableSerialConsoleLog** to **false** if you want serial console access to be enabled on VMs by default.

13.3.3.3. Set per-VM guest system log access with the web console

To troubleshoot a specific virtual machine (VM) without changing global settings, you can configure the guest system log access by using the web console.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Configuration** tab.
4. Set **Guest system log access** to on or off.

13.3.3.4. Set per-VM guest system log access with the CLI

To troubleshoot a specific virtual machine (VM) without changing global settings, you can configure the guest system log access by editing the **VirtualMachine** CR.

Prerequisites

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

Procedure

1. Edit the virtual machine manifest by running the following command:

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

2. Update the value of the **logSerialConsole** field. For example:

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          logSerialConsole: true
#...
```

   To enable access to the guest serial console log, set the **logSerialConsole** value to **true**.

3. Apply the new configuration to the VM by running the following command:

```
$ oc apply vm <vm_name>
```

4. Optional: If you edited a running VM, restart the VM to apply the new configuration. For example:

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

13.3.3.5. View guest system logs with the web console

To diagnose and troubleshoot issues with a virtual machine (VM) guest operating system, you can view the guest system logs by using the web console.

Guest system logs capture serial console output from the VM's boot process, kernel messages, and system-level events, which is useful when troubleshooting boot failures, kernel panics, or configuration issues that prevent standard access methods.

Prerequisites

- Guest system log access is enabled.

Procedure

1. Click **Virtualization → VirtualMachines** from the side menu.
2. Select a virtual machine to open the **VirtualMachine details** page.
3. Click the **Diagnostics** tab.
4. Click **Guest system logs** to load the serial console.

13.3.3.6. View guest system logs with the CLI

To diagnose and troubleshoot issues with a virtual machine (VM) guest operating system, you can view the guest system logs by running the **oc logs** command.

Guest system logs capture serial console output from the VM's boot process, kernel messages, and system-level events, which is useful when troubleshooting boot failures, kernel panics, or configuration issues that prevent standard access methods.

Prerequisites

- Guest system log access is enabled.
- You have installed the OpenShift CLI (**oc**).

Procedure

- View the logs by running the following command, substituting your own values for **<namespace>** and **<vm_name>**:

```
$ oc logs -n <namespace> -l kubevirt.io/domain=<vm_name> --tail=-1 -c guest-console-log
```

13.3.4. Log aggregation with Loki

To more easily diagnose and troubleshoot issues, you can aggregate and filter your logs.

13.3.4.1. View aggregated OpenShift Virtualization logs with Loki

You can use the Loki logging component to view aggregated logs for OpenShift Virtualization pods and containers in the web console. This is useful for troubleshooting issues and monitoring your OpenShift Virtualization environment.

Prerequisites

- You have installed the Loki Operator and deployed the **LokiStack** custom resource (CR).

Procedure

1. Navigate to **Observe → Logs** in the web console.
2. Select **application**, for **virt-launcher** pod logs, or **infrastructure**, for OpenShift Virtualization control plane pods and containers, from the log type list.
3. Click **Show Query** to display the query field.
4. Enter the LogQL query in the query field and click **Run Query** to display the filtered logs.

13.3.4.2. OpenShift Virtualization LogQL queries

To diagnose issues and monitor OpenShift Virtualization components, you can view and filter aggregated logs by running Loki Query Language (LogQL) queries on the **Observe → Logs** page in the web console.

The default log type is *infrastructure*. The **virt-launcher** log type is *application*.

Optional: You can include or exclude strings or regular expressions by using line filter expressions.



NOTE

If the query matches a large number of logs, the query might time out.

Table 13.2. OpenShift Virtualization LogQL example queries

Component	LogQL query
All	`{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster"`
cdi-apiserver **cdi-deployment** **cdi-operator**	`{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="storage"`
hco-operator	`{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="deployment"`
kubemacpool	`{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="network"`
virt-api **virt-controller** **virt-handler** **virt-operator**	`{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="compute"`
ssp-operator	`{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="schedule"`

Component	LogQL query
Container	`{log_type=~".+",kubernetes_container_name=~"<container>|<container>"} |json|kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster"` Specify one or more containers separated by a pipe (|).
virt-launcher	You must select **application** from the log type list before running this query. `{log_type=~".+", kubernetes_container_name="compute"}|json |!= "custom-ga-command"` **|!= "custom-ga-command"** excludes libvirt logs that contain the string**custom-ga-command**. (BZ#2177684)

You can filter log lines to include or exclude strings or regular expressions by using line filter expressions.

Table 13.3. Line filter expressions

Line filter expression	Description
|= "<string>"	Log line contains string
!= "<string>"	Log line does not contain string
|~ "<regex>"	Log line contains regular expression
!~ "<regex>"	Log line does not contain regular expression

Example 13.1. Example line filter expression

```
{log_type=~".+"}|json
|kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster"
|= "error" != "timeout"
```

13.3.4.3. Common error messages

Troubleshoot OpenShift Virtualization by reviewing common error messages found in the logs.

ErrImagePull or **ImagePullBackOff**

Indicates an incorrect deployment configuration or problems with the images that are referenced.

13.3.5. Data volume troubleshooting

To analyze and resolve issues, you can check the **Conditions** and **Events** sections of the **DataVolume** object.

13.3.5.1. Data volume conditions and events

To diagnose data volume issues, you can examine the **Conditions** and **Events** sections of the **oc describe** command output.

Run the following command to inspect the data volume:

```
$ oc describe dv <DataVolume>
```

The **Conditions** section displays the following **Types**:

- **Bound**
- **Running**
- **Ready**

The **Events** section provides the following additional information:

- **Type** of event
- **Reason** for logging
- **Source** of the event
- **Message** containing additional diagnostic information.

The output from **oc describe** does not always contains **Events**.

An event is generated when the **Status**, **Reason**, or **Message** changes. Both conditions and events react to changes in the state of the data volume.

For example, if you misspell the URL during an import operation, the import generates a 404 message. That message change generates an event with a reason. The output in the **Conditions** section is updated as well.

13.3.5.2. Data volume condition and event analysis

To determine the state of a data volume in relation to a Persistent Volume Claim (PVC) and whether or not an operation is actively running on the data volume, inspect the **Conditions** and **Events** section of the **oc describe** command output.

You might also receive messages that offer specific details about the status of the data volume, and how it came to be in its current state.

There are many different combinations of conditions. Each must be evaluated in its unique context.

Examples of various combinations follow.

- **Bound** - A successfully bound PVC displays in this example.
  Note that the **Type** is **Bound**, so the **Status** is **True**. If the PVC is not bound, the **Status** is **False**.

When the PVC is bound, an event is generated stating that the PVC is bound. In this case, the **Reason** is **Bound** and **Status** is **True**. The **Message** indicates which PVC owns the data volume.

Message, in the **Events** section, provides further details including how long the PVC has been bound (**Age**) and by what resource (**From**), in this case **datavolume-controller**.

Example output:

```
Status:
  Conditions:
    Last Heart Beat Time:  2020-07-15T03:58:24Z
    Last Transition Time:  2020-07-15T03:58:24Z
    Message:               PVC win10-rootdisk Bound
    Reason:                Bound
    Status:                True
    Type:                  Bound
...
  Events:
    Type    Reason    Age    From                Message
    ----    ------    ----   ----                -------
    Normal  Bound     24s    datavolume-controller  PVC example-dv Bound
```

- **Running** - In this case, note that **Type** is **Running** and **Status** is **False**, indicating that an event has occurred that caused an attempted operation to fail, changing the Status from **True** to **False**.
  However, note that **Reason** is **Completed** and the **Message** field indicates **Import Complete**.

  In the **Events** section, the **Reason** and **Message** contain additional troubleshooting information about the failed operation. In this example, the **Message** displays an inability to connect due to a **404**, listed in the **Events** section's first **Warning**.

  From this information, you conclude that an import operation was running, creating contention for other operations that are attempting to access the data volume.

  Example output:

```
Status:
  Conditions:
    Last Heart Beat Time:  2020-07-15T04:31:39Z
    Last Transition Time:  2020-07-15T04:31:39Z
    Message:               Import Complete
    Reason:                Completed
    Status:                False
    Type:                  Running
...
  Events:
    Type    Reason      Age              From                Message
    ----    ------      ----             ----                -------
    Warning  Error      12s (x2 over 14s)  datavolume-controller  Unable to connect
    to http data source: expected status code 200, got 404. Status: 404 Not Found
```

- **Ready** – If **Type** is **Ready** and **Status** is **True**, then the data volume is ready to be used, as in the following example. If the data volume is not ready to be used, the **Status** is **False**.

Example output:

```
Status:
  Conditions:
    Last Heart Beat Time: 2020-07-15T04:31:39Z
    Last Transition Time:  2020-07-15T04:31:39Z
    Status:               True
    Type:                 Ready
```

13.3.6. Additional resources

- OpenShift Dedicated events
- List of events
- View aggregated OpenShift Virtualization logs with Loki
- Connecting to virtual machine consoles
- List of known Windows BSOD issues on OpenShift Virtualization and RHEL KVM
- Recommendations when investigating Windows BSOD issues
- LogQL log queries

CHAPTER 14. BACKUP AND RESTORE

14.1. BACKUP AND RESTORE BY USING VM SNAPSHOTS

You can back up and restore virtual machines (VMs) by using snapshots.

Snapshots are supported by the following storage providers:

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

To create snapshots of a VM in the **Running** state with the highest integrity, install the QEMU guest agent if it is not included with your operating system. The QEMU guest agent is included with the default Red Hat templates.



IMPORTANT

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

Ensure that the QEMU guest agent is installed and running on the virtual machine before you take an online snapshot.

The QEMU guest agent stops responding to file system operations to ensure that the snapshot captures a consistent state.

The QEMU guest agent takes a consistent snapshot by attempting to quiesce the VM file system. This ensures that in-flight I/O is written to the disk before the snapshot is taken. If the guest agent is not present, quiescing is not possible and a best-effort snapshot is taken.

The conditions under which a snapshot is taken are reflected in the snapshot indications that are displayed in the web console or CLI. If these conditions do not meet your requirements, try creating the snapshot again or use an offline snapshot

14.1.1. VM snapshot concepts and components

A *snapshot* represents the state and data of a virtual machine (VM) at a specific point in time. You can use a snapshot to restore an existing VM to a previous state (represented by the snapshot) for backup and disaster recovery or to rapidly roll back to a previous development version.

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

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

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

You can perform the following snapshot actions:

- Create a new snapshot

- Create a clone of a virtual machine from a snapshot



IMPORTANT

Cloning a VM with a vTPM device attached to it or creating a new VM from its snapshot is not supported.

- List all snapshots attached to a specific VM
- Restore a VM from a snapshot
- Delete an existing VM snapshot

14.1.1.1. VM snapshot controller and custom resources

The VM snapshot feature introduces three new API objects defined as custom resource definitions (CRDs) for managing snapshots:

- **VirtualMachineSnapshot**: Represents a user request to create a snapshot. It contains information about the current state of the VM.
- **VirtualMachineSnapshotContent**: Represents a provisioned resource on the cluster (a snapshot). It is created by the VM snapshot controller and contains references to all resources required to restore the VM.
- **VirtualMachineRestore**: Represents a user request to restore a VM from a snapshot.

The VM snapshot controller binds a **VirtualMachineSnapshotContent** object with the **VirtualMachineSnapshot** object for which it was created, with a one-to-one mapping.

14.1.2. Application-consistent snapshots and backups

You can configure application-consistent snapshots and backups for Linux or Windows virtual machines (VMs) through a cycle of freezing and thawing. For any application, you can configure a script on a Linux VM or register on a Windows VM to be notified when a snapshot or backup is due to begin.

On a Linux VM, freeze and thaw processes trigger automatically when a snapshot is taken or a backup is started by using, for example, a plugin from Velero or another backup vendor. The freeze process, performed by QEMU Guest Agent (QEMU GA) freeze hooks, ensures that before the snapshot or backup of a VM occurs, all of the VM's filesystems are frozen and each appropriately configured application is informed that a snapshot or backup is about to start. This notification affords each application the opportunity to quiesce its state. Depending on the application, quiescing might involve temporarily refusing new requests, finishing in-progress operations, and flushing data to disk. The operating system is then directed to quiesce the filesystems by flushing outstanding writes to disk and freezing new write activity. All new connection requests are refused. When all applications have become inactive, the QEMU GA freezes the filesystems, and a snapshot is taken or a backup initiated. After the taking of the snapshot or start of the backup, the thawing process begins. Filesystems writing is reactivated and applications receive notification to resume normal operations.

The same cycle of freezing and thawing is available on a Windows VM. Applications register with the Volume Shadow Copy Service (VSS) to receive notifications that they should flush out their data because a backup or snapshot is imminent. Thawing of the applications after the backup or snapshot is complete returns them to an active state. For more details, see the Windows Server documentation about the Volume Shadow Copy Service.



IMPORTANT

Immediately after a Windows VM starts, the QEMU GA and VSS might not yet be fully responsive, and CPU usage might not yet have stabilized. Starting a backup or snapshot before CPU usage is idle and the guest agent reports as ready can cause the freeze or thaw hook to exceed its timeout and fail.

14.1.3. Create a snapshot by using the web console

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

Prerequisites

- The **snapshot** feature gate is enabled in the YAML configuration of the **kubevirt** CR.
- The VM snapshot includes disks that meet the following requirements:
  - The disks are data volumes or persistent volume claims.
  - The disks belong to a storage class that supports Container Storage Interface (CSI) volume snapshots.
  - The disks are *bound* to a persistent volume (PV) and *populated* with a datasource.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Select a VM to open the **VirtualMachine details** page.
3. Click the **Snapshots** tab and then click **Take Snapshot**.
   Alternatively, right-click the VM and select **Create snapshot** from the menu.
4. Enter the snapshot name.
5. Expand **Disks included in this Snapshot** to see the storage volumes to be included in the snapshot.
6. If your VM has disks that cannot be included in the snapshot and you wish to proceed, select **I am aware of this warning and wish to proceed**.
7. Click **Save**.

14.1.4. Create a snapshot by using the CLI

You can create a virtual machine (VM) snapshot for an offline or online VM by creating a **VirtualMachineSnapshot** object.

Prerequisites

- Ensure the **Snapshot** feature gate is enabled for the **kubevirt** CR by using the following command:

```
$ oc get kubevirt kubevirt-kubevirt-hyperconverged -n openshift-cnv -o yaml
```

Truncated output:

```
spec:
  developerConfiguration:
    featureGates:
      - Snapshot
```

- Ensure that the VM snapshot includes disks that meet the following requirements:
  - The disks are data volumes or persistent volume claims.
  - The disks belong to a storage class that supports Container Storage Interface (CSI) volume snapshots.
  - The disks are *bound* to a persistent volume (PV) and *populated* with a datasource.
- Install the OpenShift CLI (**oc**).
- Optional: Power down the VM for which you want to create a snapshot.

Procedure

1. Create a YAML file to define a **VirtualMachineSnapshot** object that specifies the name of the new **VirtualMachineSnapshot** and the name of the source VM as in the following example:

```
apiVersion: snapshot.kubevirt.io/v1beta1
kind: VirtualMachineSnapshot
metadata:
  name: <snapshot_name>
spec:
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: <vm_name>
```

2. Create the **VirtualMachineSnapshot** object:

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

The snapshot controller creates a **VirtualMachineSnapshotContent** object, binds it to the **VirtualMachineSnapshot**, and updates the **status** and **readyToUse** fields of the **VirtualMachineSnapshot** object.

Verification

1. Optional: During the snapshot creation process, you can use the **wait** command to monitor the status of the snapshot and wait until it is ready for use:
   a. Enter the following command:

```
$ oc wait <vm_name> <snapshot_name> --for condition=Ready
```

   b. Verify the status of the snapshot:

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



NOTE

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

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

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

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

2. Verify that the **VirtualMachineSnapshot** object is created and bound with **VirtualMachineSnapshotContent** and that the **readyToUse** flag is set to **true**:

```
$ oc describe vmsnapshot <snapshot_name>
```

Example output:

```
apiVersion: snapshot.kubevirt.io/v1beta1
kind: VirtualMachineSnapshot
metadata:
  creationTimestamp: "2020-09-30T14:41:51Z"
  finalizers:
  - snapshot.kubevirt.io/vmsnapshot-protection
  generation: 5
  name: mysnap
  namespace: default
  resourceVersion: "3897"
  selfLink:
/apis/snapshot.kubevirt.io/v1beta1/namespaces/default/virtualmachinesnapshots/my-
vmsnapshot
  uid: 28eedf08-5d6a-42c1-969c-2eda58e2a78d
spec:
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: my-vm
status:
  conditions:
  - lastProbeTime: null
```

```
    lastTransitionTime: "2020-09-30T14:42:03Z"
    reason: Operation complete
    status: "False"
    type: Progressing
  - lastProbeTime: null
    lastTransitionTime: "2020-09-30T14:42:03Z"
    reason: Operation complete
    status: "True"
    type: Ready
  creationTime: "2020-09-30T14:42:03Z"
  readyToUse: true
  sourceUID: 355897f3-73a0-4ec4-83d3-3c2df9486f4f
  virtualMachineSnapshotContentName: vmsnapshot-content-28eedf08-5d6a-42c1-969c-
2eda58e2a78d
  indications:
    - Online
  includedVolumes:
    - name: rootdisk
      kind: PersistentVolumeClaim
      namespace: default
    - name: datadisk1
      kind: DataVolume
      namespace: default
```

where:

status

The **status** field of the **Progressing** condition specifies if the snapshot is still being created. The **status** field of the **Ready** condition specifies if the snapshot creation process is complete.

readyToUse

Specifies if the snapshot is ready to be used.

virtualMachineSnapshotContentName

Specifies that the snapshot is bound to a **VirtualMachineSnapshotContent** object created by the snapshot controller.

indications

Specifies additional information about the snapshot, such as whether it is an online snapshot, or whether it was created with QEMU guest agent running.

includedVolumes

Lists the storage volumes that are part of the snapshot, as well as their parameters.

3. Check the **includedVolumes** section in the snapshot description to verify that the expected PVCs are included in the snapshot.

14.1.5. Verify online snapshots by using snapshot indications

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

Prerequisites

- You must have attempted to create an online VM snapshot.

Procedure

1. Display the output from the snapshot indications by performing one of the following actions:
   - Use the command line to view indicator output in the **status** stanza of the **VirtualMachineSnapshot** object YAML.
   - In the web console, click **VirtualMachineSnapshot → Status** in the **Snapshot details** screen.
2. Verify the status of your online VM snapshot by viewing the values of the **status.indications** parameter:
   - **Online** indicates that the VM was running during online snapshot creation.
   - **GuestAgent** indicates that the QEMU guest agent was active and successfully quiesced the guest file system for the online snapshot. This results in an application-consistent snapshot, preserving data integrity as if the applications had been gracefully shut down.
   - **NoGuestAgent** indicates that the QEMU guest agent was not installed, or not ready to quiesce the file system during the online snapshot. This results in a crash-consistent snapshot, which captures the VM's state like an abrupt power-off. As a result, application consistency is not guaranteed, which causes a risk of data issues for critical applications. For higher reliability, install and run the guest agent, or retry the snapshot.
   - **QuiesceFailed** indicates that an attempt to quiesce the file system failed during the online snapshot process. This means that the snapshot was created, but it is not necessarily application-consistent. To achieve proper consistency, retry the snapshot.

14.1.6. Restore a VM from a snapshot by using the web console

You can restore a virtual machine (VM) to a previous configuration represented by a snapshot in the OpenShift Dedicated web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Select a VM to open the **VirtualMachine details** page.
3. If the VM is running, click the Options menu  and select **Stop** to power it down.
4. Click the **Snapshots** tab to view a list of snapshots associated with the VM.
5. Select a snapshot to open the **Snapshot Details** screen.
6. Click the Options menu  and select **Restore VirtualMachine from snapshot**.
7. Optional: In the **Volume restore policy** section, select how restored persistent volume claims (PVCs) are named:

- **Prefix target name** - The restored PVC names use the target VM name as a prefix. This is the default setting.
- **In place** - The restored PVCs overwrite the original PVCs with the same names.
- **Randomize names** - The restored PVC names are randomly generated.

8. Click **Restore**.
9. Optional: You can also create a new VM based on the snapshot. To do so:
    a. In the Options menu  of the snapshot, select **Create VirtualMachine from Snapshot**
    b. Provide a name for the new VM.
    c. Click **Create**

14.1.7. Restore a VM from a snapshot by using the CLI

You can restore an existing virtual machine (VM) to a previous configuration by using the CLI. You can only restore from an offline VM snapshot.

Prerequisites

- You have installed the OpenShift CLI (**oc**).
- You have powered down the VM you want to restore.
- Optional: Adjust what happens if the target VM is not fully stopped (*ready*). To do so, set the **targetReadinessPolicy** parameter in the **vmrestore** YAML configuration to one of the following values:
    - **FailImmediate** - The restore process fails immediately if the VM is not ready.
    - **StopTarget** - If the VM is not ready, it gets stopped, and the restore process starts.
    - **WaitGracePeriod 5** - The restore process waits for a set amount of time, in minutes, for the VM to be ready. This is the default setting, with the default value set to 5 minutes.
    - **WaitEventually** - The restore process waits indefinitely for the VM to be ready.
- Optional: To control the naming of restored persistent volume claims (PVCs), you can set the **volumeRestorePolicy** parameter to one of the following values:
    - **PrefixTargetName** - The restored PVC names use the target VM name as a prefix: **<vm_name>-<volume_name>**.
    - **RandomizeNames** - The system generates the restored PVC names randomly: **restore-<uid>-<volume_name>**.
    - **InPlace** - The restored PVCs overwrite the original PVCs. The system deletes the original PVCs if they exist and creates new PVCs with the same names. This is the default setting.
- Optional: To control how restored persistent volume claims (PVCs) are named, you can set the **volumeRestorePolicy** parameter to one of the following values:

- **PrefixTargetName** - The restored PVC names use the target VM name as a prefix: **<vm_name>-<volume_name>**. This is the default setting.
- **RandomizeNames** - The restored PVC names are randomly generated: **restore-<uid>-<volume_name>**.
- **InPlace** - The restored PVCs overwrite the original PVCs. The original PVCs are deleted if they exist, and new PVCs are created with the same names.

Procedure

1. Create a YAML file to define a **VirtualMachineRestore** object that specifies the name of the VM you want to restore and the name of the snapshot to be used as the source as in the following example:

```
apiVersion: snapshot.kubevirt.io/v1beta1
kind: VirtualMachineRestore
metadata:
  name: <vm_restore>
spec:
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: <vm_name>
  virtualMachineSnapshotName: <snapshot_name>
  volumeRestorePolicy: PrefixTargetName
```

Where:

- **volumeRestorePolicy**: Optional. The volume restore policy determines how restored PVCs are named. Valid values are **PrefixTargetName** (default), **RandomizeNames**, or **InPlace**.

2. Optional: To customize the names, labels, and annotations of individual restored persistent volume claims (PVCs), add the **volumeRestoreOverrides** parameter to the YAML file:

```
apiVersion: snapshot.kubevirt.io/v1beta1
kind: VirtualMachineRestore
metadata:
  name: <vm_restore>
spec:
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: <vm_name>
  virtualMachineSnapshotName: <snapshot_name>
  volumeRestoreOverrides:
  - volumeName: <volume_name>
    restoreName: <custom_pvc_name>
    labels:
      custom-label: <label_value>
    annotations:
      custom-annotation: <annotation_value>
```

Where:

- **volumeName**: Required. The name of the volume from the snapshot to customize.
- **restoreName**: Optional. The custom name for the restored PVC. If not specified, the PVC name is determined by the **volumeRestorePolicy** setting.
- **labels**: Optional. Custom labels to add to the restored PVC. These labels are merged with any existing labels from the source PVC.
- **annotations**: Optional. Custom annotations to add to the restored PVC. These annotations are merged with any existing annotations from the source PVC.

3. Create the **VirtualMachineRestore** object:

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

The snapshot controller updates the status fields of the **VirtualMachineRestore** object and replaces the existing VM configuration with the snapshot content.

Verification

- Verify that the VM is restored to the previous state represented by the snapshot and that the **status.complete** flag is set to **true**:

```
$ oc get vmrestore <vm_restore>
```

Example output:

```
apiVersion: snapshot.kubevirt.io/v1beta1
kind: VirtualMachineRestore
metadata:
  creationTimestamp: "2020-09-30T14:46:27Z"
  generation: 5
  name: my-vmrestore
  namespace: default
  ownerReferences:
  - apiVersion: kubevirt.io/v1
    blockOwnerDeletion: true
    controller: true
    kind: VirtualMachine
    name: my-vm
    uid: 355897f3-73a0-4ec4-83d3-3c2df9486f4f
  resourceVersion: "5512"
  uid: 71c679a8-136e-46b0-b9b5-f57175a6a041
spec:
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: my-vm
  virtualMachineSnapshotName: my-vmsnapshot
status:
  complete: true
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2020-09-30T14:46:28Z"
    reason: Operation complete
```

```
    status: "False"
    type: Progressing
  - lastProbeTime: null
    lastTransitionTime: "2020-09-30T14:46:28Z"
    reason: Operation complete
    status: "True"
    type: Ready
  deletedDataVolumes:
  - test-dv1
  restoreTime: "2020-09-30T14:46:28Z"
  restores:
  - dataVolumeName: restore-71c679a8-136e-46b0-b9b5-f57175a6a041-datavolumedisk1
    persistentVolumeClaim: restore-71c679a8-136e-46b0-b9b5-f57175a6a041-
datavolumedisk1
    volumeName: datavolumedisk1
    volumeSnapshotName: vmsnapshot-28eedf08-5d6a-42c1-969c-2eda58e2a78d-volume-
datavolumedisk1
```



NOTE

If the **Progressing** condition has **status: "True"**, the VM is still being restored.

14.1.8. Delete a snapshot by using the web console

You can delete an existing virtual machine (VM) snapshot by using the web console.

Procedure

1. Navigate to **Virtualization → VirtualMachines** in the web console.
2. Select a VM to open the **VirtualMachine details** page.
3. Click the **Snapshots** tab to view a list of snapshots associated with the VM.
4. Click the Options menu  beside a snapshot and select **Delete snapshot**.
5. Click **Delete**.

14.1.9. Delete a snapshot by using the CLI

You can delete an existing virtual machine (VM) snapshot by deleting the appropriate **VirtualMachineSnapshot** object.

Prerequisites

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

Procedure

- Delete the **VirtualMachineSnapshot** object:

```
$ oc delete vmsnapshot <snapshot_name>
```

The snapshot controller deletes the **VirtualMachineSnapshot** along with the associated **VirtualMachineSnapshotContent** object.

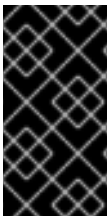
Verification

- Verify that the snapshot is deleted and no longer attached to this VM:

```
$ oc get vmsnapshot
```

14.2. BACKING UP AND RESTORING VIRTUAL MACHINES

Back up and restore virtual machines by using the OpenShift API for Data Protection.



IMPORTANT

Red Hat supports using OpenShift Virtualization 4.14 or later with OADP 1.3.x or later.

OADP versions earlier than 1.3.0 are not supported for back up and restore of OpenShift Virtualization.

14.2.1. Install and configure OADP with OpenShift Virtualization

As a cluster administrator, you can install the OpenShift API for Data Protection (OADP) with OpenShift Virtualization by installing the OADP Operator and configuring a backup location. You can then install the Data Protection Application to back up and restore virtual machines.

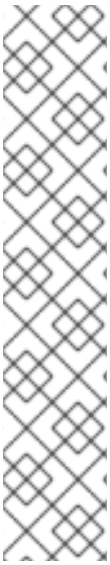


IMPORTANT

Red Hat only supports the combination of OADP versions 1.3.0 and later, and OpenShift Virtualization versions 4.14 and later.

OADP versions before 1.3.0 are not supported for back up and restore of OpenShift Virtualization.

To install the OADP Operator in a restricted network environment, you must first disable the default software catalog sources and mirror the Operator catalog.



NOTE

OpenShift API for Data Protection with OpenShift Virtualization supports the following backup and restore storage options:

- Container Storage Interface (CSI) backups
- Container Storage Interface (CSI) backups with DataMover

The following storage options are excluded:

- File system backup and restore
- Volume snapshot backup and restore

The latest version of the OADP Operator installs Velero 1.18.



WARNING

Red Hat support is limited to only the following options:

- CSI backups
- CSI backups with DataMover.

Prerequisites

- You have access to the cluster as a user with the **cluster-admin** role.

Procedure

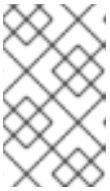
1. Install the OADP Operator according to the instructions for your storage provider.
2. Install the Data Protection Application (DPA) with the **kubevirt** and **openshift** OADP plug-ins.
3. Back up virtual machines by creating a **Backup** custom resource (CR).
   You restore the **Backup** CR by creating a **Restore** CR.

14.2.2. Install the Data Protection Application

You install the Data Protection Application (DPA) by creating an instance of the **DataProtectionApplication** API.

Prerequisites

- You have installed the OADP Operator.
- You have configured object storage as a backup location.
- If you use snapshots to back up PVs, your cloud provider supports either a native snapshot API or Container Storage Interface (CSI) snapshots.
- If the backup and snapshot locations use the same credentials, you have created a **Secret** with the default name for your storage provider: **cloud-credentials** for AWS, Multicloud Object Gateway (MCG), Red Hat OpenShift Data Foundation (ODF), IBM Cloud, or KubeVirt; **cloud-credentials-azure** for Azure; or **cloud-credentials-gcp** for GCP.
- If the backup and snapshot locations use different credentials, you have created two **Secrets**:
  - **Secret** with a custom name for the backup location. You add this **Secret** to the **DataProtectionApplication** CR.
  - **Secret** with another custom name for the snapshot location. You add this **Secret** to the **DataProtectionApplication** CR.



NOTE

If you do not want to specify backup or snapshot locations during the installation, you can create a default **Secret** with an empty **credentials-velero** file. If there is no default **Secret**, the installation will fail.

Procedure

1. Click **Ecosystem → Installed Operators** and select the OADP Operator.
2. Under **Provided APIs**, click **Create instance** in the **DataProtectionApplication** box.
3. Click **YAML View** and update the parameters of the **DataProtectionApplication** manifest for your storage provider.
4. Click **Create**.

Verification

1. Verify the installation by viewing the OpenShift API for Data Protection (OADP) resources by running the following command:

```
$ oc get all -n openshift-adp
```

```
NAME                                                READY   STATUS    RESTARTS   AGE
pod/oadp-operator-controller-manager-67d9494d47-6l8z8   2/2     Running   0          2m8s
pod/node-agent-9cq4q                                1/1     Running   0          94s
pod/node-agent-m4lts                                1/1     Running   0          94s
pod/node-agent-pv4kr                                1/1     Running   0          95s
pod/velero-588db7f655-n842v                         1/1     Running   0          95s

NAME                                                TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)    AGE
service/oadp-operator-controller-manager-metrics-service   ClusterIP   172.30.70.140
<none>        8443/TCP   2m8s
service/openshift-adp-velero-metrics-svc                   ClusterIP   172.30.10.0      <none>
8085/TCP   8h

NAME                                DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR   AGE
daemonset.apps/node-agent   3         3         3       3            3           <none>          96s

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/oadp-operator-controller-manager   1/1     1            1           2m9s
deployment.apps/velero                             1/1     1            1           96s

NAME                                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/oadp-operator-controller-manager-67d9494d47   1         1         1       2m9s
replicaset.apps/velero-588db7f655                         1         1         1       96s
```

2. Verify that the **DataProtectionApplication** (DPA) is reconciled by running the following command:

```
$ oc get dpa dpa-sample -n openshift-adp -o jsonpath='{.status}'
```

```
{"conditions":[{"lastTransitionTime":"2023-10-27T01:23:57Z","message":"Reconcile
complete","reason":"Complete","status":"True","type":"Reconciled"}]}
```

3. Verify the **type** is set to **Reconciled**.

4. Verify the backup storage location and confirm that the **PHASE** is **Available** by running the following command:

```
$ oc get backupstoragelocations.velero.io -n openshift-adp
```

```
NAME          PHASE      LAST VALIDATED   AGE     DEFAULT
dpa-sample-1  Available  1s               3d16h   true
```

14.2.2.1. DataProtectionApplication CR examples by storage provider

Configure the **DataProtectionApplication** (DPA) custom resource (CR) for your storage provider. The following examples show the DPA manifest for each supported provider with the relevant configuration parameters.

The following parameters are common across all provider configurations:

namespace

Specifies the default namespace for OADP which is **openshift-adp**. The namespace is a variable and is configurable.

openshift

Specifies that the **openshift** plugin is mandatory.

resourceTimeout

Specifies how many minutes to wait for several Velero resources such as Velero CRD availability, volumeSnapshot deletion, and backup repository availability, before timeout occurs. The default is 10m.

nodeAgent

Specifies the administrative agent that routes the administrative requests to servers.

enable

Set this value to **true** if you want to enable **nodeAgent** and perform File System Backup.

uploaderType

Specifies the uploader type. Enter **kopia** or **restic** as your uploader. You cannot change the selection after the installation. For the Built-in DataMover you must use Kopia. The **nodeAgent** deploys a daemon set, which means that the **nodeAgent** pods run on each working node. You can configure File System Backup by adding **spec.defaultVolumesToFsBackup: true** to the **Backup** CR.

nodeSelector

Specifies the nodes on which Kopia or Restic are available. By default, Kopia or Restic run on all nodes.

bucket

Specifies a bucket as the backup storage location. If the bucket is not a dedicated bucket for Velero backups, you must specify a prefix.

prefix

Specifies a prefix for Velero backups, for example, **velero**, if the bucket is used for multiple purposes.

The following sections provide DPA configuration examples and provider-specific parameters for each supported storage provider.

Amazon Web Services (AWS)

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
  namespace: openshift-adp
spec:
  configuration:
    velero:
      defaultPlugins:
        - openshift
        - aws
      resourceTimeout: 10m
    nodeAgent:
      enable: true
      uploaderType: kopia
      podConfig:
        nodeSelector: <node_selector>
  backupLocations:
    - name: default
      velero:
        provider: aws
        default: true
        objectStorage:
          bucket: <bucket_name>
          prefix: <prefix>
        config:
          region: <region>
          profile: "default"
          s3ForcePathStyle: "true"
          s3Url: <s3_url>
        credential:
          key: cloud
          name: cloud-credentials
  snapshotLocations:
    - name: default
      velero:
        provider: aws
        config:
          region: <region>
          profile: "default"
        credential:
          key: cloud
          name: cloud-credentials
```

s3ForcePathStyle

Specifies whether to force path style URLs for S3 objects (Boolean). Not Required for AWS S3. Required only for S3 compatible storage.

s3Url

Specifies the URL of the object store that you are using to store backups. Not required for AWS S3. Required only for S3 compatible storage.

name

Specifies the name of the **Secret** object that you created. If you do not specify this value, the default name, **cloud-credentials**, is used. If you specify a custom name, the custom name is used for the backup location.

snapshotLocations

Specifies a snapshot location, unless you use CSI snapshots or a File System Backup (FSB) to back up PVs.

region

Specifies that the snapshot location must be in the same region as the PVs.

IBM Cloud

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  namespace: openshift-adp
  name: <dpa_name>
spec:
  configuration:
    velero:
      defaultPlugins:
      - openshift
      - aws
      - csi
  backupLocations:
    - velero:
        provider: aws
        default: true
        objectStorage:
          bucket: <bucket_name>
          prefix: velero
        config:
          insecureSkipTLSVerify: 'true'
          profile: default
          region: <region_name>
          s3ForcePathStyle: 'true'
          s3Url: <s3_url>
        credential:
          key: cloud
          name: cloud-credentials
```

provider

Specifies that the provider is **aws** when you use IBM Cloud as a backup storage location.

bucket

Specifies the IBM Cloud Object Storage (COS) bucket name.

region

Specifies the COS region name, for example, **eu-gb**.

s3Url

Specifies the S3 URL of the COS bucket. For example, **http://s3.eu-gb.cloud-object-storage.appdomain.cloud**. Here, **eu-gb** is the region name. Replace the region name according to your bucket region.

name

Specifies the name of the secret you created by using the access key and the secret access key from the **HMAC** credentials.

Microsoft Azure

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
  namespace: openshift-adp
spec:
  configuration:
    velero:
      defaultPlugins:
        - azure
        - openshift
      resourceTimeout: 10m
    nodeAgent:
      enable: true
      uploaderType: kopia
      podConfig:
        nodeSelector: <node_selector>
  backupLocations:
    - velero:
        config:
          resourceGroup: <azure_resource_group>
          storageAccount: <azure_storage_account_id>
          subscriptionId: <azure_subscription_id>
        credential:
          key: cloud
          name: cloud-credentials-azure
        provider: azure
        default: true
        objectStorage:
          bucket: <bucket_name>
          prefix: <prefix>
  snapshotLocations:
    - velero:
        config:
          resourceGroup: <azure_resource_group>
          subscriptionId: <azure_subscription_id>
          incremental: "true"
        name: default
        provider: azure
        credential:
          key: cloud
          name: cloud-credentials-azure
```

resourceGroup

Specifies the Azure resource group.

storageAccount

Specifies the Azure storage account ID.

subscriptionId

Specifies the Azure subscription ID.

name

Specifies the name of the **Secret** object. If you do not specify this value, the default name, **cloud-credentials-azure**, is used. If you specify a custom name, the custom name is used for the backup location.

snapshotLocations

Specifies the snapshot location. You do not need to specify a snapshot location if you use CSI snapshots or Restic to back up PVs.

Google Cloud

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
  namespace: <OPERATOR_INSTALL_NS>
spec:
  configuration:
    velero:
      defaultPlugins:
        - gcp
        - openshift
      resourceTimeout: 10m
    nodeAgent:
      enable: true
      uploaderType: kopia
      podConfig:
        nodeSelector: <node_selector>
  backupLocations:
    - velero:
        provider: gcp
        default: true
        credential:
          key: cloud
          name: cloud-credentials-gcp
        objectStorage:
          bucket: <bucket_name>
          prefix: <prefix>
  snapshotLocations:
    - velero:
        provider: gcp
        default: true
        config:
          project: <project>
          snapshotLocation: us-west1
        credential:
          key: cloud
          name: cloud-credentials-gcp
  backupImages: true
```

key

Specifies the secret key that contains credentials. For Google workload identity federation cloud authentication use **service_account.json**.

name

Specifies the secret name that contains credentials. If you do not specify this value, the default name, **cloud-credentials-gcp**, is used.

snapshotLocations

Specifies a snapshot location, unless you use CSI snapshots or Restic to back up PVs.

snapshotLocation

Specifies that the snapshot location must be in the same region as the PVs.

backupImages

Specifies that Google workload identity federation supports internal image backup. Set this field to **false** if you do not want to use image backup.

MCG (Multicloud Object Gateway)

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
  namespace: openshift-adp
spec:
  configuration:
    velero:
      defaultPlugins:
        - aws
        - openshift
      resourceTimeout: 10m
    nodeAgent:
      enable: true
      uploaderType: kopia
      podConfig:
        nodeSelector: <node_selector>
  backupLocations:
    - velero:
        config:
          profile: "default"
          region: <region_name>
          s3Url: <url>
          insecureSkipTLSVerify: "true"
          s3ForcePathStyle: "true"
        provider: aws
        default: true
        credential:
          key: cloud
          name: cloud-credentials
        objectStorage:
          bucket: <bucket_name>
          prefix: <prefix>
```

aws

Specifies that an object store plugin corresponding to your storage locations is required. For all S3 providers, the required plugin is **aws**. For Azure and Google Cloud object stores, the **azure** or **gcp** plugin is required.

region

Specifies the region, following the naming convention of the documentation of your object storage server.

s3Url

Specifies the URL of the S3 endpoint.

name

Specifies the name of the **Secret** object that you created. If you do not specify this value, the default name, **cloud-credentials**, is used. If you specify a custom name, the custom name is used for the backup location.

Red Hat OpenShift Data Foundation (ODF)

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
  namespace: openshift-adp
spec:
  configuration:
    velero:
      defaultPlugins:
        - aws
        - kubevirt
        - csi
        - openshift
      resourceTimeout: 10m
    nodeAgent:
      enable: true
      uploaderType: kopia
      podConfig:
        nodeSelector: <node_selector>
  backupLocations:
    - velero:
        provider: aws
        default: true
        credential:
          key: cloud
          name: <default_secret>
        objectStorage:
          bucket: <bucket_name>
          prefix: <prefix>
```

aws

Specifies that an object store plugin corresponding to your storage locations is required. For all S3 providers, the required plugin is **aws**. For Azure and Google Cloud object stores, the **azure** or **gcp** plugin is required.

kubevirt

Optional: The **kubevirt** plugin is used with OpenShift Virtualization.

csi

Specifies the **csi** default plugin if you use CSI snapshots to back up PVs. The **csi** plugin uses the Velero CSI beta snapshot APIs. You do not need to configure a snapshot location.

provider

Specifies the backup provider.

name

Specifies the correct default name for the **Secret**, for example, **cloud-credentials-gcp**, if you use a default plugin for the backup provider. If specifying a custom name, then the custom name is used for the backup location. If you do not specify a **Secret** name, the default name is used.

KubeVirt (OpenShift Virtualization)

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
  namespace: openshift-adp
spec:
  configuration:
    velero:
      defaultPlugins:
        - kubevirt
        - gcp
        - csi
        - openshift
      resourceTimeout: 10m
    nodeAgent:
      enable: true
      uploaderType: kopia
      podConfig:
        nodeSelector: <node_selector>
  backupLocations:
    - velero:
        provider: gcp
        default: true
        credential:
          key: cloud
          name: <default_secret>
        objectStorage:
          bucket: <bucket_name>
          prefix: <prefix>
```

kubevirt

Specifies that the **kubevirt** plugin is mandatory for OpenShift Virtualization.

gcp

Specifies the plugin for the backup provider, for example, **gcp**, if it exists.

csi

Specifies that the **csi** plugin is mandatory for backing up PVs with CSI snapshots. The **csi** plugin uses the Velero CSI beta snapshot APIs. You do not need to configure a snapshot location.

provider

Specifies the backup provider.

name

Specifies the correct default name for the **Secret**, for example, **cloud-credentials-gcp**, if you use a default plugin for the backup provider. If specifying a custom name, then the custom name is used for the backup location. If you do not specify a **Secret** name, the default name is used.