Red Hat Enterprise Linux 9

Configuring and managing virtualization

Setting up your host, creating and administering virtual machines, and understanding virtualization features in Red Hat Enterprise Linux 9
Setting up your host, creating and administering virtual machines, and understanding virtualization features in Red Hat Enterprise Linux 9
Abstract

This document describes how to manage virtualization in Red Hat Enterprise Linux 9 (RHEL 9). In addition to general information about virtualization, it describes how to manage virtualization using command-line utilities, as well as using the web console.
Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE ................................................................. 8

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION ..................................... 9

CHAPTER 1. INTRODUCING VIRTUALIZATION IN RHEL ....................................... 10
1.1. WHAT IS VIRTUALIZATION? ................................................................. 10
1.2. ADVANTAGES OF VIRTUALIZATION .................................................... 10
1.3. VIRTUAL MACHINE COMPONENTS AND THEIR INTERACTION ..................... 11
1.4. TOOLS AND INTERFACES FOR VIRTUALIZATION MANAGEMENT ............... 12
1.5. RED HAT VIRTUALIZATION SOLUTIONS ............................................. 13

CHAPTER 2. ENABLING VIRTUALIZATION ....................................................... 14
2.1. ENABLING VIRTUALIZATION ON AMD64 AND INTEL 64 ......................... 14
2.2. ENABLING VIRTUALIZATION ON IBM Z .............................................. 15
2.3. ENABLING VIRTUALIZATION ON ARM 64 ........................................... 16

CHAPTER 3. CREATING VIRTUAL MACHINES ................................................. 19
3.1. CREATING VIRTUAL MACHINES USING THE COMMAND-LINE INTERFACE ... 19
3.2. CREATING VIRTUAL MACHINES AND INSTALLING GUEST OPERATING SYSTEMS USING THE WEB CONSOLE .................................................. 22
    3.2.1. Creating virtual machines using the web console .......................... 22
    3.2.2. Creating virtual machines by importing disk images using the web console 24
    3.2.3. Installing guest operating systems using the web console .......... 25
    3.2.4. Creating virtual machines with cloud image authentication using the web console 26

CHAPTER 4. STARTING VIRTUAL MACHINES .................................................. 30
4.1. STARTING A VIRTUAL MACHINE USING THE COMMAND-LINE INTERFACE ... 30
4.2. STARTING VIRTUAL MACHINES USING THE WEB CONSOLE .................. 31
4.3. STARTING VIRTUAL MACHINES AUTOMATICALLY WHEN THE HOST STARTS ..... 31

CHAPTER 5. CONNECTING TO VIRTUAL MACHINES ...................................... 34
5.1. INTERACTING WITH VIRTUAL MACHINES USING THE WEB CONSOLE ........... 34
    5.1.1. Viewing the virtual machine graphical console in the web console .. 34
    5.1.2. Viewing the graphical console in a remote viewer using the web console 35
    5.1.3. Viewing the virtual machine serial console in the web console ...... 37
5.2. OPENING A VIRTUAL MACHINE GRAPHICAL CONSOLE USING VIRT VIEWER 38
5.3. CONNECTING TO A VIRTUAL MACHINE USING SSH ............................. 40
5.4. OPENING A VIRTUAL MACHINE SERIAL CONSOLE ............................... 41
5.5. SETTING UP EASY ACCESS TO REMOTE VIRTUALIZATION HOSTS ............ 42

CHAPTER 6. SHUTTING DOWN VIRTUAL MACHINES ....................................... 45
6.1. SHUTTING DOWN A VIRTUAL MACHINE USING THE COMMAND-LINE INTERFACE 45
6.2. SHUTTING DOWN AND RESTARTING VIRTUAL MACHINES USING THE WEB CONSOLE 45
    6.2.1. Shutting down virtual machines in the web console .................. 45
    6.2.2. Restarting virtual machines using the web console .................. 46
    6.2.3. Sending non-maskable interrupts to VMs using the web console .... 46

CHAPTER 7. DELETING VIRTUAL MACHINES ............................................... 48
7.1. DELETING VIRTUAL MACHINES USING THE COMMAND LINE INTERFACE ........ 48
7.2. DELETING VIRTUAL MACHINES USING THE WEB CONSOLE .................... 48

CHAPTER 8. MANAGING VIRTUAL MACHINES IN THE WEB CONSOLE ............... 50
8.1. OVERVIEW OF VIRTUAL MACHINE MANAGEMENT USING THE WEB CONSOLE 50
8.2. SETTING UP THE WEB CONSOLE TO MANAGE VIRTUAL MACHINES ............ 50
13.6.3. Removing ISO images from virtual optical drives 103
13.6.4. Removing optical drives from virtual machines 103
13.7. MANAGING SR-IOV DEVICES 104
13.7.1. What is SR-IOV? 104
13.7.2. Attaching SR-IOV networking devices to virtual machines 106
13.7.3. Supported devices for SR-IOV assignment 109
13.8. ATTACHING DASD DEVICES TO VIRTUAL MACHINES ON IBM Z 109

CHAPTER 14. MANAGING STORAGE FOR VIRTUAL MACHINES 113
14.1. UNDERSTANDING VIRTUAL MACHINE STORAGE 113
14.1.1. Introduction to storage pools 113
14.1.2. Introduction to storage volumes 114
14.1.3. Storage management using libvirt 114
14.1.4. Overview of storage management 114
14.1.5. Supported and unsupported storage pool types 115
14.2. MANAGING VIRTUAL MACHINE STORAGE POOLS USING THE CLI 115
14.2.1. Viewing storage pool information using the CLI 116
14.2.2. Creating directory-based storage pools using the CLI 116
14.2.3. Creating disk-based storage pools using the CLI 118
14.2.4. Creating filesystem-based storage pools using the CLI 120
14.2.5. Creating iSCSI-based storage pools using the CLI 122
14.2.6. Creating LVM-based storage pools using the CLI 123
14.2.7. Creating NFS-based storage pools using the CLI 125
14.2.8. Creating SCSI-based storage pools with vHBA devices using the CLI 126
14.2.9. Deleting storage pools using the CLI 128
14.3. MANAGING VIRTUAL MACHINE STORAGE POOLS USING THE WEB CONSOLE 129
14.3.1. Viewing storage pool information using the web console 129
14.3.2. Creating directory-based storage pools using the web console 131
14.3.3. Creating NFS-based storage pools using the web console 132
14.3.4. Creating iSCSI-based storage pools using the web console 134
14.3.5. Creating disk-based storage pools using the web console 135
14.3.6. Creating LVM-based storage pools using the web console 137
14.3.7. Removing storage pools using the web console 139
14.3.8. Deactivating storage pools using the web console 141
14.4. PARAMETERS FOR CREATING STORAGE POOLS 141
14.4.1. Directory-based storage pool parameters 142
14.4.2. Disk-based storage pool parameters 142
14.4.3. Filesystem-based storage pool parameters 143
14.4.4. iSCSI-based storage pool parameters 144
14.4.5. LVM-based storage pool parameters 146
14.4.6. NFS-based storage pool parameters 147
14.4.7. Parameters for SCSI-based storage pools with vHBA devices 148
14.5. MANAGING VIRTUAL MACHINE STORAGE VOLUMES USING THE CLI 150
14.5.1. Viewing storage volume information using the CLI 150
14.5.2. Creating and assigning storage volumes using the CLI 151
14.5.3. Deleting storage volumes using the CLI 152
14.6. MANAGING VIRTUAL MACHINE STORAGE VOLUMES USING THE WEB CONSOLE 153
14.6.1. Creating storage volumes using the web console 154
14.6.2. Removing storage volumes using the web console 155
14.7. MANAGING VIRTUAL MACHINE STORAGE DISKS USING THE WEB CONSOLE 157
14.7.1. Viewing virtual machine disk information in the web console 157
14.7.2. Adding new disks to virtual machines using the web console 158
14.7.3. Attaching existing disks to virtual machines using the web console 159
### Red Hat Enterprise Linux 9 Configuring and managing virtualization

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.7.4. Detaching disks from virtual machines using the web console</td>
<td>161</td>
</tr>
<tr>
<td>14.8. SECURING ISCSI STORAGE POOLS WITH LIBVIRT SECRETS</td>
<td>162</td>
</tr>
<tr>
<td>14.9. CREATING VHBAS</td>
<td>163</td>
</tr>
<tr>
<td><strong>CHAPTER 15. MANAGING GPU DEVICES IN VIRTUAL MACHINES</strong></td>
<td>166</td>
</tr>
<tr>
<td>15.1. ASSIGNING A GPU TO A VIRTUAL MACHINE</td>
<td>166</td>
</tr>
<tr>
<td>15.2. MANAGING NVIDIA VGPU DEVICES</td>
<td>168</td>
</tr>
<tr>
<td>15.2.1. Setting up NVIDIA vGPU devices</td>
<td>169</td>
</tr>
<tr>
<td>15.2.2. Removing NVIDIA vGPU devices</td>
<td>172</td>
</tr>
<tr>
<td>15.2.3. Obtaining NVIDIA vGPU information about your system</td>
<td>173</td>
</tr>
<tr>
<td>15.2.4. Remote desktop streaming services for NVIDIA vGPU</td>
<td>174</td>
</tr>
<tr>
<td>15.2.5. Additional resources</td>
<td>175</td>
</tr>
<tr>
<td><strong>CHAPTER 16. CONFIGURING VIRTUAL MACHINE NETWORK CONNECTIONS</strong></td>
<td>176</td>
</tr>
<tr>
<td>16.1. UNDERSTANDING VIRTUAL NETWORKING</td>
<td>176</td>
</tr>
<tr>
<td>16.1.1. How virtual networks work</td>
<td>176</td>
</tr>
<tr>
<td>16.1.2. Virtual networking default configuration</td>
<td>177</td>
</tr>
<tr>
<td>16.2. USING THE WEB CONSOLE FOR MANAGING VIRTUAL MACHINE NETWORK INTERFACES</td>
<td>178</td>
</tr>
<tr>
<td>16.2.1. Viewing and editing virtual network interface information in the web console</td>
<td>178</td>
</tr>
<tr>
<td>16.2.2. Adding and connecting virtual network interfaces in the web console</td>
<td>180</td>
</tr>
<tr>
<td>16.2.3. Disconnecting and removing virtual network interfaces in the web console</td>
<td>180</td>
</tr>
<tr>
<td>16.3. RECOMMENDED VIRTUAL MACHINE NETWORKING CONFIGURATIONS USING THE COMMAND-LINE INTERFACE</td>
<td>180</td>
</tr>
<tr>
<td>16.3.1. Configuring externally visible virtual machines using the command-line interface</td>
<td>181</td>
</tr>
<tr>
<td>16.3.2. Isolating virtual machines from each other using the command-line interface</td>
<td>182</td>
</tr>
<tr>
<td>16.4. RECOMMENDED VIRTUAL MACHINE NETWORKING CONFIGURATIONS USING THE WEB CONSOLE</td>
<td>184</td>
</tr>
<tr>
<td>16.4.1. Configuring externally visible virtual machines using the web console</td>
<td>184</td>
</tr>
<tr>
<td>16.4.2. Isolating virtual machines from each other using the web console</td>
<td>186</td>
</tr>
<tr>
<td>16.5. TYPES OF VIRTUAL MACHINE NETWORK CONNECTIONS</td>
<td>187</td>
</tr>
<tr>
<td>16.5.1. Virtual networking with network address translation</td>
<td>187</td>
</tr>
<tr>
<td>16.5.2. Virtual networking in routed mode</td>
<td>188</td>
</tr>
<tr>
<td>16.5.3. Virtual networking in bridged mode</td>
<td>189</td>
</tr>
<tr>
<td>16.5.4. Virtual networking in isolated mode</td>
<td>190</td>
</tr>
<tr>
<td>16.5.5. Virtual networking in open mode</td>
<td>191</td>
</tr>
<tr>
<td>16.5.6. Direct attachment of the virtual network device</td>
<td>191</td>
</tr>
<tr>
<td>16.5.7. Comparison of virtual machine connection types</td>
<td>191</td>
</tr>
<tr>
<td>16.6. BOOTING VIRTUAL MACHINES FROM A PXE SERVER</td>
<td>192</td>
</tr>
<tr>
<td>16.6.1. Setting up a PXE boot server on a virtual network</td>
<td>192</td>
</tr>
<tr>
<td>16.6.2. Booting virtual machines using PXE and a virtual network</td>
<td>194</td>
</tr>
<tr>
<td>16.6.3. Booting virtual machines using PXE and a bridged network</td>
<td>194</td>
</tr>
<tr>
<td>16.7. ADDITIONAL RESOURCES</td>
<td>195</td>
</tr>
<tr>
<td><strong>CHAPTER 17. OPTIMIZING VIRTUAL MACHINE PERFORMANCE</strong></td>
<td>196</td>
</tr>
<tr>
<td>17.1. WHAT INFLUENCES VIRTUAL MACHINE PERFORMANCE</td>
<td>196</td>
</tr>
<tr>
<td>The impact of virtualization on system performance</td>
<td>196</td>
</tr>
<tr>
<td>Reducing VM performance loss</td>
<td>196</td>
</tr>
<tr>
<td>17.2. OPTIMIZING VIRTUAL MACHINE PERFORMANCE USING TUNED</td>
<td>197</td>
</tr>
<tr>
<td>17.3. OPTIMIZING LIBVIRT DAEMONS</td>
<td>198</td>
</tr>
<tr>
<td>17.3.1. Types of libvirt daemons</td>
<td>198</td>
</tr>
<tr>
<td>17.3.2. Enabling modular libvirt daemons</td>
<td>198</td>
</tr>
<tr>
<td>17.4. CONFIGURING VIRTUAL MACHINE MEMORY</td>
<td>200</td>
</tr>
<tr>
<td>17.4.1. Adding and removing virtual machine memory using the web console</td>
<td>200</td>
</tr>
<tr>
<td>17.4.2. Adding and removing virtual machine memory using the command-line interface</td>
<td>201</td>
</tr>
</tbody>
</table>
21.1. GENERATING LIBVIRT DEBUG LOGS
   21.1.1. Understanding libvirt debug logs
   21.1.2. Enabling persistent settings for libvirt debug logs
   21.1.3. Enabling libvirt debug logs during runtime
   21.1.4. Attaching libvirt debug logs to support requests

21.2. DUMPING A VIRTUAL MACHINE CORE
   21.2.1. How virtual machine core dumping works
   21.2.2. Creating a virtual machine core dump file

21.3. BACKTRACING VIRTUAL MACHINE PROCESSES

CHAPTER 22. FEATURE SUPPORT AND LIMITATIONS IN RHEL 9 VIRTUALIZATION

22.1. HOW RHEL VIRTUALIZATION SUPPORT WORKS
22.2. RECOMMENDED FEATURES IN RHEL 9 VIRTUALIZATION
22.3. UNSUPPORTED FEATURES IN RHEL 9 VIRTUALIZATION
22.4. RESOURCE ALLOCATION LIMITS IN RHEL 9 VIRTUALIZATION
22.5. HOW VIRTUALIZATION ON IBM Z DIFFERS FROM AMD64 AND INTEL 64
22.6. HOW VIRTUALIZATION ON ARM 64 DIFFERS FROM AMD64 AND INTEL 64
22.7. AN OVERVIEW OF VIRTUALIZATION FEATURES SUPPORT IN RHEL 9
MAKING OPEN SOURCE MORE INCLUSIVE

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

We appreciate your input on our documentation. Please let us know how we could make it better.

- For simple comments on specific passages:
  1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.
  2. Use your mouse cursor to highlight the part of text that you want to comment on.
  3. Click the **Add Feedback** pop-up that appears below the highlighted text.
  4. Follow the displayed instructions.

- For submitting feedback via Bugzilla, create a new ticket:
  1. Go to the **Bugzilla** website.
  2. As the Component, use **Documentation**.
  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click **Submit Bug**.
CHAPTER 1. INTRODUCING VIRTUALIZATION IN RHEL

If you are unfamiliar with the concept of virtualization or its implementation in Linux, the following sections provide a general overview of virtualization in RHEL 9: its basics, advantages, components, and other possible virtualization solutions provided by Red Hat.

1.1. WHAT IS VIRTUALIZATION?

RHEL 9 provides the virtualization functionality, which enables a machine running RHEL 9 to host multiple virtual machines (VMs), also referred to as guests. VMs use the host’s physical hardware and computing resources to run a separate, virtualized operating system (guest OS) as a user-space process on the host’s operating system.

In other words, virtualization makes it possible to have operating systems within operating systems.

VMs enable you to safely test software configurations and features, run legacy software, or optimize the workload efficiency of your hardware. For more information on the benefits, see Advantages of virtualization.

For more information on what virtualization is, see the Red Hat Customer Portal.

Next steps

- To start using virtualization in Red Hat Enterprise Linux 9, see Enabling virtualization in Red Hat Enterprise Linux 9.

- In addition to Red Hat Enterprise Linux 9 virtualization, Red Hat offers a number of specialized virtualization solutions, each with a different user focus and features. For more information, see Red Hat virtualization solutions.

1.2. ADVANTAGES OF VIRTUALIZATION

Using virtual machines (VMs) has the following benefits in comparison to using physical machines:

- Flexible and fine-grained allocation of resources
  A VM runs on a host machine, which is usually physical, and physical hardware can also be assigned for the guest OS to use. However, the allocation of physical resources to the VM is done on the software level, and is therefore very flexible. A VM uses a configurable fraction of the host memory, CPUs, or storage space, and that configuration can specify very fine-grained resource requests.

  For example, what the guest OS sees as its disk can be represented as a file on the host file system, and the size of that disk is less constrained than the available sizes for physical disks.

- Software-controlled configurations
  The entire configuration of a VM is saved as data on the host, and is under software control. Therefore, a VM can easily be created, removed, cloned, migrated, operated remotely, or connected to remote storage.

- Separation from the host
  A guest OS runs on a virtualized kernel, separate from the host OS. This means that any OS can be installed on a VM, and even if the guest OS becomes unstable or is compromised, the host is not affected in any way.

- Space and cost efficiency
A single physical machine can host a large number of VMs. Therefore, it avoids the need for multiple physical machines to do the same tasks, and thus lowers the space, power, and maintenance requirements associated with physical hardware.

- **Software compatibility**
  Because a VM can use a different OS than its host, virtualization makes it possible to run applications that were not originally released for your host OS. For example, using a RHEL 7 guest OS, you can run applications released for RHEL 7 on a RHEL 9 host system.

**NOTE**
Not all operating systems are supported as a guest OS in a RHEL 9 host. For details, see Recommended features in RHEL 9 virtualization.

### 1.3. VIRTUAL MACHINE COMPONENTS AND THEIR INTERACTION

Virtualization in RHEL 9 consists of the following principal software components:

**Hypervisor**

The basis of creating virtual machines (VMs) in RHEL 9 is the hypervisor, a software layer that controls hardware and enables running multiple operating systems on a host machine.

The hypervisor includes the Kernel-based Virtual Machine (KVM) module and virtualization kernel drivers. These components ensure that the Linux kernel on the host machine provides resources for virtualization to user-space software.

At the user-space level, the **QEMU** emulator simulates a complete virtualized hardware platform that the guest operating system can run in, and manages how resources are allocated on the host and presented to the guest.

In addition, the **libvirt** software suite serves as a management and communication layer, making QEMU easier to interact with, enforcing security rules, and providing a number of additional tools for configuring and running VMs.

**XML configuration**

A host-based XML configuration file (also known as a domain XML file) determines all settings and devices in a specific VM. The configuration includes:

- Metadata such as the name of the VM, time zone, and other information about the VM.
- A description of the devices in the VM, including virtual CPUs (vCPUs), storage devices, input/output devices, network interface cards, and other hardware, real and virtual.
- VM settings such as the maximum amount of memory it can use, restart settings, and other settings about the behavior of the VM.

For more information on the contents of an XML configuration, see Sample virtual machine XML configuration.

**Component interaction**

When a VM is started, the hypervisor uses the XML configuration to create an instance of the VM as a user-space process on the host. The hypervisor also makes the VM process accessible to the host-based interfaces, such as the **virsh**, **virt-install**, and **guestfish** utilities, or the web console GUI.
When these virtualization tools are used, libvirt translates their input into instructions for QEMU. QEMU communicates the instructions to KVM, which ensures that the kernel appropriately assigns the resources necessary to carry out the instructions. As a result, QEMU can execute the corresponding user-space changes, such as creating or modifying a VM, or performing an action in the VM’s guest operating system.

NOTE

While QEMU is an essential component of the architecture, it is not intended to be used directly on RHEL 9 systems, due to security concerns. Therefore, using `qemu-*` commands is not supported by Red Hat, and it is highly recommended to interact with QEMU using libvirt.

For more information on the host-based interfaces, see Tools and interfaces for virtualization management.

Figure 1.1. RHEL 9 virtualization architecture

1.4. TOOLS AND INTERFACES FOR VIRTUALIZATION MANAGEMENT

You can manage virtualization in RHEL 9 using the command-line interface (CLI) or several graphical user interfaces (GUIs).

Command-line interface

The CLI is the most powerful method of managing virtualization in RHEL 9. Prominent CLI commands for virtual machine (VM) management include:

- **virsh** - A versatile virtualization command-line utility and shell with a great variety of purposes, depending on the provided arguments. For example:
  - Starting and shutting down a VM - `virsh start` and `virsh shutdown`
  - Listing available VMs - `virsh list`
Creating a VM from a configuration file - **virsh create**

Entering a virtualization shell - **virsh**

For more information, see the **virsh(1)** man page.

- **virt-install** - A CLI utility for creating new VMs. For more information, see the **virt-install(1)** man page.

- **virt-xml** - A utility for editing the configuration of a VM.

- **guestfish** - A utility for examining and modifying VM disk images. For more information, see the **guestfish(1)** man page.

**Graphical interfaces**

You can use the following GUIs to manage virtualization in RHEL 9:

- The **RHEL 9 web console**, also known as Cockpit, provides a remotely accessible and easy to use graphical user interface for managing VMs and virtualization hosts.

For instructions on basic virtualization management with the web console, see Managing virtual machines in the web console.

### 1.5. RED HAT VIRTUALIZATION SOLUTIONS

The following Red Hat products are built on top of RHEL 9 virtualization features and expand the KVM virtualization capabilities available in RHEL 9. In addition, many limitations of RHEL 9 virtualization do not apply to these products:

**OpenShift Virtualization**

Based on the KubeVirt technology, OpenShift Virtualization is a part of the Red Hat OpenShift Container Platform, and makes it possible to run virtual machines in containers.

For more information about OpenShift Virtualization see the Red Hat Hybrid Cloud pages.

**Red Hat OpenStack Platform (RHOSP)**

Red Hat OpenStack Platform offers an integrated foundation to create, deploy, and scale a secure and reliable public or private OpenStack cloud.

For more information about Red Hat OpenStack Platform, see the Red Hat Customer Portal or the Red Hat OpenStack Platform documentation suite.

**NOTE**

For details on virtualization features not supported in RHEL but supported in other Red Hat virtualization solutions, see Unsupported features in RHEL 9 virtualization.
CHAPTER 2. ENABLING VIRTUALIZATION

To use virtualization in RHEL 9, you must install virtualization packages and ensure your system is configured to host virtual machines (VMs). The specific steps to do this vary based on your CPU architecture.

2.1. ENABLING VIRTUALIZATION ON AMD64 AND INTEL 64

To set up a KVM hypervisor and create virtual machines (VMs) on an AMD64 or Intel 64 system running RHEL 9, follow the instructions below.

Prerequisites

- Red Hat Enterprise Linux 9 is installed and registered on your host machine.
- Your system meets the following hardware requirements to work as a virtualization host:
  - The architecture of your host machine supports KVM virtualization.
  - The following minimum system resources are available:
    - 6 GB free disk space for the host, plus another 6 GB for each intended VM.
    - 2 GB of RAM for the host, plus another 2 GB for each intended VM.

Procedure

1. Install the virtualization hypervisor packages.

   # dnf install qemu-kvm libvirt virt-install virt-viewer

2. Start the virtualization services:

   # for drv in qemu network nodedev nwfilter secret storage interface; do systemctl start virt${drv}d{,-ro,-admin}.socket; done

Verification

1. Verify that your system is prepared to be a virtualization host:

   # virt-host-validate

   [...]
   QEMU: Checking for device assignment IOMMU support : PASS
   QEMU: Checking if IOMMU is enabled by kernel : WARN (IOMMU appears to be disabled in kernel. Add intel_iommu=on to kernel cmdline arguments)
   LXC: Checking for Linux >= 2.6.26 : PASS
   [...]
   LXC: Checking for cgroup 'blkio' controller mount-point : PASS
   LXC: Checking if device /sys/fs/fuse/connections exists : FAIL (Load the 'fuse' module to enable /proc/ overrides)

2. If all virt-host-validate checks return a PASS value, your system is prepared for creating VMs. If any of the checks return a FAIL value, follow the displayed instructions to fix the problem.
If any of the checks return a **WARN** value, consider following the displayed instructions to improve virtualization capabilities.

**Troubleshooting**

- If KVM virtualization is not supported by your host CPU, `virt-host-validate` generates the following output:

  ```
  QEMU: Checking for hardware virtualization: FAIL (Only emulated CPUs are available, performance will be significantly limited)
  ```

  However, VMs on such a host system will fail to boot, rather than have performance problems.

  To work around this, you can change the `<domain type>` value in the XML configuration of the VM to `qemu`. Note, however, that Red Hat does not support VMs that use the `qemu` domain type, and setting this is highly discouraged in production environments.

**Next steps**

- Create a virtual machine on your RHEL 9 host

# 2.2. ENABLING VIRTUALIZATION ON IBM Z

To set up a KVM hypervisor and create virtual machines (VMs) on an IBM Z system running RHEL 9, follow the instructions below.

**Prerequisites**

- The following minimum system resources are available:
  - 6 GB free disk space for the host, plus another 6 GB for each intended VM.
  - 2 GB of RAM for the host, plus another 2 GB for each intended VM.
  - 4 CPUs on the host. VMs can generally run with a single assigned vCPU, but Red Hat recommends assigning 2 or more vCPUs per VM to avoid VMs becoming unresponsive during high load.

- Your IBM Z host system is using a z13 CPU or later.

- RHEL 9 is installed on a logical partition (LPAR). In addition, the LPAR supports the **SIE** virtualization functions.
  
  To verify this, search for `sie` in your `/proc/cpuinfo` file.

  ```
  # grep sie /proc/cpuinfo/
  features : esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs te sie
  ```

**Procedure**

1. Install the virtualization packages:

   ```
   # dnf install qemu-kvm libvirt virt-install
   ```

2. Start the virtualization services:
# for drv in qemu network nodevex nwfilter secret storage interface; do systemctl start virt${drv}d{,-ro,-admin}.socket; done

## Verification

1. Verify that your system is prepared to be a virtualization host.

```bash
# virt-host-validate
[...
QEMU: Checking if device /dev/kvm is accessible : PASS
QEMU: Checking if device /dev/vhost-net exists : PASS
QEMU: Checking if device /dev/net/tun exists : PASS
QEMU: Checking for cgroup 'memory' controller support : PASS
QEMU: Checking for cgroup 'memory' controller mount-point : PASS
[...]
```

2. If all `virt-host-validate` checks return a **PASS** value, your system is prepared for creating VMs. If any of the checks return a **FAIL** value, follow the displayed instructions to fix the problem.

   If any of the checks return a **WARN** value, consider following the displayed instructions to improve virtualization capabilities.

### Troubleshooting

- If KVM virtualization is not supported by your host CPU, `virt-host-validate` generates the following output:

  ```text
  QEMU: Checking for hardware virtualization: FAIL (Only emulated CPUs are available, performance will be significantly limited)
  
  However, VMs on such a host system will fail to boot, rather than have performance problems.
  
  To work around this, you can change the `<domain type>` value in the XML configuration of the VM to `qemu`. Note, however, that Red Hat does not support VMs that use the `qemu` domain type, and setting this is highly discouraged in production environments.
  ```

### 2.3. ENABLING VIRTUALIZATION ON ARM 64

To set up a KVM hypervisor for creating virtual machines (VMs) on an ARM 64 system running RHEL 9, follow the instructions below.

---

**IMPORTANT**

Virtualization on ARM 64 is only provided as a Technology Preview on RHEL 9, and is therefore unsupported.

**Prerequisites**

- The following minimum system resources are available:
  - 6 GB free disk space for the host, plus another 6 GB for each intended guest.
  - 4 GB of RAM for the host, plus another 4 GB for each intended guest.
Procedure

1. Install the virtualization packages:

   # dnf install qemu-kvm libvirt virt-install

2. Start the virtualization services:

   # for drv in qemu network nodedev nwfilter secret storage interface; do systemctl start virt${drv}d{,-ro,-admin}.socket; done

Verification

1. Verify that your system is prepared to be a virtualization host:

   # virt-host-validate

   QEMU: Checking if device /dev/vhost-net exists : PASS
   QEMU: Checking if device /dev/net/tun exists : PASS
   QEMU: Checking for cgroup 'memory' controller support : PASS
   QEMU: Checking for cgroup 'memory' controller mount-point : PASS
   QEMU: Checking for cgroup 'blkio' controller support : PASS
   QEMU: Checking for cgroup 'blkio' controller mount-point : PASS
   QEMU: Checking if IOMMU is enabled by kernel : WARN (Unknown if this platform has IOMMU support)

   2. If all virt-host-validate checks return a PASS value, your system is prepared for creating virtual machines.
   
   If any of the checks return a FAIL value, follow the displayed instructions to fix the problem.

   If any of the checks return a WARN value, consider following the displayed instructions to improve virtualization capabilities.

Troubleshooting

- If KVM virtualization is not supported by your host CPU, virt-host-validate generates the following output:

   QEMU: Checking for hardware virtualization: FAIL (Only emulated CPUs are available, performance will be significantly limited)

   However, VMs on such a host system will fail to boot, rather than have performance problems.

   To work around this, you can change the <domain type> value in the XML configuration of the VM to qemu. Note, however, that Red Hat does not support VMs that use the qemu domain type, and setting this is highly discouraged in production environments.

Next steps

- Creating virtual machines

Additional resources
• How virtualization on ARM 64 differs from AMD64 and Intel 64
CHAPTER 3. CREATING VIRTUAL MACHINES

To create a virtual machine (VM) in RHEL 9, use the command-line interface or the RHEL 9 web console.

Prerequisites

- Virtualization is installed and enabled on your system.

- You have sufficient amount of system resources to allocate to your VMs, such as disk space, RAM, or CPUs. The recommended values may vary significantly depending on the intended tasks and workload of the VMs.

WARNING

Installing from a host CD-ROM or DVD-ROM device is not possible in RHEL 9. If you select a CD-ROM or DVD-ROM as the installation source when using any VM installation method available in RHEL 9, the installation will fail. For more information, see the Red Hat Knowledgebase.

3.1. CREATING VIRTUAL MACHINES USING THE COMMAND-LINE INTERFACE

To create a virtual machine (VM) on your RHEL 9 host using the `virt-install` utility, follow the instructions below.

Prerequisites

- Virtualization is enabled on your host system.

- You have sufficient a amount of system resources to allocate to your VMs, such as disk space, RAM, or CPUs. The recommended values may vary significantly depending on the intended tasks and workload of the VMs.

- An operating system (OS) installation source is available locally or on a network. This can be one of the following:
  - An ISO image of an installation medium
  - A disk image of an existing VM installation
WARNING

Installing from a host CD-ROM or DVD-ROM device is not possible in RHEL 9. If you select a CD-ROM or DVD-ROM as the installation source when using any VM installation method available in RHEL 9, the installation will fail. For more information, see the Red Hat Knowledgebase.

Optional: A Kickstart file can be provided for faster and easier configuration of the installation.

Procedure

To create a VM and start its OS installation, use the `virt-install` command, along with the following mandatory arguments:

- The name of the new machine (`--name`)
- The amount of allocated memory (`--memory`)
- The number of allocated virtual CPUs (`--vcpus`)
- The type and size of the allocated storage (`--disk`)
- The type and location of the OS installation source (`--cdrom` or `--location`)

Based on the chosen installation method, the necessary options and values can vary. See below for examples:

- The following creates a VM named `demo-guest1` that installs the Windows 10 OS from an ISO image locally stored in the `/home/username/Downloads/Win10install.iso` file. This VM is also allocated with 2048 MiB of RAM and 2 vCPUs, and an 80 GiB qcow2 virtual disk is automatically configured for the VM.

```
# virt-install --name demo-guest1 --memory 2048 --vcpus 2 --disk size=80 --os-variant win10 --cdrom /home/username/Downloads/Win10install.iso
```

- The following creates a VM named `demo-guest2` that uses the `/home/username/Downloads/rhel9.iso` image to run a RHEL 9 OS from a live CD. No disk space is assigned to this VM, so changes made during the session will not be preserved. In addition, the VM is allocated with 4096 MiB of RAM and 4 vCPUs.

```
# virt-install --name demo-guest2 --memory 4096 --vcpus 4 --disk none --livedcd --os-variant rhel9.0 --cdrom /home/username/Downloads/rhel9.iso
```

- The following creates a RHEL 9 VM named `demo-guest3` that connects to an existing disk image, `/home/usernamebackup/disk.qcow2`. This is similar to physically moving a hard drive between machines, so the OS and data available to demo-guest3 are determined by how the image was handled previously. In addition, this VM is allocated with 2048 MiB of RAM and 2 vCPUs.

```
# virt-install --name demo-guest3 --memory 2048 --vcpus 2 --disk /home/username/backup/disk.qcow2
```
# virt-install --name demo-guest3 --memory 2048 --vcpus 2 --os-variant rhel9.0 --import --disk /home/username/backup/disk.qcow2

Note that the --os-variant option is highly recommended when importing a disk image. If it is not provided, the performance of the created VM will be negatively affected.

- The following creates a VM named demo-guest4 that installs from the http://example.com/OS-install URL. For the installation to start successfully, the URL must contain a working OS installation tree. In addition, the OS is automatically configured using the /home/username/ks.cfg kickstart file. This VM is also allocated with 2048 MiB of RAM, 2 vCPUs, and a 160 GiB qcow2 virtual disk.

# virt-install --name demo-guest4 --memory 2048 --vcpus 2 --disk size=160 --os-variant rhel9.0 --location http://example.com/OS-install --initrd-inject /home/username/ks.cfg --extra-args="inst.ks=file:/ks.cfg console=tty0 console=ttyS0,115200n8"

- The following creates a VM named demo-guest5 that installs from a RHEL9.iso image file in text-only mode, without graphics. It connects the guest console to the serial console. The VM has 16384 MiB of memory, 16 vCPUs, and 280 GiB disk. This kind of installation is useful when connecting to a host over a slow network link.

# virt-install --name demo-guest5 --memory 16384 --vcpus 16 --disk size=280 --os-variant rhel9.0 --location RHEL9.iso --graphics none --extra-args='console=ttyS0'

- The following creates a VM named demo-guest6, which has the same configuration as demo-guest5, but resides on the 10.0.0.1 remote host.

# virt-install --connect qemu+ssh://root@10.0.0.1/system --name demo-guest6 --memory 16384 --vcpus 16 --disk size=280 --os-variant rhel9.0 --location RHEL9.iso --graphics none --extra-args='console=ttyS0'

- The following creates a VM named demo-guest-7, which has the same configuration as demo-guest5, but for its storage, it uses a DASD mediated device mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8, and assigns it device number 1111.

# virt-install --name demo-guest7 --memory 16384 --vcpus 16 --disk size=280 --os-variant rhel9.0 --location RHEL9.iso --graphics none --hostdev mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8,address.type=ccw,address.cssid=0xfe,address.ssid=0x0,address.devno=0x1111,boot-order=1 --extra-args 'rd.dasd=0.0.1111'

Note that the name of the mediated device available for installation can be retrieved using the virsh nodedev-list --cap mdev command.

Verification

- If the VM is created successfully, a virt-viewer window opens with a graphical console of the VM and starts the guest OS installation.

Troubleshooting

- If virt-install fails with a cannot find default network error:
  
a. Ensure that the libvirt-daemon-config-network package is installed:
b. Verify that the libvirt default network is active and configured to start automatically:

```
# virsh net-list --all
Name   State     Autostart   Persistent
--------------------------------------------
default active yes yes
```

If it is not, activate the default network and set it to auto-start:

```
# virsh net-autostart default
Network default marked as autostarted

# virsh net-start default
Network default started
```

i. If activating the default network fails with the following error, the libvirt-daemon-config-network package has not been installed correctly.

```
error: failed to get network 'default'
error: Network not found: no network with matching name 'default'
```

To fix this, re-install libvirt-daemon-config-network.

```
# dnf reinstall libvirt-daemon-config-network
```

ii. If activating the default network fails with an error similar to the following, a conflict has occurred between the default network’s subnet and an existing interface on the host.

```
error: Failed to start network default
error: internal error: Network is already in use by interface ens2
```

To fix this, use the virsh net-edit default command and change the 192.168.122.* values in the configuration to a subnet not already in use on the host.

### Additional resources

- The man virt-install command

### 3.2. CREATING VIRTUAL MACHINES AND INSTALLING GUEST OPERATING SYSTEMS USING THE WEB CONSOLE

To manage virtual machines (VMs) in a GUI on a RHEL 9 host, use the web console. The following sections provide information on how to use the RHEL 9 web console to create VMs and install guest operating systems on them.

#### 3.2.1. Creating virtual machines using the web console
To create a virtual machine (VM) on the host machine to which the web console is connected, follow the instructions below.

**Prerequisites**

- Virtualization is enabled on your host system.
- The web console VM plug-in is installed on your system.
- You have sufficient amount of system resources to allocate to your VMs, such as disk space, RAM, or CPUs. The recommended values may vary significantly depending on the intended tasks and workload of the VMs.

**Procedure**

1. In the Virtual Machines interface of the web console, click **Create VM**. The Create new virtual machine dialog appears.

   ![Create new virtual machine dialog](image)

   - **Name**: Unique name
   - **Installation type**: Download an OS
   - **Operating system**: Choose an operating system
   - **Storage**: Create new volume
   - **Size**: 10 GiB
     - Up to 42.2 GiB available on the default location
   - **Memory**: 1 GiB
     - Up to 3.6 GiB available on the host
   - **Run unattended installation**: unchecked
   - **Immediately start VM**: checked
   - **Create** button
   - **Cancel** button

2. Enter the basic configuration of the VM you want to create.

   - **Name** - The name of the VM.
   - **Connection** - The type of libvirt connection, system or session. For more details, see [System and session connections](#).
   - **Installation type** - The installation can use a local installation medium, a URL, a PXE network boot, a cloud base image, or download an OS from a limited set of operating systems.
• **Operating system** - The VM’s operating system. Note that Red Hat provides support only for a limited set of guest operating systems.

• **Storage** - The type of storage with which to configure the VM.

• **Size** - The amount of storage space with which to configure the VM.

• **Memory** - The amount of memory with which to configure the VM.

• **Run unattended installation** - Whether or not to run the installation unattended. This option is available only when the Installation type is Download an OS.

• **Immediately Start VM** - Whether or not the VM will start immediately after it is created.

3. Click **Create**. The VM is created. If the Immediately Start VM checkbox is selected, the VM will immediately start and begin installing the guest operating system.

**Additional resources**

• Installing an operating system on a VM

### 3.2.2. Creating virtual machines by importing disk images using the web console

To create a virtual machine (VM) by importing a disk image of an existing VM installation, follow the instructions below.

**Prerequisites**

• The web console VM plug-in is installed on your system.

• You have sufficient a amount of system resources to allocate to your VMs, such as disk space, RAM, or CPUs. The recommended values can vary significantly depending on the intended tasks and workload of the VMs.

• Make sure you have a disk image of an existing VM installation

**Procedure**

1. In the Virtual Machines interface of the web console, click **Import VM**. The Import a virtual machine dialog appears.
2. Enter the basic configuration of the VM you want to create.

- **Name** - The name of the VM.
- **Connection** - The type of libvirt connection, system or session. For more details, see System and session connections.
- **Disk image** - The path to the existing disk image of a VM on the host system.
- **Operating system** - The VM's operating system. Note that Red Hat provides support only for a limited set of guest operating systems.
- **Memory** - The amount of memory with which to configure the VM.
- **Immediately start VM** - Whether or not the VM will start immediately after it is created.

3. Click **Import**.

### 3.2.3. Installing guest operating systems using the web console

The first time a virtual machine (VM) loads, you must install an operating system on the VM.

**NOTE**

If the *Immediately Start VM* checkbox in the Create New Virtual Machine dialog is checked, the installation routine of the operating system starts automatically when the VM is created.

**Prerequisites**

- The web console VM plug-in is installed on your system.
- A VM on which to install an operating system must be available.

**Procedure**

1. In the **Virtual Machines** interface, click the VM on which you want to install a guest OS.
A new page opens with basic information about the selected VM and controls for managing various aspects of the VM.

2. **Optional**: Change the firmware.

   **NOTE**

   You can change the firmware only if you had not selected the *Immediately Start VM* check box in the *Create New Virtual Machine* dialog, and the OS has not already been installed on the VM.

   a. Click the firmware.

   b. In the Change Firmware window, select the desired firmware.

   c. Click **Save**.

3. Click **Install**.

   The installation routine of the operating system runs in the VM console.

**Troubleshooting**

- If the installation routine fails, the VM must be deleted and recreated.

**3.2.4. Creating virtual machines with cloud image authentication using the web console**

By default, distro cloud images have no login accounts. However, using the RHEL web console, you can now create a virtual machine (VM) and specify the root and user account login credentials, which are then passed to cloud-init.

**Prerequisites**

- The web console VM plug-in is installed on your system.

- Virtualization is enabled on your host system.
• You have a sufficient amount of system resources to allocate to your VMs, such as disk space, RAM, or CPUs. The recommended values may vary significantly depending on the intended tasks and workload of the VMs.

Procedure

1. In the Virtual Machines interface of the web console, click Create VM. The Create new virtual machine dialog appears.

2. In the Name field, enter a name for the VM.

3. In the Installation type field, select Cloud base image.
4. In the **Installation source** field, set the path to the image file on your host system.

5. Enter the configuration for the VM that you want to create.
   - **Operating system** - The VM’s operating system. Note that Red Hat provides support only for a **limited set of guest operating systems**.
   - **Storage** - The type of storage with which to configure the VM.
   - **Size** - The amount of storage space with which to configure the VM.
   - **Memory** - The amount of memory with which to configure the VM.

6. Select **Set cloud init parameters**
   Set your cloud authentication credentials.
   - **Root password** - Enter a root password for your VM. Leave the field blank if you do not wish to set a root password.
   - **User login** - Enter a cloud-init user login.
- **User password** - Enter a password. Leave the field blank if you do not wish to set a password.

7. Click **Create**.
   The VM is created.

**Additional resources**

- **Installing an operating system on a VM**
CHAPTER 4. STARTING VIRTUAL MACHINES

To start a virtual machine (VM) in RHEL 9, you can use the command line interface or the web console GUI.

Prerequisites

- Before a VM can be started, it must be created and, ideally, also installed with an OS. For instruction to do so, see Creating virtual machines.

4.1. STARTING A VIRTUAL MACHINE USING THE COMMAND-LINE INTERFACE

You can use the command line interface (CLI) to start a shut-down virtual machine (VM) or restore a saved VM. Using the CLI, you can start both local and remote VMs.

Prerequisites

- An inactive VM that is already defined.
- The name of the VM.
- For remote VMs:
  - The IP address of the host where the VM is located.
  - Root access privileges to the host.

Procedure

- For a local VM, use the virsh start utility. For example, the following command starts the demo-guest1 VM.

  ```
  # virsh start demo-guest1
  Domain 'demo-guest1’ started
  ```

- For a VM located on a remote host, use the virsh start utility along with the QEMU+SSH connection to the host. For example, the following command starts the demo-guest1 VM on the 192.168.123.123 host.

  ```
  # virsh -c qemu+ssh://root@192.168.123.123/system start demo-guest1

  root@192.168.123.123's password:
  Domain 'demo-guest1’ started
  ```

Additional resources

- The `virsh start --help` command
- Setting up easy access to remote virtualization hosts
- Starting virtual machines automatically when the host starts
4.2. STARTING VIRTUAL MACHINES USING THE WEB CONSOLE

If a virtual machine (VM) is in the shut off state, you can start it using the RHEL 9 web console. You can also configure the VM to be started automatically when the host starts.

Prerequisites

- The web console VM plug-in is installed on your system.
- An inactive VM that is already defined.
- The name of the VM.

Procedure

1. In the Virtual Machines interface, click the VM you want to start. A new page opens with detailed information about the selected VM and controls for shutting down and deleting the VM.
2. Click Run. The VM starts, and you can connect to its console or graphical output.
3. Optional: To configure the VM to start automatically when the host starts, click the Autostart checkbox. If you use network interfaces that are not managed by libvirt, you must also make additional changes to the systemd configuration. Otherwise, the affected VMs might fail to start, see starting virtual machines automatically when the host starts.

Additional resources

- Shutting down virtual machines in the web console
- Restarting virtual machines using the web console

4.3. STARTING VIRTUAL MACHINES AUTOMATICALLY WHEN THE HOST STARTS

When a host with a running virtual machine (VM) restarts, the VM is shut down, and must be started again manually by default. To ensure a VM is active whenever its host is running, you can configure the VM to be started automatically.

Prerequisites

- A created virtual machine

Procedure

1. Use the virsh autostart utility to configure the VM to start automatically when the host starts. For example, the following command configures the demo-guest1 VM to start automatically.

```
# virsh autostart demo-guest1
Domain 'demo-guest1' marked as autostarted
```
2. If you use network interfaces that are not managed by **libvirt**, you must also make additional changes to the systemd configuration. Otherwise, the affected VMs might fail to start.

**NOTE**

These interfaces include for example:

- Bridge devices created by **NetworkManager**
- Networks configured to use `<forward mode='bridge'/>`

a. In the systemd configuration directory tree, create a `virtqemud.service.d` directory if it does not exist yet.

```
# mkdir -p /etc/systemd/system/virtqemud.service.d/
```

b. Create a `10-network-online.conf` systemd unit override file in the previously created directory. The content of this file overrides the default systemd configuration for the `virtqemud` service.

```
# touch /etc/systemd/system/virtqemud.service.d/10-network-online.conf
```

c. Add the following lines to the `10-network-online.conf` file. This configuration change ensures systemd starts the `virtqemud` service only after the network on the host is ready.

```
[Unit]
After=network-online.target
```

**Verification**

1. View the VM configuration, and check that the `autostart` option is enabled. For example, the following command displays basic information about the `demo-guest1` VM, including the `autostart` option.

```
# virsh dominfo demo-guest1
Id:             2
Name:           demo-guest1
UUID:           e46bc81c-74e2-406e-bd7a-67042bae80d1
OS Type:        hvm
State:          running
CPU(s):         2
CPU time:       385.9s
Max memory:     4194304 KiB
Used memory:    4194304 KiB
Persistent:     yes
Autostart:      enable
Managed save:   no
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c873,c919 (enforcing)
```

2. If you use network interfaces that are not managed by libvirt, check if the content of the `10-network-online.conf` file matches the following output.
$ cat /etc/systemd/system/virtqemud.service.d/10-network-online.conf
[Unit]
After=network-online.target

Additional resources

- The **virsh autostart --help** command
- **Starting virtual machines using the web console**.
CHAPTER 5. CONNECTING TO VIRTUAL MACHINES

To interact with a virtual machine (VM) in RHEL 9, you need to connect to it by doing one of the following:

- When using the web console interface, use the Virtual Machines pane in the web console interface. For more information, see Interacting with virtual machines using the web console.

- If you need to interact with a VM graphical display without using the web console, use the Virt Viewer application. For details, see Opening a virtual machine graphical console using Virt Viewer.

- When a graphical display is not possible or not necessary, use an SSH terminal connection.

- When the virtual machine is not reachable from your system by using a network, use the virsh console.

If the VMs to which you are connecting are on a remote host rather than a local one, you can optionally configure your system for more convenient access to remote hosts.

Prerequisites

- The VMs you want to interact with are installed and started.

5.1. INTERACTING WITH VIRTUAL MACHINES USING THE WEB CONSOLE

To interact with a virtual machine (VM) in the RHEL 9 web console, you need to connect to the VM’s console. These include both graphical and serial consoles.

- To interact with the VM’s graphical interface in the web console, use the graphical console.

- To interact with the VM’s graphical interface in a remote viewer, use the graphical console in remote viewers.

- To interact with the VM’s CLI in the web console, use the serial console.

5.1.1. Viewing the virtual machine graphical console in the web console

Using the virtual machine (VM) console interface, you can view the graphical output of a selected VM in the RHEL 9 web console.

Prerequisites

- The web console VM plug-in is installed on your system.

- Ensure that both the host and the VM support a graphical interface.

Procedure

1. In the Virtual Machines interface, click the VM whose graphical console you want to view. A new page opens with an Overview and a Console section for the VM.

2. Select VNC console in the console drop down menu.
The VNC console appears below the menu in the web interface.

The graphical console appears in the web interface.

3. Click **Expand**

You can now interact with the VM console using the mouse and keyboard in the same manner you interact with a real machine. The display in the VM console reflects the activities being performed on the VM.

**NOTE**

The host on which the web console is running may intercept specific key combinations, such as **Ctrl+Alt+Del**, preventing them from being sent to the VM.

To send such key combinations, click the **Send key** menu and select the key sequence to send.

For example, to send the **Ctrl+Alt+Del** combination to the VM, click the **Send key** and select the **Ctrl+Alt+Del** menu entry.

**Troubleshooting**

- If clicking in the graphical console does not have any effect, expand the console to full screen. This is a known issue with the mouse cursor offset.

**Additional resources**

- Viewing the graphical console in a remote viewer using the web console
- Viewing the virtual machine serial console in the web console

**5.1.2. Viewing the graphical console in a remote viewer using the web console**

Using the web console interface, you can display the graphical console of a selected virtual machine (VM) in a remote viewer, such as Virt Viewer.
NOTE

You can launch Virt Viewer from within the web console. Other VNC remote viewers can be launched manually.

Prerequisites

- The web console VM plug-in is installed on your system.
- Ensure that both the host and the VM support a graphical interface.
- Before you can view the graphical console in Virt Viewer, you must install Virt Viewer on the machine to which the web console is connected.

1. Click Launch remote viewer.
   A .vv file downloads.

2. Open the file to launch Virt Viewer.

NOTE

Remote Viewer is available on most operating systems. However, some browser extensions and plug-ins do not allow the web console to open Virt Viewer.

Procedure

1. In the Virtual Machines interface, click the VM whose graphical console you want to view. A new page opens with an Overview and a Console section for the VM.

2. Select Desktop Viewer in the console drop down menu.

3. Click Launch Remote Viewer.
   The graphical console opens in Virt Viewer.
You can interact with the VM console using the mouse and keyboard in the same manner you interact with a real machine. The display in the VM console reflects the activities being performed on the VM.

**NOTE**

The server on which the web console is running can intercept specific key combinations, such as **Ctrl+Alt+Del**, preventing them from being sent to the VM.

To send such key combinations, click the **Send key** menu and select the key sequence to send.

For example, to send the **Ctrl+Alt+Del** combination to the VM, click the **Send key** menu and select the **Ctrl+Alt+Del** menu entry.

**Troubleshooting**

- If clicking in the graphical console does not have any effect, expand the console to full screen. This is a known issue with the mouse cursor offset.

- If launching the Remote Viewer in the web console does not work or is not optimal, you can manually connect with any viewer application using the following protocols:
  - **Address** – The default address is **127.0.0.1**. You can modify the **vnc_listen** parameter in `/etc/libvirt/qemu.conf` to change it to the host’s IP address.
  - **VNC port** – 5901

**Additional resources**

- Viewing the virtual machine graphical console in the web console
- Viewing the virtual machine serial console in the web console

5.1.3. Viewing the virtual machine serial console in the web console
You can view the serial console of a selected virtual machine (VM) in the RHEL 9 web console. This is useful when the host machine or the VM is not configured with a graphical interface.

For more information about the serial console, see Opening a virtual machine serial console.

**Prerequisites**

- The web console VM plug-in is installed on your system.

**Procedure**

1. In the **Virtual Machines** pane, click the VM whose serial console you want to view.
   A new page opens with an **Overview** and a **Console** section for the VM.

2. Select **Serial console** in the console drop down menu.
   The graphical console appears in the web interface.

You can disconnect and reconnect the serial console from the VM.

- To disconnect the serial console from the VM, click **Disconnect**.
- To reconnect the serial console to the VM, click **Reconnect**.

**Additional resources**

- Viewing the virtual machine graphical console in the web console
- Viewing the graphical console in a remote viewer using the web console

**5.2. OPENING A VIRTUAL MACHINE GRAPHICAL CONSOLE USING VIRT VIEWER**

To connect to a graphical console of a KVM virtual machine (VM) and open it in the **Virt Viewer** desktop application, follow the procedure below.

**Prerequisites**

- Your system, as well as the VM you are connecting to, must support graphical displays.
• If the target VM is located on a remote host, connection and root access privileges to the host are needed.

• **Optional:** If the target VM is located on a remote host, set up your libvirt and SSH for more convenient access to remote hosts.

**Procedure**

• To connect to a local VM, use the following command and replace `guest-name` with the name of the VM you want to connect to:

  ```
  # virt-viewer guest-name
  ```

• To connect to a remote VM, use the `virt-viewer` command with the SSH protocol. For example, the following command connects as root to a VM called `guest-name`, located on remote system 10.0.0.1. The connection also requires root authentication for 10.0.0.1.

  ```
  # virt-viewer --direct --connect qemu+ssh://root@10.0.0.1/system guest-name
  ```

  `root@10.0.0.1`'s password:

**Verification**

If the connection works correctly, the VM display is shown in the **Virt Viewer** window.

![VM Display](image)

You can interact with the VM console using the mouse and keyboard in the same manner you interact with a real machine. The display in the VM console reflects the activities being performed on the VM.

**Troubleshooting**

• If clicking in the graphical console does not have any effect, expand the console to full screen. This is a known issue with the mouse cursor offset.

**Additional resources**
5.3. CONNECTING TO A VIRTUAL MACHINE USING SSH

To interact with the terminal of a virtual machine (VM) using the SSH connection protocol, follow the procedure below:

Prerequisites

- You have network connection and root access privileges to the target VM.
- If the target VM is located on a remote host, you also have connection and root access privileges to that host.
- Your VM network assigns IP addresses by **dnsmasq** generated by **libvirt**. This is the case for example in **libvirt** NAT networks.
- The **libvirt-nss** component is installed and enabled on the VM's host. If it is not, do the following:
  a. Install the **libvirt-nss** package:
     
     ```
     # dnf install libvirt-nss
     ```
  b. Edit the **/etc/nsswitch.conf** file and add **libvirt_guest** to the **hosts** line:
     
     ```
     [...]
     passwd: compat
     shadow: compat
     group: compat
     hosts: files libvirt_guest dns
     [...]
     ```

Procedure

1. When connecting to a remote VM, SSH into its physical host first. The following example demonstrates connecting to a host machine 10.0.0.1 using its root credentials:

   ```
   # ssh root@10.0.0.1
   root@10.0.0.1's password:
   Last login: Mon Sep 24 12:05:36 2021
   root~#
   ```

2. Use the VM's name and user access credentials to connect to it. For example, the following connects to the **testguest1** VM using its root credentials:

   ```
   # ssh root@testguest1
   root@testguest1's password:
   Last login: Wed Sep 12 12:05:36 2018
   root~]#
   ```
Troubleshooting

- If you do not know the VM’s name, you can list all VMs available on the host using the `virsh list -all` command:

```
# virsh list --all
Id    Name                           State
----------------------------------------------------
2     testguest1                    running
-     testguest2                    shut off
```

Additional resources

- Upstream libvirt documentation

## 5.4. OPENING AVIRTUAL MACHINE SERIAL CONSOLE

Using the `virsh console` command, it is possible to connect to the serial console of a virtual machine (VM).

This is useful when the VM:

- Does not provide VNC protocols, and thus does not offer video display for GUI tools.
- Does not have a network connection, and thus cannot be interacted with using SSH.

### Prerequisites

- The VM must have a serial console device configured, such as `console type='pty'`. To verify, do the following:

```
# *virsh dumpxml vm-name | grep console
<console type='pty' tty='/dev/pts/2'>
</console>
```

- The VM must have the serial console configured in its kernel command line. To verify this, the `cat /proc/cmdline` command output on the VM should include `console=ttyS0`. For example:

```
# cat /proc/cmdline
BOOT_IMAGE=/vmlinuz-3.10.0-948.el7.x86_64 root=/dev/mapper/rhel-root ro console=tty0 console=ttyS0,9600n8 rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb
```

If the serial console is not set up properly on a VM, using `virsh console` to connect to the VM connects you to an unresponsive guest console. However, you can still exit the unresponsive console by using the `Ctrl+` shortcut.

To set up serial console on the VM, do the following:

a. On the VM, edit the `/etc/default/grub` file and add `console=ttyS0` to the line that starts with `GRUB_CMDLINE_LINUX`.

b. Clear the kernel options that may prevent your changes from taking effect.

```
# grub2-editenv - unset kernelopts
```
c. Reload the Grub configuration:

```
# grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-3.10.0-948.el7.x86_64
Found initrd image: /boot/initramfs-3.10.0-948.el7.x86_64.img
[...]
done
```

d. Reboot the VM.

Procedure

1. On your host system, use the `virsh console` command. The following example connects to the `guest1` VM, if the libvirt driver supports safe console handling:

```
# virsh console guest1 --safe
Connected to domain 'guest1'
Escape character is ^]  
Subscription-name
Kernel 3.10.0-948.el7.x86_64 on an x86_64
localhost login:
```

2. You can interact with the virsh console in the same way as with a standard command-line interface.

Additional resources

- The `virsh` man page

5.5. SETTING UP EASY ACCESS TO REMOTE VIRTUALIZATION HOSTS

When managing VMs on a remote host system using libvirt utilities, it is recommended to use the `-c qemu+ssh://root@hostname/system` syntax. For example, to use the `virsh list` command as root on the 10.0.0.1 host:

```
# virsh -c qemu+ssh://root@10.0.0.1/system list
root@10.0.0.1’s password:
Id   Name              State
---------------------------------
1    remote-guest      running
```

However, for convenience, you can remove the need to specify the connection details in full by modifying your SSH and libvirt configuration. For example, you will be able to do:

```
# virsh -c remote-host list
root@10.0.0.1’s password:
```
To enable this improvement, follow the instructions below.

Procedure

1. Edit or create the ~/.ssh/config file, and add the following to it, where host-alias is a shortened name associated with a specific remote host, and hosturl is the URL address of the host.

   ```
   Host host-alias
   User root
   Hostname hosturl
   ```

   For example, the following sets up the tyrannosaurus alias for root@10.0.0.1:

   ```
   Host tyrannosaurus
   User root
   Hostname 10.0.0.1
   ```

2. Edit or create the /etc/libvirt/libvirt.conf file, and add the following, where qemu-host-alias is a host alias that QEMU and libvirt utilities will associate with the intended host:

   ```
   uri_aliases = [
     "qemu-host-alias=qemu+ssh://host-alias/system",
   ]
   ```

   For example, the following uses the tyrannosaurus alias configured in the previous step to set up the t-rex alias, which stands for qemu+ssh://10.0.0.1/system:

   ```
   uri_aliases = [
     "t-rex=qemu+ssh://tyrannosaurus/system",
   ]
   ```

Verification

1. Confirm that you can manage remote VMs by using libvirt-based utilities on the local system with an added -c qemu-host-alias parameter. This automatically performs the commands over SSH on the remote host.

   For example, verify that the following lists VMs on the 10.0.0.1 remote host, the connection to which was set up as t-rex in the previous steps:

   ```
   $ virsh -c t-rex list
   root@10.0.0.1’s password:
   ```

   ```
   Id Name State
   ------------------
   1 velociraptor running
   ```
NOTE

In addition to virsh, the -c (or --connect) option and the remote host access configuration described above can be used by the following utilities:

- virt-install
- virt-viewer

Next steps

- If you want to use libvirt utilities exclusively on a single remote host, you can also set a specific connection as the default target for libvirt-based utilities. To do so, edit the /etc/libvirt/libvirt.conf file and set the value of the uri_default parameter to qemu-host-alias. For example, the following uses the t-rex host alias set up in the previous steps as a default libvirt target.

```
# These can be used in cases when no URI is supplied by the application
# (@uri_default also prevents probing of the hypervisor driver).
uri_default = "t-rex"
```

As a result, all libvirt-based commands will automatically be performed on the specified remote host.

```
$ virsh list
root@10.0.0.1's password:

Id   Name              State
---------------------------------
1    velociraptor      running
```

However, this is not recommended if you also want to manage VMs on your local host or on different remote hosts.

- When connecting to a remote host, you can avoid having to provide the root password to the remote system. To do so, use one or more of the following methods:
  - Set up key-based SSH access to the remote host
  - Use SSH connection multiplexing to connect to the remote system
  - Kerberos authentication in Identity Management
- The -c (or --connect) option can be used to run the virt-install, virt-viewer, and virsh commands on a remote host.
CHAPTER 6. SHUTTING DOWN VIRTUAL MACHINES

To shut down a running virtual machine hosted on RHEL 9, use the command line interface or the web console GUI.

6.1. SHUTTING DOWN A VIRTUAL MACHINE USING THE COMMAND-LINE INTERFACE

To shut down a responsive virtual machine (VM), do one of the following:

- Use a shutdown command appropriate to the guest OS while connected to the guest.
- Use the `virsh shutdown` command on the host:
  - If the VM is on a local host:
    ```
    # virsh shutdown demo-guest1
    Domain 'demo-guest1' is being shutdown
    ```
  - If the VM is on a remote host, in this example 10.0.0.1:
    ```
    # virsh -c qemu+ssh://root@10.0.0.1/system shutdown demo-guest1
    root@10.0.0.1's password:
    Domain 'demo-guest1' is being shutdown
    ```

To force a VM to shut down, for example if it has become unresponsive, use the `virsh destroy` command on the host:

```
# virsh destroy demo-guest1
Domain 'demo-guest1' destroyed
```

NOTE

The `virsh destroy` command does not actually delete or remove the VM configuration or disk images. It only terminates the running VM instance of the VM, similarly to pulling the power cord from a physical machine. As such, in rare cases, `virsh destroy` may cause corruption of the VM’s file system, so using this command is only recommended if all other shutdown methods have failed.

6.2. SHUTTING DOWN AND RESTARTING VIRTUAL MACHINES USING THE WEB CONSOLE

Using the RHEL 9 web console, you can shut down or restart running virtual machines. You can also send a non-maskable interrupt to an unresponsive virtual machine.

6.2.1. Shutting down virtual machines in the web console

If a virtual machine (VM) is in the running state, you can shut it down using the RHEL 9 web console.

Prerequisites
The web console VM plug-in is installed on your system.

Procedure

1. In the **Virtual Machines** interface, find the row of the VM you want to shut down.
2. On the right side of the row, click **Shut Down**.
   The VM shuts down.

Troubleshooting

- If the VM does not shut down, click the Menu button ⋱ next to the **Shut Down** button and select **Force Shut Down**.
- To shut down an unresponsive VM, you can also send a non-maskable interrupt.

Additional resources

- Starting virtual machines using the web console
- Restarting virtual machines using the web console

### 6.2.2. Restarting virtual machines using the web console

If a virtual machine (VM) is in the **running** state, you can restart it using the RHEL 9 web console.

Prerequisites

- The web console VM plug-in is installed on your system.

Procedure

1. In the **Virtual Machines** interface, find the row of the VM you want to restart.
2. On the right side of the row, click the Menu button ⋱.
   A drop-down menu of actions appears.
3. In the drop-down menu, click **Reboot**.
   The VM shuts down and restarts.

Troubleshooting

- If the VM does not restart, click the Menu button ⋱ next to the **Restart** button and select **Force Restart**.
- To shut down an unresponsive VM, you can also send a non-maskable interrupt.

Additional resources

- Starting virtual machines using the web console
- Shutting down virtual machines in the web console

### 6.2.3. Sending non-maskable interrupts to VMs using the web console
Sending a non-maskable interrupt (NMI) may cause an unresponsive running virtual machine (VM) to respond or shut down. For example, you can send the \texttt{Ctrl+Alt+Del} NMI to a VM that is not responding to standard input.

**Prerequisites**

- The web console VM plug-in is installed on your system.

**Procedure**

1. In the \textbf{Virtual Machines} interface, find the row of the VM to which you want to send an NMI.
2. On the right side of the row, click the Menu button \(\cdots\). A drop-down menu of actions appears.
3. In the drop-down menu, click \textbf{Send Non-Maskable Interrupt} An NMI is sent to the VM.

**Additional resources**

- Starting virtual machines using the web console
- Restarting virtual machines using the web console
- Shutting down virtual machines in the web console
CHAPTER 7. DELETING VIRTUAL MACHINES

To delete virtual machines in RHEL 9, use the command line interface or the web console GUI.

7.1. DELETING VIRTUAL MACHINES USING THE COMMAND LINE INTERFACE

To delete a virtual machine (VM), you can remove its XML configuration and associated storage files from the host using the command line. Follow the procedure below:

Prerequisites

- Back up important data from the VM.
- Shut down the VM.
- Make sure no other VMs use the same associated storage.

Procedure

- Use the `virsh undefine` utility.
  For example, the following command removes the `guest1` VM, its associated storage volumes, and non-volatile RAM, if any.

  ```
  # virsh undefine guest1 --remove-all-storage --nvram
  Domain 'guest1' has been undefined
  Volume 'vda' (/home/images/guest1.qcow2) removed.
  ```

Additional resources

- The `virsh undefine --help` command
- The `virsh` man page

7.2. DELETING VIRTUAL MACHINES USING THE WEB CONSOLE

To delete a virtual machine (VM) and its associated storage files from the host to which the RHEL 9 web console is connected with, follow the procedure below:

Prerequisites

- The web console VM plug-in is installed on your system.
- Back up important data from the VM.
- Make sure no other VM uses the same associated storage.
- Optional: Shut down the VM.

Procedure

1. In the Virtual Machines interface, click the Menu button of the VM that you want to delete.
   A drop down menu appears with controls for various VM operations.
2. Click **Delete**.
   A confirmation dialog appears.

![Confirm deletion of Grid_v2](image_url)

3. **Optional:** To delete all or some of the storage files associated with the VM, select the checkboxes next to the storage files you want to delete.

4. Click **Delete**.
   The VM and any selected storage files are deleted.
CHAPTER 8. MANAGING VIRTUAL MACHINES IN THE WEB CONSOLE

To manage virtual machines in a graphical interface on a RHEL 9 host, you can use the Virtual Machines pane in the RHEL 9 web console.

8.1. OVERVIEW OF VIRTUAL MACHINE MANAGEMENT USING THE WEB CONSOLE

The RHEL 9 web console is a web-based interface for system administration. As one of its features, the web console provides a graphical view of virtual machines (VMs) on the host system, and makes it possible to create, access, and configure these VMs.

Note that to use the web console to manage your VMs on RHEL 9, you must first install a web console plug-in for virtualization.

Next steps

- For instructions on enabling VMs management in your web console, see Setting up the web console to manage virtual machines.
- For a comprehensive list of VM management actions that the web console provides, see Virtual machine management features available in the web console.

8.2. SETTING UP THE WEB CONSOLE TO MANAGE VIRTUAL MACHINES

Before using the RHEL 9 web console to manage virtual machines (VMs), you must install the web console virtual machine plug-in on the host.

Prerequisites

- Ensure that the web console is installed and enabled on your machine.

```bash
# systemctl status cockpit.socket
cockpit.socket - Cockpit Web Service Socket
```
If this command returns **Unit cockpit.socket could not be found**, follow the Installing the web console document to enable the web console.

**Procedure**

- Install the **cockpit-machines** plug-in.

  ```
  # dnf install cockpit-machines
  ```

**Verification**

1. Access the web console, for example by entering the [https://localhost:9090](https://localhost:9090) address in your browser.

2. Log in.

3. If the installation was successful, **Virtual Machines** appears in the web console side menu.

**Additional resources**

- Managing systems using the RHEL 9 web console

### 8.3. RENAMING VIRTUAL MACHINES USING THE WEB CONSOLE

After create a virtual machine (VM), you might wish to rename the VM to avoid conflicts or assign a new unique name based on your use case. You can use the RHEL web console to rename the VM.

**Prerequisites**

- The web console VM plug-in is installed on your system.

- Ensure that the VM is shut down.

**Procedure**

1. In the **Virtual Machines** interface, click the Menu button of the VM that you want to rename.
A drop down menu appears with controls for various VM operations.

2. Click **Rename**.
   The Rename a VM dialog appears.

   ![Rename VM Grid_v2](image)

   **Rename VM Grid_v2**

   **New name**

   ![Grid_v2](image)

   **Rename**  **Cancel**

3. In the **New name** field, enter a name for the VM.

4. Click **Rename**.

**Verification**

- The new VM name should appear in the **Virtual Machines** interface.

### 8.4. VIRTUAL MACHINE MANAGEMENT FEATURES AVAILABLE IN THE WEB CONSOLE

Using the RHEL 9 web console, you can perform the following actions to manage the virtual machines (VMs) on your system.

**Table 8.1. VM management tasks that you can perform in the RHEL 9 web console**

<table>
<thead>
<tr>
<th>Task</th>
<th>For details, see</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a VM and install it with a guest operating system</td>
<td>Creating virtual machines and installing guest operating systems using the web console</td>
</tr>
<tr>
<td>Delete a VM</td>
<td>Deleting virtual machines using the web console</td>
</tr>
<tr>
<td>Task</td>
<td>For details, see</td>
</tr>
<tr>
<td>---------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td>Start, shut down, and restart the VM</td>
<td>Starting virtual machines using the web console</td>
</tr>
<tr>
<td></td>
<td>Shutting down and restarting virtual machines using the web console</td>
</tr>
<tr>
<td>Connect to and interact with a VM using a variety of consoles</td>
<td>Interacting with virtual machines using the web console</td>
</tr>
<tr>
<td>View a variety of information about the VM</td>
<td>Viewing virtual machine information using the web console</td>
</tr>
<tr>
<td>Adjust the host memory allocated to a VM</td>
<td>Adding and removing virtual machine memory using the web console</td>
</tr>
<tr>
<td>Manage network connections for the VM</td>
<td>Using the web console for managing virtual machine network interfaces</td>
</tr>
<tr>
<td>Manage the VM storage available on the host and attach virtual disks to the VM</td>
<td>Managing storage for virtual machines using the web console</td>
</tr>
<tr>
<td>Configure the virtual CPU settings of the VM</td>
<td>Managing virtual CPUs using the web console</td>
</tr>
<tr>
<td>Live migrate a VM</td>
<td>Live migrating a virtual machine using the web console</td>
</tr>
<tr>
<td>Manage host devices</td>
<td>Managing host devices using the web console</td>
</tr>
</tbody>
</table>
CHAPTER 9. VIEWING INFORMATION ABOUT VIRTUAL MACHINES

When you need to adjust or troubleshoot any aspect of your virtualization deployment on RHEL 9, the first step you need to perform usually is to view information about the current state and configuration of your virtual machines. To do so, you can use the command-line interface or the web console. You can also view the information in the VM's XML configuration.

9.1. VIEWING VIRTUAL MACHINE INFORMATION USING THE COMMAND-LINE INTERFACE

To retrieve information about virtual machines (VMs) on your host and their configurations, use one or more of the following commands.

Procedure

- To obtain a list of VMs on your host:

```bash
# virsh list --all

Id   Name              State
----------------------------------
1    testguest1             running
-    testguest2             shut off
-    testguest3             shut off
-    testguest4             shut off
```

- To obtain basic information about a specific VM:

```bash
# virsh dominfo testguest1

Id:             1
Name:           testguest1
UUID:           a973666f-2f6e-415a-8949-75a7a98569e1
OS Type:        hvm
State:          running
CPU(s):         2
CPU time:       188.3s
Max memory:     4194304 KiB
Used memory:    4194304 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c486,c538 (enforcing)
```

- To obtain the complete XML configuration of a specific VM:

```bash
# virsh dumpxml testguest2

<domain type='kvm' id='1'>
  <name>testguest2</name>
</domain>
```
For information about a VM’s disks and other block devices:

```
# virsh domblklist testguest3
```

<table>
<thead>
<tr>
<th>Target</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>vda</td>
<td>/var/lib/libvirt/images/testguest3.qcow2</td>
</tr>
<tr>
<td>sda</td>
<td>-</td>
</tr>
<tr>
<td>sdb</td>
<td>/home/username/Downloads/virt-p2v-1.36.10-1.el7.iso</td>
</tr>
</tbody>
</table>

To obtain information about a VM’s file systems and their mountpoints:

```
# virsh domfsinfo testguest3
```

<table>
<thead>
<tr>
<th>Mountpoint</th>
<th>Name</th>
<th>Type</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>dm-0</td>
<td>xfs</td>
<td></td>
</tr>
<tr>
<td>/boot</td>
<td>vda1</td>
<td>xfs</td>
<td></td>
</tr>
</tbody>
</table>

To obtain more details about the vCPUs of a specific VM:

```
# virsh vcpuinfo testguest4
```

- VCPU: 0  
  CPU: 3  
  State: running  
  CPU time: 103.1s  
  CPU Affinity: yyyy

- VCPU: 1  
  CPU: 0  
  State: running  
  CPU time: 88.6s  
  CPU Affinity: yyyy

To configure and optimize the vCPUs in your VM, see Optimizing virtual machine CPU performance.

To list all virtual network interfaces on your host:

```
# virsh net-list --all
```

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
<th>Autostart</th>
<th>Persistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>active</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>labnet</td>
<td>active</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

For information about a specific interface:

```
# virsh net-info default
```

- Name: default
- UUID: c699f9f6-9202-4ca8-91d0-6b8cb9024116
- Active: yes
Persistent: yes
Autostart: yes
Bridge: virbr0

For details about network interfaces, VM networks, and instructions for configuring them, see Configuring virtual machine network connections.

9.2. VIEWING VIRTUAL MACHINE INFORMATION USING THE WEB CONSOLE

Using the RHEL 9 web console, you can view information about all VMs and storage pools the web console session can access.

You can view information about a selected VM to which the web console session is connected. This includes information about its disks, virtual network interface and resource usage.

9.2.1. Viewing a virtualization overview in the web console

Using the web console, you can access a virtualization overview that contains summarized information about available virtual machines (VMs), storage pools, and networks.

Prerequisites
- The web console VM plug-in is installed on your system.

Procedure
- Click Virtual Machines in the web console’s side menu. A dialog box appears with information about the available storage pools, available networks, and the VMs to which the web console is connected.

The information includes the following:

- **Storage Pools** - The number of storage pools, active or inactive, that can be accessed by the web console and their state.

- **Networks** - The number of networks, active or inactive, that can be accessed by the web console and their state.
- **Name** - The name of the VM.
- **Connection** - The type of libvirt connection, system or session.
- **State** - The state of the VM.

**Additional resources**

- Viewing virtual machine information using the web console

**9.2.2. Viewing storage pool information using the web console**

Using the web console, you can view detailed information about storage pools available on your system. Storage pools can be used to create disk images for your virtual machines.

**Prerequisites**

- The web console VM plug-in is installed on your system.

**Procedure**

1. Click **Storage Pools** at the top of the **Virtual Machines** interface. The Storage Pools window appears, showing a list of configured storage pools.

   ![Storage Pools Window](image)

   The information includes the following:

   - **Name** - The name of the storage pool.
   - **Size** - The current allocation and the total capacity of the storage pool.
   - **Connection** - The connection used to access the storage pool.
   - **State** - The state of the storage pool.

2. Click the arrow next to the storage pool whose information you want to see. The row expands to reveal the Overview pane with detailed information about the selected storage pool.
The information includes:

- **Target path** - The source for the types of storage pools backed by directories, such as `dir` or `netfs`.
- **Persistent** - Indicates whether or not the storage pool has a persistent configuration.
- **Autostart** - Indicates whether or not the storage pool starts automatically when the system boots up.
- **Type** - The type of the storage pool.

3. To view a list of storage volumes associated with the storage pool, click **Storage Volumes**. The Storage Volumes pane appears, showing a list of configured storage volumes.

The information includes:

- **Name** - The name of the storage volume.
- **Used by** - The VM that is currently using the storage volume.
- **Size** - The size of the volume.

Additional resources

- Viewing virtual machine information using the web console

9.2.3. Viewing basic virtual machine information in the web console

Using the web console, you can view basic information, such as assigned resources or hypervisor details, about a selected virtual machine (VM).

**Prerequisites**

- The web console VM plug-in is installed on your system.

**Procedure**

1. Click **Virtual Machines** in the web console side menu.
2. Click the VM whose information you want to see.
A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM's graphical interface.

The Overview section includes the following general VM details:

- **State** - The VM state, Running or Shut off.
- **Memory** - The amount of memory assigned to the VM.
- **vCPUs** - The number of virtual CPUs configured for the VM.
- **CPU Type** - The architecture of the virtual CPUs configured for the VM.
- **Boot Order** - The boot order configured for the VM.
- **Autostart** - Whether or not autostart is enabled for the VM.

The information also includes the following hypervisor details:

- **Emulated Machine** - The machine type emulated by the VM.
- **Firmware** - The firmware of the VM.

Additional resources

- Viewing virtual machine information using the web console
- Managing virtual CPUs using the web console

9.2.4. Viewing virtual machine resource usage in the web console

Using the web console, you can view memory and virtual CPU usage of a selected virtual machine (VM).

Prerequisites

- The web console VM plug-in is installed on your system.

Procedure

1. In the Virtual Machines interface, click the VM whose information you want to see.
A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM's graphical interface.

2. Scroll to **Usage**.
The Usage section displays information about the memory and virtual CPU usage of the VM.

![Usage](image)

### Additional resources
- Viewing virtual machine information using the web console

#### 9.2.5. Viewing virtual machine disk information in the web console

Using the web console, you can view detailed information about disks assigned to a selected virtual machine (VM).

**Prerequisites**
- The web console VM plug-in is installed on your system.

**Procedure**

1. Click the VM whose information you want to see.
   A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM's graphical interface.

2. Scroll to **Disks**.
The Disks section displays information about the disks assigned to the VM as well as options to Add, Remove, or Edit disks.

<table>
<thead>
<tr>
<th>Disks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
</tr>
<tr>
<td>--------</td>
</tr>
</tbody>
</table>
| disk   | 8.9 GB | 10 GB    | virtio | Writeable | File | /var/lib/libvirt/images/Gril_v2.png
| disk   | 0 GB | 15 GB    | virtio | Writeable | Pool | default
|        |      |          |       |         | Volume | v2 |

The information includes the following:
- **Device** - The device type of the disk.
- **Used** - The amount of disk currently allocated.
- **Capacity** - The maximum size of the storage volume.
- **Bus** - The type of disk device that is emulated.
- **Access** - Whether the disk is **Writeable** or **Read-only**. For **raw** disks, you can also set the access to **Writeable and shared**.
- **Source** - The disk device or file.

**Additional resources**

- **Viewing virtual machine information using the web console**

### 9.2.6. Viewing and editing virtual network interface information in the web console

Using the RHEL 9 web console, you can view and modify the virtual network interfaces on a selected virtual machine (VM):

**Prerequisites**

- The web console VM plug-in is installed on your system.

**Procedure**

1. In the **Virtual Machines** interface, click the VM whose information you want to see. A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM’s graphical interface.

2. Scroll to **Network Interfaces**. The Networks Interfaces section displays information about the virtual network interface configured for the VM as well as options to **Add**, **Delete**, **Edit**, or **Unplug** network interfaces.

<table>
<thead>
<tr>
<th>Network interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
</tr>
<tr>
<td>network</td>
</tr>
</tbody>
</table>

- The information includes the following:
  - **Type** - The type of network interface for the VM. The types include virtual network, bridge to LAN, and direct attachment.

**NOTE**

Generic Ethernet connection is not supported in RHEL 9 and later.

- **Model type** - The model of the virtual network interface.
- **MAC Address** - The MAC address of the virtual network interface.
- **IP Address** - The IP address of the virtual network interface.
- **Source** - The source of the network interface. This is dependent on the network type.
- **State** - The state of the virtual network interface.

3. To edit the virtual network interface settings, Click **Edit**. The Virtual Network Interface Settings dialog opens.

```
52:54:00:b4:2a:62 virtual network interface settings
```

<table>
<thead>
<tr>
<th>Interface type</th>
<th>Virtual network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>default</td>
</tr>
<tr>
<td>Model</td>
<td>(Linux, perf)</td>
</tr>
<tr>
<td>MAC address</td>
<td>52:64:00:b4:2a:63</td>
</tr>
</tbody>
</table>

4. Change the interface type, source, model, or MAC address.

5. Click **Save**. The network interface is modified.

**NOTE**

Changes to the virtual network interface settings take effect only after restarting the VM.

Additionally, MAC address can only be modified when the VM is shut off.

**Additional resources**

- Viewing virtual machine information using the web console

### 9.3. SAMPLE VIRTUAL MACHINE XML CONFIGURATION

The XML configuration of a VM, also referred to as a domain XML, determines the VM’s settings and components. The following table shows sections of a sample XML configuration of a virtual machine (VM) and explains the contents.

To obtain the XML configuration of a VM, you can use the **virsh dumpxml** command followed by the VM’s name.

```
# virsh dumpxml testguest1
```

**Table 9.1. Sample XML configuration**
<table>
<thead>
<tr>
<th>Domain XML Section</th>
<th>Description</th>
</tr>
</thead>
</table>
| `<domain type='kvm'>`  
  `<name>Testguest1</name>`  
  `<uuid>ec6fbaa1-3eb4-49da-bf61-bb02fbec4967</uuid>`  
  `<memory unit='KiB'>1048576</memory>`  
  `<currentMemory unit='KiB'>1048576</currentMemory>` | This is a KVM virtual machine called Testguest1, with 1024 MiB allocated RAM. |
| `<vcpu placement='static'>1</vcpu>` | The VM is allocated with a single virtual CPU (vCPU).  
For information about configuring vCPUs, see Optimizing virtual machine CPU performance. |
| `<os>`  
  `<type arch='x86_64' machine='pc-q35-rhel9.0.0'>hvm</type>`  
  `<boot dev='hd'/>` | The machine architecture is set to the AMD64 and Intel 64 architecture, and uses the Intel Q35 machine type to determine feature compatibility. The OS is set to be booted from the hard drive.  
For information about creating a VM with an installed OS, see Creating virtual machines and installing guest operating systems using the web console. |
| `<features>`  
  `<acpi/>`  
  `<apic/>` | The acpi and apic hypervisor features are disabled. |
<p>| <code>&lt;cpu mode='host-model' check='partial'/&gt;</code> | The host CPU definitions from capabilities XML (obtainable with virsh domcapabilities) are automatically copied into the VM’s XML configuration. Therefore, when the VM is booted, libvirt picks a CPU model that is similar to the host CPU, and then adds extra features to approximate the host model as closely as possible. |</p>
<table>
<thead>
<tr>
<th>Domain XML Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`&lt;clock offset='utc'&gt;</td>
<td>The VM’s virtual hardware clock uses the UTC time zone. In addition, three different timers are set up for synchronization with the QEMU hypervisor.</td>
</tr>
<tr>
<td><code>&lt;timer name='rtc' tickpolicy='catchup'/&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;timer name='pit' tickpolicy='delay'/&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;timer name='hpet' present='no'/&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;clock&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;on_poweroff&gt;destroy&lt;/on_poweroff&gt;</code></td>
<td>When the VM powers off, or its OS terminates unexpectedly, libvirt terminates the VM and releases all its allocated resources. When the VM is rebooted, libvirt restarts it with the same configuration.</td>
</tr>
<tr>
<td><code>&lt;on_reboot&gt;restart&lt;/on_reboot&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;on_crash&gt;destroy&lt;/on_crash&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;pm&gt;</code></td>
<td>The S3 and S4 ACPI sleep states are disabled for this VM.</td>
</tr>
<tr>
<td><code>&lt;suspend-to-mem enabled='no'/&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;suspend-to-disk enabled='no'/&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;pm&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;devices&gt;</code></td>
<td>The VM uses the /usr/libexec/qemu-kvm binary file for emulation and it has two disk devices attached.</td>
</tr>
<tr>
<td><code>&lt;emulator&gt;/usr/libexec/qemu-kvm&lt;/emulator&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;disk type='file' device='disk'&gt;</code></td>
<td>The first disk is a virtualized hard-drive based on the /var/lib/libvirt/images/Testguest.qcow2 stored on the host, and its logical device name is set to vda. In windows guests, it is recommended to use sata bus instead of virtio.</td>
</tr>
<tr>
<td><code>&lt;driver name='qemu' type='qcow2'/&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;source file='/var/lib/libvirt/images/Testguest.qcow2'/&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;target dev='vda' bus='virtio'/&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;disk&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;disk type='file' device='cdrom'&gt;</code></td>
<td>The second disk is a virtualized CD-ROM and its logical device name is set to sdb.</td>
</tr>
<tr>
<td><code>&lt;driver name='qemu' type='raw'/&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;source file='/var/lib/libvirt/images/Testguest.qcow2'/&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;target dev='sdb' bus='sata'/&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;readonly/&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;disk&gt;</code></td>
<td></td>
</tr>
</tbody>
</table>
The VM uses a single controller for attaching USB devices, and a root controller for PCI-Express (PCIe) devices. In addition, a **virtio-serial** controller is available, which enables the VM to interact with the host in a variety of ways, such as the serial console.

For more information about virtual devices, see [Types of virtual devices](#).

A network interface is set up in the VM that uses the **default** virtual network and the **virtio** network device model. In Windows guests, it is recommended to use **e1000e** model instead of **virtio**.

For information about configuring the network interface, see [Optimizing virtual machine network performance](#).
<table>
<thead>
<tr>
<th>Domain XML Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;serial type='pty'&gt; &lt;target type='isa-serial' port='0'&gt; &lt;model name='isa-serial'/&gt;</td>
<td>A <em>pty</em> serial console is set up on the VM, which enables rudimentary VM communication with the host. The console uses the <em>UNIX</em> channel on port 1. This is set up automatically and changing these settings is not recommended. For more information about interacting with VMs, see <a href="#">Interacting with virtual machines using the web console</a>.</td>
</tr>
<tr>
<td>&lt;/target&gt; &lt;serial&gt; &lt;console type='pty'&gt; &lt;target type='serial' port='0'/&gt; &lt;/console&gt; &lt;channel type='unix'&gt; &lt;target type='virtio' name='org.qemu.guest_agent.0'/&gt; &lt;address type='virtio-serial' controller='0' bus='0' port='1'/&gt; &lt;/channel&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;input type='tablet' bus='usb'&gt; &lt;address type='usb' bus='0' port='1'/&gt; &lt;/input&gt;</td>
<td>The VM uses a virtual <em>usb</em> port, which is set up to receive tablet input, and a virtual <em>ps2</em> port set up to receive mouse and keyboard input. This is set up automatically and changing these settings is not recommended.</td>
</tr>
<tr>
<td>&lt;input type='mouse' bus='ps2'/&gt; &lt;input type='keyboard' bus='ps2'/&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;graphics type='vnc' port='-1' autoport='yes' listen='127.0.0.1'&gt;</td>
<td>The VM uses the <em>vnc</em> protocol for rendering its graphical output.</td>
</tr>
<tr>
<td>&lt;/graphics&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;redirdev bus='usb' type='tcp'&gt; &lt;source mode='connect' host='localhost' service='4000'/&gt;</td>
<td>The VM uses <em>tcp</em> re-director for attaching USB devices remotely, and memory ballooning is turned on. This is set up automatically and changing these settings is not recommended.</td>
</tr>
<tr>
<td>&lt;protocol type='raw'/&gt; &lt;/redirdev&gt; &lt;memballoon model='virtio'&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/memballoon&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/devices&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;/domain&gt;</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 10. SAVING AND RESTORING VIRTUAL MACHINES

To free up system resources, you can shut down a virtual machine (VM) running on that system. However, when you require the VM again, you must boot up the guest operating system (OS) and restart the applications, which may take a considerable amount of time. To reduce this downtime and enable the VM workload to start running sooner, you can use the save and restore feature to avoid the OS shutdown and boot sequence entirely.

This section provides information about saving VMs, as well as about restoring them to the same state without a full VM boot-up.

10.1. HOW SAVING AND RESTORING VIRTUAL MACHINES WORKS

Saving a virtual machine (VM) saves its memory and device state to the host’s disk, and immediately stops the VM process. You can save a VM that is either in a running or paused state, and upon restoring, the VM will return to that state.

This process frees up RAM and CPU resources on the host system in exchange for disk space, which may improve the host system performance. When the VM is restored, because the guest OS does not need to be booted, the long boot-up period is avoided as well.

To save a VM, you can use the command-line interface (CLI). For instructions, see Saving virtual machines using the command line interface.

To restore a VM you can use the CLI or the web console GUI.

10.2. SAVING A VIRTUAL MACHINE USING THE COMMAND LINE INTERFACE

You can save a virtual machine (VM) and its current state to the host’s disk. This is useful, for example, when you need to use the host’s resources for some other purpose. The saved VM can then be quickly restored to its previous running state.

To save a VM using the command line, follow the procedure below.

Prerequisites

- Make sure you have sufficient disk space to save the VM and its configuration. Note that the space occupied by the VM depends on the amount of RAM allocated to that VM.
- Make sure the VM is persistent.
- Optional: Back up important data from the VM if required.

Procedure

- Use the `virsh managedsave` utility.
  For example, the following command stops the `demo-guest1` VM and saves its configuration.

  ```bash
  # virsh managedsave demo-guest1
  Domain 'demo-guest1' saved by libvirt
  ```

  The saved VM file is located by default in the `/var/lib/libvirt/qemu/save` directory as `demo-guest1.save`.
The next time the VM is started, it will automatically restore the saved state from the above file.

Verification

- You can make sure that the VM is in a saved state or shut off using the virsh list utility.
  To list the VMs that have managed save enabled, use the following command. The VMs listed as saved have their managed save enabled.

```bash
# virsh list --managed-save --all
Id    Name                           State
----------------------------------------------------
-     demo-guest1                    saved
-     demo-guest2                    shut off
```

To list the VMs that have a managed save image:

```bash
# virsh list --with-managed-save --all
Id    Name                           State
----------------------------------------------------
-     demo-guest1                    shut off
```

Note that to list the saved VMs that are in a shut off state, you must use the --all or --inactive options with the command.

Troubleshooting

- If the saved VM file becomes corrupted or unreadable, restoring the VM will initiate a standard VM boot instead.

Additional resources

- The virsh managedsave --help command
- Restoring a saved VM using the command-line interface
- Restoring a saved VM using the web console

10.3. STARTING A VIRTUAL MACHINE USING THE COMMAND-LINE INTERFACE

You can use the command line interface (CLI) to start a shut-down virtual machine (VM) or restore a saved VM. Using the CLI, you can start both local and remote VMs.

Prerequisites

- An inactive VM that is already defined.
- The name of the VM.
- For remote VMs:
  - The IP address of the host where the VM is located.
  - Root access privileges to the host.
**Procedure**

- For a local VM, use the `virsh start` utility. For example, the following command starts the `demo-guest1` VM.

  ```bash
  # virsh start demo-guest1
  Domain 'demo-guest1' started
  ```

- For a VM located on a remote host, use the `virsh start` utility along with the QEMU+SSH connection to the host. For example, the following command starts the `demo-guest1` VM on the 192.168.123.123 host.

  ```bash
  # virsh -c qemu+ssh://root@192.168.123.123/system start demo-guest1
  root@192.168.123.123's password:
  Domain 'demo-guest1' started
  ```

**Additional resources**

- The `virsh start --help` command
- Setting up easy access to remote virtualization hosts
- Starting virtual machines automatically when the host starts

### 10.4. STARTING VIRTUAL MACHINES USING THE WEB CONSOLE

If a virtual machine (VM) is in the `shut off` state, you can start it using the RHEL 9 web console. You can also configure the VM to be started automatically when the host starts.

**Prerequisites**

- The web console VM plug-in is installed on your system.
- An inactive VM that is already defined.
- The name of the VM.

**Procedure**

1. In the **Virtual Machines** interface, click the VM you want to start. A new page opens with detailed information about the selected VM and controls for shutting down and deleting the VM.

2. Click **Run**. The VM starts, and you can connect to its console or graphical output.

3. **Optional**: To configure the VM to start automatically when the host starts, click the **Autostart** checkbox. If you use network interfaces that are not managed by libvirt, you must also make additional changes to the systemd configuration. Otherwise, the affected VMs might fail to start, see **starting virtual machines automatically when the host starts**.
Additional resources

- Shutting down virtual machines in the web console
- Restarting virtual machines using the web console
CHAPTER 11. CLONING VIRTUAL MACHINES

To quickly create a new virtual machine (VM) with a specific set of properties, you can clone an existing VM.

Cloning creates a new VM that uses its own disk image for storage, but most of the clone’s configuration and stored data is identical to the source VM. This makes it possible to prepare a number of VMs optimized for a certain task without the need to optimize each VM individually.

11.1. HOW CLONING VIRTUAL MACHINES WORKS

Cloning a virtual machine (VM) copies the XML configuration of the source VM and its disk images, and makes adjustments to the configurations to ensure the uniqueness of the new VM. This includes changing the name of the VM and ensuring it uses the disk image clones. Nevertheless, the data stored on the clone’s virtual disks is identical to the source VM.

This process is faster than creating a new VM and installing it with a guest operating system, and can be used to rapidly generate VMs with a specific configuration and content.

If you are planning to create multiple clones of a VM, first create a VM template that does not contain:

- Unique settings, such as persistent network MAC configuration, which can prevent the clones from working correctly.
- Sensitive data, such as SSH keys and password files.

For instructions, see Creating virtual machines templates.

Additional resources

- Cloning a virtual machine using the command-line interface
- Cloning a virtual machine using the web console

11.2. CREATING VIRTUAL MACHINE TEMPLATES

To create multiple virtual machine (VM) clones that work correctly, you can remove information and configurations that are unique to a source VM, such as SSH keys or persistent network MAC configuration. This creates a VM template, which you can use to easily and safely create VM clones.

You can create VM templates using the virt-sysprep utility or you can create them manually based on your requirements.

11.2.1. Creating a virtual machine template using virt-sysrep

To create a template from an existing virtual machine (VM), you can use the virt-sysprep utility to quickly unconfigure a guest VM to prepare it for cloning. The virt-sysprep utility creates a template by automatically removing certain configurations from a VM that should not be copied to a clone.

Prerequisites

- The virt-sysprep utility is installed on your host:

  # dnf install /usr/bin/virt-sysprep
The VM intended as a template is shut down.

You must know where the disk image for the source VM is located, and be the owner of the VM’s disk image file.

Note that disk images for VMs created in the system connection of libvirt are by default located in the /var/lib/libvirt/images directory and owned by the root user:

```bash
# ls -la /var/lib/libvirt/images
-rw-------.  1 root root  9665380352 Jul 23 14:50 a-really-important-vm.qcow2
-rw-------.  1 root root  8591507456 Jul 26  2017 an-actual-vm-that-i-use.qcow2
-rw-------.  1 root root  8591507456 Jul 26  2017 totally-not-a-fake-vm.qcow2
-rw-------.  1 root root 10739318784 Sep 20 17:57 another-vm-example.qcow2
```

Optional: Any important data on the VM’s disk has been backed up. If you want to preserve the source VM intact, clone it first and edit the clone to create a template.

Procedure

1. Ensure you are logged in as the owner of the VM’s disk image:

```bash
# whoami
root
```

2. Optional: Copy the disk image of the VM.

```bash
# cp /var/lib/libvirt/images/a-really-important-vm.qcow2 /var/lib/libvirt/images/a-really-
important-vm-original.qcow2
```

This is used later to verify the VM was successfully turned into a template.

3. Use the following command, and replace /var/lib/libvirt/images/a-really-important-vm.qcow2 with the path to the disk image of the source VM.

```bash
# virt-sysprep -a /var/lib/libvirt/images/a-really-important-vm.qcow2
```

Verification

To confirm that the process was successful, compare the modified disk image to the original one. The following example shows a successful creation of a template:

```bash
# virt-diff -a /var/lib/libvirt/images/a-really-important-vm-orig.qcow2 -A /var/lib/libvirt/images/a-really-important-vm.qcow2
- - 0644  1001 /etc/group-
- - 0000  797 /etc/gshadow-
= - 0444  33 /etc/machine-id
[...]
- - 0600  409 /home/username/.bash_history
```
11.2.2. Creating a virtual machine template manually

To create a template from an existing virtual machine (VM), you can manually reset or unconfigure a guest VM to prepare it for cloning.

Prerequisites

- Ensure that you know the location of the disk image for the source VM and are the owner of the VM's disk image file.
- Note that disk images for VMs created in the system connection of libvirt are by default located in the /var/lib/libvirt/images directory and owned by the root user:

```
# ls -la /var/lib/libvirt/images
-rw-------. 1 root root 9665380352 Jul 23 14:50 a-really-important-vm.qcow2
-rw-------. 1 root root 8591507456 Jul 26  2017 an-actual-vm-that-i-use.qcow2
-rw-------. 1 root root 8591507456 Jul 26  2017 totally-not-a-fake-vm.qcow2
-rw-------. 1 root root 10739318784 Sep 20 17:57 another-vm-example.qcow2
```

- Ensure that the VM is shut down.
- **Optional:** Any important data on the VM's disk has been backed up. If you want to preserve the source VM intact, clone it first and edit the clone to create a template.

Procedure

1. Configure the VM for cloning:
   a. Install any software needed on the clone.
   b. Configure any non-unique settings for the operating system.
   c. Configure any non-unique application settings.

2. Remove the network configuration:
   a. Remove any persistent udev rules using the following command:

```
# rm -f /etc/udev/rules.d/70-persistent-net.rules
```

**NOTE**

If udev rules are not removed, the name of the first NIC might be `eth1` instead of `eth0`.
b. Remove unique network details from ifcfg scripts by editing `/etc/sysconfig/network-scripts/ifcfg-eth[x]` as follows:

i. Remove the HWADDR and Static lines:

```
NOTE
If the HWADDR does not match the new guest’s MAC address, the ifcfg will be ignored.
```

```
DEVICE=eth[x] BOOTPROTO=none ONBOOT=yes #NETWORK=10.0.1.0 <- REMOVE #NETMASK=255.255.255.0 <- REMOVE #IPADDR=10.0.1.20 <- REMOVE #HWADDR=xx:xx:xx:xx:xx <- REMOVE #USERCTL=no <- REMOVE #
```

Remove any other "unique or non-desired settings, such as UUID."

ii. Configure DHCP but do not include HWADDR or any other unique information:

```
DEVICE=eth[x] BOOTPROTO= dhcp ONBOOT=yes
```

c. Ensure the following files also contain the same content, if they exist on your system:

- `/etc/sysconfig/networking/devices/ifcfg-eth[x]`
- `/etc/sysconfig/networking/profiles/default/ifcfg-eth[x]`

```
NOTE
If you had used NetworkManager or any special settings with the VM, ensure that any additional unique information is removed from the ifcfg scripts.
```

3. Remove registration details:

- For VMs registered on the Red Hat Network (RHN):

```
# rm /etc/sysconfig/rhn/systemid
```

- For VMs registered with Red Hat Subscription Manager (RHSM):
  - If you do not plan to use the original VM:

```
# subscription-manager unsubscribe --all # subscription-manager unregister #
subscription-manager clean
```

  - If you plan to use the original VM:

```
# subscription-manager clean
```
NOTE
The original RHSM profile remains in the Portal along with your ID code. Use the following command to reactivate your RHSM registration on the VM after it is cloned:

```
subscription-manager register --consumerid=71rd64fx-6216-4409-bf3a-e4b7c7bd8ac9
```

4. Remove other unique details:
   a. Remove ssh public/private key pairs:

```
rm -rf /etc/ssh/ssh_host_example
```

   b. Remove any other application-specific identifiers or configurations that might cause conflicts if running on multiple machines.

5. Remove the `gnome-initial-setup-done` file to configure the VM to run the configuration wizard on the next boot:

```
rm ~/.config/gnome-initial-setup-done
```

NOTE
The wizard that runs on the next boot depends on the configurations that have been removed from the VM. In addition, on the first boot of the clone, it is recommended that you change the hostname.

11.3. CLONING A VIRTUAL MACHINE USING THE COMMAND-LINE INTERFACE

To quickly create a new virtual machine (VM) with a specific set of properties, for example for testing purposes, you can clone an existing VM. To do so using the CLI, follow the instructions below.

**Prerequisites**

- The source VM is shut down.
- Ensure that there is sufficient disk space to store the cloned disk images.
- **Optional:** When creating multiple VM clones, remove unique data and settings from the source VM to ensure the cloned VMs work properly. For instructions, see `Creating virtual machine templates`.

**Procedure**

1. Use the `virt-clone` utility with options that are appropriate for your environment and use case.

   **Sample use cases**

   - The following command clones a local VM named `doppelganger` and creates the `doppelganger-clone` VM. It also creates the `doppelganger-clone.qcow2` disk image in the same location as the disk image of the original VM, and with the same data:
**# virt-clone --original doppelganger --auto-clone**
Allocating 'doppelganger-clone.qcow2' | 50.0 GB  00:05:37

Clone 'doppelganger-clone' created successfully.

- The following command clones a VM named *geminus1*, and creates a local VM named *geminus2*, which uses only two of *geminus1*’s multiple disks.

**# virt-clone --original geminus1 --name geminus2 --file /var/lib/libvirt/images/disk1.qcow2 --file /var/lib/libvirt/images/disk2.qcow2**
Allocating 'disk1-clone.qcow2' | 78.0 GB  00:05:37
Allocating 'disk2-clone.qcow2' | 80.0 GB  00:05:37

Clone 'geminus2' created successfully.

- To clone your VM to a different host, migrate the VM without undefining it on the local host. For example, the following commands clone the previously created *geminus2* VM to the 10.0.0.1 remote system, including its local disks. Note that using these commands also requires root privileges for 10.0.0.1.

**# virsh migrate --offline --persistent geminus2 qemu+ssh://root@10.0.0.1/system root@10.0.0.1’s password:**

**# scp /var/lib/libvirt/images/disk1-clone.qcow2 root@10.0.0.1/user@remote_host.com://var/lib/libvirt/images/**

**# scp /var/lib/libvirt/images/disk2-clone.qcow2 root@10.0.0.1/user@remote_host.com://var/lib/libvirt/images/**

**Verification**

To verify the VM has been successfully cloned and is working correctly:

1. Confirm the clone has been added to the list of VMs on your host.

**# virsh list --all**

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>doppelganger</td>
<td>shut off</td>
</tr>
<tr>
<td></td>
<td>doppelganger-clone</td>
<td>shut off</td>
</tr>
</tbody>
</table>

2. Start the clone and observe if it boots up.

**# virsh start doppelganger-clone**

Domain 'doppelganger-clone' started

**Additional resources**

- The **virt-clone** man page
- **Migrating virtual machines**

**11.4. CLONING A VIRTUAL MACHINE USING THE WEB CONSOLE**
To quickly create new virtual machines (VMs) with a specific set of properties, you can clone a VM that you had previously configured. The following instructions explain how to do so using the web console.

**NOTE**
Cloning a VM also clones the disks associated with that VM.

**Prerequisites**

- The web console VM plug-in is installed on your system.
- Ensure that the VM you want to clone is shut down.

**Procedure**

1. In the Virtual Machines interface of the web console, click the Menu button ⋮ of the VM that you want to clone.
   A drop down menu appears with controls for various VM operations.

2. Click **Clone**.
   The Create a clone VM dialog appears.

   ![Create a clone VM dialog](image)

3. **Optional**: Enter a new name for the VM clone.

4. Click **Clone**.
   A new VM is created based on the source VM.

**Verification**

- Confirm whether the cloned VM appears in the list of VMs available on your host.
CHAPTER 12. MIGRATING VIRTUAL MACHINES

If the current host of a virtual machine (VM) becomes unsuitable or cannot be used anymore, or if you want to redistribute the hosting workload, you can migrate the VM to another KVM host.

12.1. HOW MIGRATING VIRTUAL MACHINES WORKS

The essential part of virtual machine (VM) migration is copying the XML configuration of a VM to a different host machine. If the migrated VM is not shut down, the migration also transfers the state of the VM’s memory and any virtualized devices to a destination host machine. For the VM to remain functional on the destination host, the VM’s disk images must remain available to it.

By default, the migrated VM is transient on the destination host, and remains defined also on the source host.

You can migrate a running VM using live or non-live migrations. To migrate a shut-off VM, you must use an offline migration. For details, see the following table.

Table 12.1. VM migration types

<table>
<thead>
<tr>
<th>Migration type</th>
<th>Description</th>
<th>Use case</th>
<th>Storage requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Live migration</td>
<td>The VM continues to run on the source host machine while KVM is transferring the VM’s memory pages to the destination host. When the migration is nearly complete, KVM very briefly suspends the VM, and resumes it on the destination host.</td>
<td>Useful for VMs that require constant uptime. However, VMs that modify memory pages faster than KVM can transfer them, such as VMs under heavy I/O load, cannot be live-migrated, and non-live migration must be used instead.</td>
<td>The VM’s disk images must be located on a shared network, accessible both to the source host and the destination host.</td>
</tr>
<tr>
<td>Non-live migration</td>
<td>Suspends the VM, copies its configuration and its memory to the destination host, and resumes the VM.</td>
<td>Creates downtime for the VM, but is generally more reliable than live migration. Recommended for VMs under heavy I/O load.</td>
<td>The VM’s disk images must be located on a shared network, accessible both to the source host and the destination host.</td>
</tr>
<tr>
<td>Offline migration</td>
<td>Moves the VM’s configuration to the destination host</td>
<td>Recommended for shut-off VMs.</td>
<td>The VM’s disk images do not have to be available on a shared network, and can be copied or moved manually to the destination host instead.</td>
</tr>
</tbody>
</table>

Additional resources

- Benefits of migrating virtual machines
- Sharing virtual machine disk images with other hosts
12.2. BENEFITS OF MIGRATING VIRTUAL MACHINES

Migrating virtual machines (VMs) can be useful for:

Load balancing
VMs can be moved to host machines with lower usage if their host becomes overloaded, or if another host is under-utilized.

Hardware independence
When you need to upgrade, add, or remove hardware devices on the host machine, you can safely relocate VMs to other hosts. This means that VMs do not experience any downtime for hardware improvements.

Energy saving
VMs can be redistributed to other hosts, and the unloaded host systems can thus be powered off to save energy and cut costs during low usage periods.

Geographic migration
VMs can be moved to another physical location for lower latency or when required for other reasons.

12.3. LIMITATIONS FOR MIGRATING VIRTUAL MACHINES

Before migrating virtual machines (VMs) in RHEL 9, ensure you are aware of the migration’s limitations.

- It is possible to perform live storage migration on RHEL 9. However, a support for live storage migration is provided only with the Red Hat Virtualization support subscription.

- Migrating VMs from or to a session connection of libvirt is unreliable and therefore not recommended.

- VMs that use certain features and configurations will not work correctly if migrated, or the migration will fail. Such features include:
  - Device passthrough
  - SR-IOV device assignment
  - Mediated devices, such as vGPUs

- A migration between hosts that use Non-Uniform Memory Access (NUMA) pinning works only if the hosts have similar topology. However, the performance on running workloads might be negatively affected by the migration.

- The emulated CPUs, both on the source VM and the destination VM, must be identical, otherwise the migration might fail. Any differences between the VMs in the following CPU related areas can cause problems with the migration:
  - CPU model

  **NOTE**
  Migrating between an Intel 64 host and an AMD64 host is unsupported, even though they share the x86-64 instruction set.

- Firmware settings
Live migrating a VM that uses more than 1 TB of memory may in some cases not work reliably. The stability of such a migration depends on the following:

- The current workload of the VM
- The network bandwidth that the host can use for migration
- The downtime that your deployment is able to support

For live migration scenarios that involve VMs with more than 1 TB of memory, customers should consult Red Hat.

### 12.4. SHARING VIRTUAL MACHINE DISK IMAGES WITH OTHER HOSTS

To perform a live migration of a virtual machine (VM) between supported KVM hosts, shared VM storage is required. This section provides instructions for sharing a locally stored VM image with the source host and the destination host using the NFS protocol.

**Prerequisites**

- The VM intended for migration is shut down.
- **Optional:** A host system is available for hosting the storage that is not the source or destination host, but both the source and the destination host can reach it through the network. This is the optimal solution for shared storage and is recommended by Red Hat.
- Make sure that NFS file locking is not used as it is not supported in KVM.
- The NFS is installed and enabled on the source and destination hosts. If it is not:
  a. Install the NFS packages:

      ```bash
      # dnf install nfs-utils
      ```
  b. Make sure that the ports for NFS, such as 2049, are open in the firewall.

      ```bash
      # firewall-cmd --permanent --add-service=nfs
      # firewall-cmd --permanent --add-service=mountd
      # firewall-cmd --permanent --add-service=rpc-bind
      # firewall-cmd --permanent --add-port=2049/tcp
      # firewall-cmd --permanent --add-port=2049/udp
      # firewall-cmd --reload
      ```
  c. Start the NFS service.

      ```bash
      # systemctl start nfs-server
      ```
Procedure

1. Connect to the host that will provide shared storage. In this example, it is the `cargo-bay` host:

   ```
   # ssh root@cargo-bay
   root@cargo-bay's password:
   Last login: Mon Sep 24 12:05:36 2019
   root~#
   ```

2. Create a directory that will hold the disk image and will be shared with the migration hosts.

   ```
   # mkdir /var/lib/libvirt/shared-images
   ```

3. Copy the disk image of the VM from the source host to the newly created directory. For example, the following copies the disk image of the `wanderer1` VM to the `/var/lib/libvirt/shared-images/` directory on the `cargo-bay` host:

   ```
   # scp /var/lib/libvirt/images/wanderer1.qcow2 root@cargo-bay:/var/lib/libvirt/shared-images/wanderer1.qcow2
   ```

4. On the host that you want to use for sharing the storage, add the sharing directory to the `/etc/exports` file. The following example shares the `/var/lib/libvirt/shared-images` directory with the `source-example` and `dest-example` hosts:

   ```
   /var/lib/libvirt/shared-images source-example(rw,no_root_squash) dest-example(rw,no_root_squash)
   ```

5. On both the source and destination host, mount the shared directory in the `/var/lib/libvirt/images` directory:

   ```
   # mount cargo-bay:/var/lib/libvirt/shared-images /var/lib/libvirt/images
   ```

Verification

- To verify the process was successful, start the VM on the source host and observe if it boots correctly.

12.5. MIGRATING A VIRTUAL MACHINE USING THE COMMAND-LINE INTERFACE

If the current host of a virtual machine (VM) becomes unsuitable or cannot be used anymore, or if you want to redistribute the hosting workload, you can migrate the VM to another KVM host. This section provides instructions and examples for various scenarios of such migrations.

Prerequisites

- The source host and the destination host both use the KVM hypervisor.
- The source host and the destination host are able to reach each other over the network. Use the `ping` utility to verify this.
Ensure the following ports are open on the destination host.

- Port 22 is needed for connecting to the destination host by using SSH.
- Port 16509 is needed for connecting to the destination host by using TLS.
- Port 16514 is needed for connecting to the destination host by using TCP.
- Ports 49152–49215 are needed by QEMU for transferring the memory and disk migration data.

For the migration to be supportable by Red Hat, the source host and destination host must be using specific operating systems and machine types. To ensure this is the case, see Supported hosts for virtual machine migration.

The disk images of VMs that will be migrated are located on a separate networked location accessible to both the source host and the destination host. This is optional for offline migration, but required for migrating a running VM. For instructions to set up such shared VM storage, see Sharing virtual machine disk images with other hosts.

When migrating a running VM, your network bandwidth must be higher than the rate in which the VM generates dirty memory pages. To obtain the dirty page rate of your VM before you start the live migration, do the following:

a. Monitor the rate of dirty page generation of the VM for a short period of time.

```
# virsh domdirtyrate-calc vm-name 30
```

b. After the monitoring finishes, obtain its results:

```
# virsh domstats vm-name --dirtyrate
Domain: 'vm-name'
  dirtyrate.calc_status=2
  dirtyrate.calc_start_time=200942
  dirtyrate.calc_period=30
  dirtyrate.megabytes_per_second=2
```

In this example, the VM is generating 2 MB of dirty memory pages per second. Attempting to live-migrate such a VM on a network with a bandwidth of 2 MB/s or less will cause the live migration not to progress if you do not pause the VM or lower its workload.

To ensure that the live migration finishes successfully, Red Hat recommends that your network bandwidth is significantly greater than the VM’s dirty page generation rate.

When migrating an existing VM in a public bridge tap network, the source and destination hosts must be located on the same network. Otherwise, the VM network will not operate after migration.

When performing a VM migration, the `virsh` client on the source host can use one of several protocols to connect to the libvirt daemon on the destination host. Examples in the following procedure use an SSH connection, but you can choose a different one.

- If you want libvirt to use an SSH connection, ensure that the `virtqemud` socket is enabled and running on the destination host.
# systemctl enable --now virtqemud.socket

- If you want libvirt to use a TLS connection, ensure that the `virtproxyd-tls` socket is enabled and running on the destination host.

# systemctl enable --now virtproxyd-tls.socket

- If you want libvirt to use a TCP connection, ensure that the `virtproxyd-tcp` socket is enabled and running on the destination host.

# systemctl enable --now virtproxyd-tcp.socket

Procedure

1. Use the `virsh migrate` command with options appropriate for your migration requirements.
   - The following migrates the `wanderer1` VM from your local host to the system connection of the `dest-example` host using an SSH tunnel. The VM will remain running during the migration.

   ```
   # virsh migrate --persistent --live wanderer1 qemu+ssh://dest-example/system
   ```

   - The following enables you to make manual adjustments to the configuration of the `wanderer2` VM running on your local host, and then migrates the VM to the `dest-example` host. The migrated VM will automatically use the updated configuration.

   ```
   # virsh dumpxml --migratable wanderer2 >wanderer2.xml
   # vi wanderer2.xml
   # virsh migrate --live --persistent --xml wanderer2.xml wanderer2 qemu+ssh://dest-example/system
   ```

   This procedure can be useful for example when the destination host needs to use a different path to access the shared VM storage or when configuring a feature specific to the destination host.

   - The following suspends the `wanderer3` VM from the `source-example` host, migrates it to the `dest-example` host, and instructs it to use the adjusted XML configuration, provided by the `wanderer3-alt.xml` file. When the migration is completed, `libvirt` resumes the VM on the destination host.

   ```
   # virsh migrate wanderer3 qemu+ssh://source-example/system qemu+ssh://dest-example/system --xml wanderer3-alt.xml
   ```

   After the migration, the VM is in the shut off state on the source host, and the migrated copy is deleted after it is shut down.

   - The following deletes the shut-down `wanderer4` VM from the `source-example` host, and moves its configuration to the `dest-example` host.

   ```
   # virsh migrate --offline --persistent --undefinesource wanderer4 qemu+ssh://source-example/system qemu+ssh://dest-example/system
   ```

Note that this type of migration does not require moving the VM’s disk image to shared storage.
Note that this type of migration does not require moving the VM’s disk image to shared storage. However, for the VM to be usable on the destination host, you also need to migrate the VM’s disk image. For example:

```bash
# scp root@source-example:/var/lib/libvirt/images/wanderer4.qcow2 root@dest-example:/var/lib/libvirt/images/wanderer4.qcow2
```

2. Wait for the migration to complete. The process may take some time depending on network bandwidth, system load, and the size of the VM. If the `--verbose` option is not used for `virsh migrate`, the CLI does not display any progress indicators except errors. When the migration is in progress, you can use the `virsh domjobinfo` utility to display the migration statistics.

Verification

- On the destination host, list the available VMs to verify if the VM has been migrated:

  ```bash
  # virsh list
  Id Name       State
  ------------------
  10 wanderer1  running
  ```

  If the migration is still running, this command will list the VM state as `paused`.

Troubleshooting

- In some cases, the target host will not be compatible with certain values of the migrated VM’s XML configuration, such as the network name or CPU type. As a result, the VM will fail to boot on the target host. To fix these problems, you can update the problematic values by using the `virsh edit` command. After updating the values, you must restart the VM for the changes to be applied.

- If a live migration is taking a long time to complete, this may be because the VM is under heavy load and too many memory pages are changing for live migration to be possible. To fix this problem, change the migration to a non-live one by suspending the VM.

  ```bash
  # virsh suspend wanderer1
  ```

Additional resources

- The `virsh migrate --help` command
- The `virsh` man page

### 12.6. LIVE MIGRATING A VIRTUAL MACHINE USING THE WEB CONSOLE

If you wish to migrate a virtual machine (VM) that is performing tasks which require it to be constantly running, you can migrate that VM to another KVM host without shutting it down. This is also known as live migration. The following instructions explain how to do so using the web console.
WARNING
For tasks that modify memory pages faster than KVM can transfer them, such as heavy I/O load tasks, it is recommended that you do not live migrate the VM.

Prerequisites

- The web console VM plug-in is installed on your system.
- The source and destination hosts are running.
- Ensure the following ports are open on the destination host.
  - Port 22 is needed for connecting to the destination host by using SSH.
  - Port 16509 is needed for connecting to the destination host by using TLS.
  - Port 16514 is needed for connecting to the destination host by using TCP.
  - Ports 49152-49215 are needed by QEMU for transferring the memory and disk migration data.
- The VM's disk images are located on a shared storage that is accessible to the source host as well as the destination host.
- When migrating a running VM, your network bandwidth must be higher than the rate in which the VM generates dirty memory pages.

To obtain the dirty page rate of your VM before you start the live migration, do the following in your command-line interface:

a. Monitor the rate of dirty page generation of the VM for a short period of time.

```
# virsh domdirtyrate-calc vm-name 30
```

b. After the monitoring finishes, obtain its results:

```
# virsh domstats vm-name --dirtyrate
Domain: 'vm-name'
dirtyrate.calc_status=2
dirtyrate.calc_start_time=200942
dirtyrate.calc_period=30
dirtyrate.megabytes_per_second=2
```

In this example, the VM is generating 2 MB of dirty memory pages per second. Attempting to live-migrate such a VM on a network with a bandwidth of 2 MB/s or less will cause the live migration not to progress if you do not pause the VM or lower its workload.

To ensure that the live migration finishes successfully, Red Hat recommends that your network bandwidth is significantly greater than the VM’s dirty page generation rate.

Procedure
1. In the Virtual Machines interface of the web console, click the Menu button ⋮ of the VM that you want to migrate. A drop down menu appears with controls for various VM operations.

2. Click Migrate
   The Migrate VM to another host dialog appears.

3. Enter the URI of the destination host.

4. Configure the duration of the migration:
   - **Permanent** - Do not check the box if you wish to migrate the VM permanently. Permanent migration completely removes the VM configuration from the source host.
   - **Temporary** - Temporary migration migrates a copy of the VM to the destination host. This copy is deleted from the destination host when the VM is shut down. The original VM remains on the source host.

5. Click Migrate
   Your VM is migrated to the destination host.

**Verification**
To verify whether the VM has been successfully migrated and is working correctly:
- Confirm whether the VM appears in the list of VMs available on the destination host.
- Start the migrated VM and observe if it boots up.

**12.7. SUPPORTED HOSTS FOR VIRTUAL MACHINE MIGRATION**
For the virtual machine (VM) migration to work properly and be supported by Red Hat, the source and destination hosts must be specific RHEL versions and machine types. The following table shows supported VM migration paths.

**Table 12.2. Live migration compatibility**

<table>
<thead>
<tr>
<th>Migration method</th>
<th>Release type</th>
<th>Future version example</th>
<th>Support status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td>Minor release</td>
<td>9.0.1 → 9.1</td>
<td>On supported RHEL 9 systems: machine type q35.</td>
</tr>
<tr>
<td>Backward</td>
<td>Minor release</td>
<td>9.1 → 9.0.1</td>
<td>On supported RHEL 9 systems: machine type q35.</td>
</tr>
</tbody>
</table>

**NOTE**

Support level is different for other virtualization solutions provided by Red Hat, including RHOSP and OpenShift Virtualization.
CHAPTER 13. MANAGING VIRTUAL DEVICES

One of the most effective ways to manage the functionality, features, and performance of a virtual machine (VM) is to adjust its virtual devices.

The following sections provide a general overview of what virtual devices are, and instructions on how to manage them using the CLI or the web console.

13.1. HOW VIRTUAL DEVICES WORK

Just like physical machines, virtual machines (VMs) require specialized devices to provide functions to the system, such as processing power, memory, storage, networking, or graphics. Physical systems usually use hardware devices for these purposes. However, because VMs work as software implements, they need to use software abstractions of such devices instead, referred to as virtual devices.

The basics

Virtual devices attached to a VM can be configured when creating the VM, and can also be managed on an existing VM. Generally, virtual devices can be attached or detached from a VM only when the VM is shut off, but some can be added or removed when the VM is running. This feature is referred to as device hot plug and hot unplug.

When creating a new VM, libvirt automatically creates and configures a default set of essential virtual devices, unless specified otherwise by the user. These are based on the host system architecture and machine type, and usually include:

- the CPU
- memory
- a keyboard
- a network interface controller (NIC)
- various device controllers
- a video card
- a sound card

To manage virtual devices after the VM is created, use the command-line interface (CLI). However, to manage virtual storage devices and NICs, you can also use the RHEL 9 web console.

Performance or flexibility

For some types of devices, RHEL 9 supports multiple implementations, often with a trade-off between performance and flexibility.

For example, the physical storage used for virtual disks can be represented by files in various formats, such as qcow2 or raw, and presented to the VM using a variety of controllers:

- an emulated controller
- virtio-scsi
- virtio-blk
An emulated controller is slower than a **virtio** controller, because **virtio** devices are designed specifically for virtualization purposes. On the other hand, emulated controllers make it possible to run operating systems that have no drivers for **virtio** devices. Similarly, **virtio-scsi** offers a more complete support for SCSI commands, and makes it possible to attach a larger number of disks to the VM. Finally, **virtio-blk** provides better performance than both **virtio-scsi** and emulated controllers, but a more limited range of use-cases. For example, attaching a physical disk as a LUN device to a VM is not possible when using **virtio-blk**.

For more information on types of virtual devices, see [Types of virtual devices](#).

### 13.2. TYPES OF VIRTUAL DEVICES

Virtualization in RHEL 9 can present several distinct types of virtual devices that you can attach to virtual machines (VMs):

#### Emulated devices

Emulated devices are software implementations of widely used physical devices. Drivers designed for physical devices are also compatible with emulated devices. Therefore, emulated devices can be used very flexibly. However, since they need to faithfully emulate a particular type of hardware, emulated devices may suffer a significant performance loss compared with the corresponding physical devices or more optimized virtual devices.

The following types of emulated devices are supported:

- Virtual CPUs (vCPUs), with a large choice of CPU models available. The performance impact of emulation depends significantly on the differences between the host CPU and the emulated vCPU.
- Emulated system components, such as PCI bus controllers.
- Emulated storage controllers, such as SATA, SCSI or even IDE.
- Emulated sound devices, such as ICH9, ICH6 or AC97.
- Emulated graphics cards, such as VGA cards.
- Emulated network devices, such as rtl8139.

#### Paravirtualized devices

Paravirtualization provides a fast and efficient method for exposing virtual devices to VMs. Paravirtualized devices expose interfaces that are designed specifically for use in VMs, and thus significantly increase device performance. RHEL 9 provides paravirtualized devices to VMs using the **virtio** API as a layer between the hypervisor and the VM. The drawback of this approach is that it requires a specific device driver in the guest operating system. It is recommended to use paravirtualized devices instead of emulated devices for VM whenever possible, notably if they are running I/O intensive applications. Paravirtualized devices decrease I/O latency and increase I/O throughput, in some cases bringing them very close to bare-metal performance. Other paravirtualized devices also add functionality to VMs that is not otherwise available.

The following types of paravirtualized devices are supported:

- The paravirtualized network device (**virtio-net**).
- Paravirtualized storage controllers:
  - virtio-blk - provides block device emulation.
  - virtio-scsi - provides more complete SCSI emulation.
- The paravirtualized clock.
- The paravirtualized serial device (virtio-serial).
- The balloon device (virtio-balloon), used to dynamically distribute memory between a VM and its host.
- The paravirtualized random number generator (virtio-rng).

Physically shared devices

Certain hardware platforms enable VMs to directly access various hardware devices and components. This process is known as device assignment or passthrough. When attached in this way, some aspects of the physical device are directly available to the VM as they would be to a physical machine. This provides superior performance for the device when used in the VM. However, devices physically attached to a VM become unavailable to the host, and also cannot be migrated.

Nevertheless, some devices can be shared across multiple VMs. For example, a single physical device can in certain cases provide multiple mediated devices, which can then be assigned to distinct VMs.

The following types of passthrough devices are supported:

- USB, PCI, and SCSI passthrough - expose common industry standard buses directly to VMs in order to make their specific features available to guest software.
- Single-root I/O virtualization (SR-IOV) - a specification that enables hardware-enforced isolation of PCI Express resources. This makes it safe and efficient to partition a single physical PCI resource into virtual PCI functions. It is commonly used for network interface cards (NICs).
- N_Port ID virtualization (NPIV) - a Fibre Channel technology to share a single physical host bus adapter (HBA) with multiple virtual ports.
- GPUs and vGPUs - accelerators for specific kinds of graphic or compute workloads. Some GPUs can be attached directly to a VM, while certain types also offer the ability to create virtual GPUs (vGPUs) that share the underlying physical hardware.

13.3. MANAGING DEVICES ATTACHED TO VIRTUAL MACHINES USING THE CLI

To modify the functionality of your virtual machine (VM), you can manage the devices attached to your VM using the command-line interface (CLI).

You can use the CLI to:

- Attach devices
- Modify devices
- Remove devices
13.3.1. Attaching devices to virtual machines

You can add a specific functionality to your virtual machines (VMs) by attaching a new virtual device.

The following procedure demostrates how to create and attach virtual devices to your virtual machines (VMs) using the command-line interface (CLI). Some devices can also be attached to VMs using the RHEL web console.

For example, you can increase the storage capacity of a VM by attaching a new virtual disk device to it. This is also referred to as memory hot plug.

WARNING
Removing a memory device from a VM, also known as memory hot unplug, is not supported in RHEL 9, and Red Hat highly discourages its use.

Prerequisites

- Obtain the required options for the device you intend to attach to a VM. To see the available options for a specific device, use the `virt-xml --device=` command. For example:

  ```
  # virt-xml --network=?
  --network options:
  [..]
  address.unit
  boot_order
  clearxml
  driver_name
  [..]
  ```

Procedure

1. To attach a device to a VM, use the `virt-xml --add-device` command, including the definition of the device and the required options:

   - For example, the following command creates a 20GB `newdisk` qcow2 disk image in the `/var/lib/libvirt/images/` directory, and attaches it as a virtual disk to the running `testguest` VM on the next start-up of the VM:

     ```
     # virt-xml testguest --add-device --disk
     /var/lib/libvirt/images/newdisk.qcow2,format=qcow2,size=20
     Domain 'testguest' defined successfully.
     Changes will take effect after the domain is fully powered off.
     ```

   - The following attaches a USB flash drive, attached as device 004 on bus 002 on the host, to the `testguest2` VM while the VM is running:

     ```
     # virt-xml testguest2 --add-device --update --hostdev 002.004
     Device hotplug successful.
     Domain 'testguest2' defined successfully.
     ```
The bus-device combination for defining the USB can be obtained using the `lsusb` command.

**Verification**

To verify the device has been added, do any of the following:

- Use the `virsh dumpxml` command and see if the device’s XML definition has been added to the `<devices>` section in the VM’s XML configuration.
  
  For example, the following output shows the configuration of the `testguest` VM and confirms that the 002.004 USB flash disk device has been added.

```
# virsh dumpxml testguest
[...]
<hostdev mode='subsystem' type='usb' managed='yes'>
  <source>
    <vendor id='0x4146'/>
    <product id='0x902e'/>
    <address bus='2' device='4'/>
  </source>
  <alias name='hostdev0'/>
  <address type='usb' bus='0' port='3'/>
</hostdev>
[...]
```

- Run the VM and test if the device is present and works properly.

**Additional resources**

- The `man virt-xml` command

**13.3.2. Modifying devices attached to virtual machines**

You can change the functionality of your virtual machines (VMs) by editing a configuration of the attached virtual devices. For example, if you want to optimize the performance of your VMs, you can change their virtual CPU models to better match the CPUs of the hosts.

The following procedure provides general instructions for modifying virtual devices using the command-line interface (CLI). Some devices attached to your VM, such as disks and NICs, can also be modified using the RHEL 9 web console.

**Prerequisites**

- Obtain the required options for the device you intend to attach to a VM. To see the available options for a specific device, use the `virt-xml --device=?` command. For example:

```
# virt-xml --network=?
--network options:
  [...]
  address.unit
  boot_order
  clearxml
  driver_name
  [...]
```
Optional: Back up the XML configuration of your VM by using `virsh dumpxml vm-name` and sending the output to a file. For example, the following backs up the configuration of your Motoko VM as the `motoko.xml` file:

```bash
# virsh dumpxml Motoko > motoko.xml
# cat motoko.xml
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  <name>Motoko</name>
  <uuid>ede29304-fe0c-4ca4-abcd-d246481acd18</uuid>
  [...]
</domain>
```

Procedure

1. Use the `virt-xml --edit` command, including the definition of the device and the required options:
   For example, the following clears the `<cpu>` configuration of the shut-off `testguest` VM and sets it to `host-model`:

   ```bash
   # virt-xml testguest --edit --cpu host-model,clearxml=yes
   Domain 'testguest' defined successfully.
   ```

Verification

To verify the device has been modified, do any of the following:

- Run the VM and test if the device is present and reflects the modifications.

- Use the `virsh dumpxml` command and see if the device’s XML definition has been modified in the VM’s XML configuration.
  For example, the following output shows the configuration of the `testguest` VM and confirms that the CPU mode has been configured as `host-model`.

```bash
# virsh dumpxml testguest
[...]
  <cpu mode='host-model' check='partial'>
    <model fallback='allow'/>
  </cpu>
[...]
```

Troubleshooting

- If modifying a device causes your VM to become unbootable, use the `virsh define` utility to restore the XML configuration by reloading the XML configuration file you backed up previously.

```bash
# virsh define testguest.xml
```

NOTE

For small changes to the XML configuration of your VM, you can use the `virsh edit` command – for example `virsh edit testguest`. However, do not use this method for more extensive changes, as it is more likely to break the configuration in ways that could prevent the VM from booting.
13.3.3. Removing devices from virtual machines

You can change the functionality of your virtual machines (VMs) by removing a virtual device. For example, you can remove a virtual disk device from one of your VMs if it is no longer needed.

The following procedure demonstrates how to remove virtual devices from your virtual machines (VMs) using the command-line interface (CLI). Some devices, such as disks or NICs, can also be removed from VMs using the RHEL 9 web console.

Prerequisites

- Optional: Back up the XML configuration of your VM by using `virsh dumpxml vm-name` and sending the output to a file. For example, the following backs up the configuration of your *Motoko* VM as the `motoko.xml` file:

```
# virsh dumpxml Motoko > motoko.xml
# cat motoko.xml
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  <name>Motoko</name>
  <uuid>ede29304-fe0c-4ca4-abcd-d246481acd18</uuid>
  [...]
</domain>
```

Procedure

1. Use the `virt-xml --remove-device` command, including a definition of the device. For example:

   - The following removes the storage device marked as `vdb` from the running `testguest` VM after it shuts down:

     ```
     # virt-xml testguest --remove-device --disk target=vdb
     Domain 'testguest' defined successfully.
     Changes will take effect after the domain is fully powered off.
     ```

   - The following immediately removes a USB flash drive device from the running `testguest2` VM:

     ```
     # virt-xml testguest2 --remove-device --update --hostdev type=usb
     Device hotunplug successful.
     Domain 'testguest2' defined successfully.
     ```

Troubleshooting

- If removing a device causes your VM to become unbootable, use the `virsh define` utility to restore the XML configuration by reloading the XML configuration file you backed up previously.

```
# virsh define testguest.xml
```

Additional resources

- The `man virt-xml` command

Additional resources

- Red Hat Enterprise Linux 9 Configuring and managing virtualization
13.4. MANAGING HOST DEVICES USING THE WEB CONSOLE

To modify the functionality of your virtual machine (VM), you can manage the host devices attached to your VM using the Red Hat Enterprise Linux 9 web console.

Host devices are physical devices that are attached to the host system. Based on your requirements, you can enable your VMs to directly access these hardware devices and components.

You can use the web console to:

- View devices
- Attach devices
- Remove devices

13.4.1. Viewing devices attached to virtual machines using the web console

Before adding or modifying the devices attached to your virtual machine (VM), you may want to view the devices that are already attached to your VM. The following procedure provides instructions for viewing such devices using the web console.

Prerequisites

- The web console VM plug-in is installed on your system.

Procedure

1. In the Virtual Machines interface, click the VM whose information you want to see. A new page opens with detailed information about the VM.

2. Scroll to the Host devices section.
13.4.2. Attaching devices to virtual machines using the web console

To add specific functionalities to your virtual machine (VM), you can use the web console to attach host devices to the VM.

**NOTE**

Attaching multiple host devices at the same time does not work. You can attach only one device at a time.

For more information, see RHEL 9 Known Issues.

**Prerequisites**

- If you are attaching PCI devices, ensure that the status of the `managed` attribute of the `hostdev` element is set to `yes`.

**NOTE**

When attaching PCI devices to your VM, do not omit the `managed` attribute of the `hostdev` element, or set it to `no`. If you do so, PCI devices cannot automatically detach from the host when you pass them to the VM. They also cannot automatically reattach to the host when you turn off the VM.

As a consequence, the host may become unresponsive or shut down unexpectedly.

You can find the status of the `managed` attribute in your VM’s XML configuration. The following example opens the XML configuration of the Ag47 VM:

```
# virsh edit Ag47
```

- Back up important data from the VM.

- **Optional:** Back up the XML configuration of your VM. For example, to back up the Centurion VM:

```
# virsh dumpxml Centurion > Centurion.xml
```

- The web console VM plug-in is installed on your system.
Procedure

1. In the Virtual Machines interface, click the VM to which you want to attach a host device. A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM’s graphical interface.

2. Scroll to Host devices. The Host devices section displays information about the devices attached to the VM as well as options to Add or Remove devices.

<table>
<thead>
<tr>
<th>Host devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
</tr>
<tr>
<td>usb</td>
</tr>
<tr>
<td>usb</td>
</tr>
<tr>
<td>pc</td>
</tr>
</tbody>
</table>

3. Click Add host device. The Add host device dialog appears.

4. Select the device you wish to attach to the VM.

5. Click Add. The selected device is attached to the VM.

Verification

- Run the VM and check if the device appears in the Host devices section.

13.4.3. Removing devices from virtual machines using the web console
To free up resources, modify the functionalities of your VM, or both, you can use the web console to modify the VM and remove host devices that are no longer required.

WARNING
Removing attached USB host devices using the web console may fail because of incorrect correlation between the device and bus numbers of the USB device.

For more information, see RHEL 9 Known Issues.

As a workaround, remove the <hostdev> part of the USB device, from the VM’s XML configuration, using the "virsh" utility. The following example opens the XML configuration of the Ag47 VM:

```bash
# virsh edit Ag47
```

Prerequisites

- The web console VM plug-in is installed on your system.
- Optional: Back up the XML configuration of your VM by using `virsh dumpxml vm-name` and sending the output to a file. For example, the following backs up the configuration of your Motoko VM as the motoko.xml file:

```bash
# virsh dumpxml Motoko > motoko.xml
# cat motoko.xml
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  <name>Motoko</name>
  <uuid>ede29304-fe0c-4ca4-abcd-d246481acd18</uuid>
  […]
</domain>
```

Procedure

1. In the Virtual Machines interface, click the VM from which you want to remove a host device. A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM’s graphical interface.

2. Scroll to Host devices. The Host devices section displays information about the devices attached to the VM as well as options to Add or Remove devices.
3. Click the **Remove** button next to the device you want to remove from the VM. A remove device confirmation dialog appears.

![Remove USB host device dialog](image)

4. Click **Remove**. The device is removed from the VM.

**Troubleshooting**

- If removing a host device causes your VM to become unbootable, use the `virsh define` utility to restore the XML configuration by reloading the XML configuration file you backed up previously.

```
# virsh define motoko.xml
```

### 13.5. MANAGING VIRTUAL USB DEVICES

When using a virtual machine (VM), you can access and control a USB device, such as a flash drive or a webcam, that is attached to the host system. In this scenario, the host system passes control of the device to the VM. This is also known as a USB-passthrough.

The following sections provide information about using the command line to:

- **Attach a USB device** to a VM
- **Remove a USB device** from a VM

#### 13.5.1. Attaching USB devices to virtual machines

To attach a USB device to a virtual machine (VM), you can include the USB device information in the XML configuration file of the VM.

**Prerequisites**

- Ensure the device you want to pass through to the VM is attached to the host.

**Procedure**

1. Locate the bus and device values of the USB that you want to attach to the VM. For example, the following command displays a list of USB devices attached to the host. The device we will use in this example is attached on bus 001 as device 005.

```
# lsusb
[...]
Bus 001 Device 003: ID 2567:0a2b Intel Corp.
Bus 001 Device 005: ID 0407:6252 Kingston River 2.0
[...]
```
2. Use the `virt-xml` utility along with the `--add-device` argument. For example, the following command attaches a USB flash drive to the Library VM.

```bash
# virt-xml Library --add-device --hostdev 001.005
Domain 'Library' defined successfully.
```

**NOTE**

To attach a USB device to a running VM, add the `--update` argument to the previous command.

**Verification**

- Run the VM and test if the device is present and works as expected.
- Use the `virsh dumpxml` command to see if the device’s XML definition has been added to the `<devices>` section in the VM’s XML configuration file.

```bash
# virsh dumpxml Library
[...]
<hostdev mode='subsystem' type='usb' managed='yes'>
  <source>
    <vendor id='0x0407'/>
    <product id='0x6252'/>
    <address bus='1' device='5'/>
  </source>
  <alias name='hostdev0'/>
  <address type='usb' bus='0' port='3'/>
</hostdev>
[...]
```

**Additional resources**

- The `man virt-xml` command
- Attaching devices to virtual machines

### 13.5.2. Removing USB devices from virtual machines

To remove a USB device from a virtual machine (VM), you can remove the USB device information from the XML configuration of the VM.

**Procedure**

1. Locate the bus and device values of the USB that you want to remove from the VM. For example, the following command displays a list of USB devices attached to the host. The device we will use in this example is attached on bus 001 as device 005.

```bash
# lsusb
[...]
Bus 001 Device 003: ID 2567:0a2b Intel Corp.
Bus 001 Device 005: ID 0407:6252 Kingston River 2.0
[...]
```
2. Use the `virt-xml` utility along with the `--remove-device` argument. For example, the following command removes a USB flash drive, attached to the host as device 005 on bus 001, from the `Library` VM.

```
# virt-xml Library --remove-device --hostdev 001.005
Domain 'Library' defined successfully.
```

**NOTE**

To remove a USB device from a running VM, add the `--update` argument to the previous command.

**Verification**

- Run the VM and check if the device has been removed from the list of devices.

**Additional resources**

- The `man virt-xml` command
- [Attaching devices to virtual machines](#)

## 13.6. MANAGING VIRTUAL OPTICAL DRIVES

When using a virtual machine (VM), you can access information stored in an ISO image on the host. To do so, attach the ISO image to the VM as a virtual optical drive, such as a CD drive or a DVD drive.

The following sections provide information about using the command line to:

- Attach a drive and an ISO image to a VM
- Replace an ISO image in a virtual optical drive
- Remove an ISO image from a virtual optical drive
- Remove a drive from the VM

### 13.6.1. Attaching optical drives to virtual machines

To attach an ISO image as a virtual optical drive, edit the XML configuration file of the virtual machine (VM) and add the new drive.

**Prerequisites**

- You must store the ISO image on the local host.
- You must know the path to the ISO image.

**Procedure**

- Use the `virt-xml` utility with the `--add-device` argument. For example, the following command attaches the `Doc10` ISO image, stored in the `/MC/tank/` directory, to the `DN1` VM.
# virt-xml DN1 --add-device --disk /MC/tank/Doc10.iso,device=cdrom
Domain ‘DN1’ defined successfully.

Verification

- Run the VM and test if the device is present and works as expected.

Additional resources

- The `man virt-xml` command
- Attaching devices to virtual machines

13.6.2. Replacing ISO images in virtual optical drives

To replace an ISO image attached as a virtual optical drive to a virtual machine (VM), edit the XML configuration file of the VM and specify the replacement.

Prerequisites

- You must store the ISO image on the local host.
- You must know the path to the ISO image.

Procedure

1. Locate the target device where the CD-ROM is attached to the VM. You can find this information in the VM’s XML configuration file.
   For example, the following command displays the DN1 VM’s XML configuration file, where the target device for CD-ROM is **sda**.

   ```bash
   # virsh dumpxml DN1
   ...
   <disk>
   ...
   <source file='/MC/tank/Doc10.iso'/>
   <target dev='sda' bus='sata'/>
   ...
   </disk>
   ...
   ```

2. Use the `virt-xml` utility with the **--edit** argument.
   For example, the following command replaces the Doc10 ISO image, attached to the DN1 VM at target **sda**, with the DrDN ISO image stored in the `/Dvrs/current/` directory.

   ```bash
   # virt-xml DN1 --edit target=sda --disk /Dvrs/current/DrDN.iso
   Domain ‘DN1’ defined successfully.
   ```

Verification

- Run the VM and test if the device is replaced and works as expected.

Additional resources
13.6.3. Removing ISO images from virtual optical drives

To remove an ISO image from a virtual optical drive attached to a virtual machine (VM), edit the XML configuration file of the VM.

Procedure

1. Locate the target device where the CD-ROM is attached to the VM. You can find this information in the VM’s XML configuration file.
   For example, the following command displays the `DN1` VM’s XML configuration file, where the target device for CD-ROM is `sda`.

   ```bash
   # virsh dumpxml DN1
   ...
   <disk>
   ...
   <source file='/Dvrs/current/DrDN'/>
   <target dev='sda' bus='sata'/>
   ...
   </disk>
   ...
   ``

2. Use the `virt-xml` utility with the `--edit` argument.
   For example, the following command removes the `DrDN` ISO image from the CD drive attached to the `DN1` VM.

   ```bash
   # virt-xml DN1 --edit target=sda --disk path=
   Domain ‘DN1’ defined successfully.
   ``

Verification

- Run the VM and check that image is no longer available.

Additional resources

- The `man virt-xml` command

13.6.4. Removing optical drives from virtual machines

To remove an optical drive attached to a virtual machine (VM), edit the XML configuration file of the VM.

Procedure

1. Locate the target device where the CD-ROM is attached to the VM. You can find this information in the VM’s XML configuration file.
   For example, the following command displays the `DN1` VM’s XML configuration file, where the target device for CD-ROM is `sda`.

   ```bash
   # virsh dumpxml DN1
   ...
   ```
2. Use the `virt-xml` utility with the `--remove-device` argument. For example, the following command removes the optical drive attached as target `sda` from the `DN1` VM.

```
# virt-xml DN1 --remove-device --disk target=sda
Domain 'DN1' defined successfully.
```

**Verification**

- Confirm that the device is no longer listed in the XML configuration file of the VM.

**Additional resources**

- The `man virt-xml` command

## 13.7. MANAGING SR-IOV DEVICES

An emulated virtual device often uses more CPU and memory than a hardware network device. This can limit the performance of a virtual machine (VM). However, if any devices on your virtualization host support Single Root I/O Virtualization (SR-IOV), you can use this feature to improve the device performance, and possibly also the overall performance of your VMs.

### 13.7.1. What is SR-IOV?

Single-root I/O virtualization (SR-IOV) is a specification that enables a single PCI Express (PCIe) device to present multiple separate PCI devices, called virtual functions (VFs), to the host system. Each of these devices:

- Is able to provide the same or similar service as the original PCIe device.
- Appears at a different address on the host PCI bus.
- Can be assigned to a different VM using VFIO assignment.

For example, a single SR-IOV capable network device can present VFs to multiple VMs. While all of the VFs use the same physical card, the same network connection, and the same network cable, each of the VMs directly controls its own hardware network device, and uses no extra resources from the host.

**How SR-IOV works**

The SR-IOV functionality is possible thanks to the introduction of the following PCIe functions:

- **Physical functions (PFs)** - A PCIe function that provides the functionality of its device (for example networking) to the host, but can also create and manage a set of VFs. Each SR-IOV capable device has one or more PFs.

- **Virtual functions (VFs)** - Lightweight PCIe functions that behave as independent devices. Each VF is derived from a PF. The maximum number of VFs a device can have depends on the device
hardware. Each VF can be assigned only to a single VM at a time, but a VM can have multiple
VFs assigned to it.

VMs recognize VFs as virtual devices. For example, a VF created by an SR-IOV network device appears
as a network card to a VM to which it is assigned, in the same way as a physical network card appears to
the host system.

Figure 13.1. SR-IOV architecture

Benefits
The primary advantages of using SR-IOV VFs rather than emulated devices are:

- Improved performance
- Reduced use of host CPU and memory resources

For example, a VF attached to a VM as a vNIC performs at almost the same level as a physical NIC, and
much better than paravirtualized or emulated NICs. In particular, when multiple VFs are used
simultaneously on a single host, the performance benefits can be significant.

Disadvantages

- To modify the configuration of a PF, you must first change the number of VFs exposed by the
  PF to zero. Therefore, you also need to remove the devices provided by these VFs from the VM
to which they are assigned.

- A VM with an VFIO-assigned devices attached, including SR-IOV VFs, cannot be migrated to
  another host. In some cases, you can work around this limitation by pairing the assigned device
  with an emulated device. For example, you can bond an assigned networking VF to an emulated
In addition, VFIO-assigned devices require pinning of VM memory, which increases the memory consumption of the VM and prevents the use of memory ballooning on the VM.

Additional resources
- Supported devices for SR-IOV assignment

13.7.2. Attaching SR-IOV networking devices to virtual machines

To attach an SR-IOV networking device to a virtual machine (VM) on an Intel or AMD host, you must create a virtual function (VF) from an SR-IOV capable network interface on the host and assign the VF as a device to a specified VM. For details, see the following instructions.

Prerequisites
- The CPU and the firmware of your host support the I/O Memory Management Unit (IOMMU).
  - If using an Intel CPU, it must support the Intel Virtualization Technology for Directed I/O (VT-d).
  - If using an AMD CPU, it must support the AMD-Vi feature.
- The host system uses Access Control Service (ACS) to provide direct memory access (DMA) isolation for PCIe topology. Verify this with the system vendor. For additional information, see Hardware Considerations for Implementing SR-IOV.
- The physical network device supports SR-IOV. To verify if any network devices on your system support SR-IOV, use the `lspci -v` command and look for Single Root I/O Virtualization (SR-IOV) in the output.

```bash
# lspci -v
[..]
02:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
Subsystem: Intel Corporation Gigabit ET Dual Port Server Adapter
Flags: bus master, fast devsel, latency 0, IRQ 16, NUMA node 0
Memory at fcba0000 (32-bit, non-prefetchable) [size=128K]
[..]
Capabilities: [150] Alternative Routing-ID Interpretation (ARI)
Capabilities: [160] Single Root I/O Virtualization (SR-IOV)
Kernel driver in use: igb
Kernel modules: igb
[..]
```

- The host network interface you want to use for creating VFs is running. For example, to activate the `eth1` interface and verify it is running:

```bash
# ip link set eth1 up
# ip link show eth1
8: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT qlen 1000
link/ether a0:36:9f:8f:3f:b8 brd ff:ff:ff:ff:ff:ff
vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
```
For SR-IOV device assignment to work, the IOMMU feature must be enabled in the host BIOS and kernel. To do so:

- On an Intel host, enable VT-d:
  - If your Intel host uses multiple boot entries:
    - A. Edit the `/etc/default/grub` file and add the `intel_iommu=on` and `iommu=pt` parameters at the end of the `GRUB_CMDLINE_LINUX` line:
      ```
      GRUB_CMDLINE_LINUX="crashkernel=auto resume=/dev/mapper/rhel_dell-per730-27-swap rd.lvm.lv=rhel_dell-per730-27/root rd.lvm.lv=rhel_dell-per730-27/swap console=ttyS0,115200n81 intel_iommu=on iommu=pt"
      ```
    - B. Regenerate the GRUB configuration:
      ```
      # grub2-mkconfig -o /boot/grub2/grub.cfg
      ```
      C. Reboot the host.
  - If your Intel host uses a single boot entry:
    - A. Regenerate the GRUB configuration with the `intel_iommu=on iommu=pt` parameter:
      ```
      # grubby --args="intel_iommu=on iommu=pt" --update-kernel DEFAULT
      ```
      B. Reboot the host.
- On an AMD host, enable AMD-Vi:
  - If your AMD host uses multiple boot entries:
    - A. Edit the `/etc/default/grub` file and add the `iommu=pt` parameter at the end of the `GRUB_CMDLINE_LINUX` line:
      ```
      GRUB_CMDLINE_LINUX="crashkernel=auto resume=/dev/mapper/rhel_dell-per730-27-swap rd.lvm.lv=rhel_dell-per730-27/root rd.lvm.lv=rhel_dell-per730-27/swap console=ttyS0,115200n81 iommu=pt"
      ```
    - B. Regenerate the GRUB configuration:
      ```
      # grub2-mkconfig -o /boot/grub2/grub.cfg
      ```
    - C. Reboot the host.
  - If your AMD host uses a single boot entry:
    - A. Regenerate the GRUB configuration with the `iommu=pt` parameter:
      ```
      # grubby --args="iommu=pt" --update-kernel DEFAULT
      ```
B. Reboot the host.

Procedure

1. Optional: Confirm the maximum number of VFs your network device can use. To do so, use the following command and replace eth1 with your SR-IOV compatible network device.

   ```
   # cat /sys/class/net/eth1/device/sriov_totalvfs
   7
   ```

2. Use the following command to create a virtual function (VF):

   ```
   # echo VF-number > /sys/class/net/network-interface/device/sriov_numvfs
   ```

   In the command, replace:
   - `VF-number` with the number of VFs you want to create on the PF.
   - `network-interface` with the name of the network interface for which the VFs will be created.

   The following example creates 2 VFs from the eth1 network interface:

   ```
   # echo 2 > /sys/class/net/eth1/device/sriov_numvfs
   ```

3. Verify the VFs have been added:

   ```
   # lspci | grep Ethernet
   82:00.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)
   82:00.1 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)
   82:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
   82:10.2 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
   ```

4. Make the created VFs persistent by creating a udev rule for the network interface you used to create the VFs. For example, for the eth1 interface, create the `/etc/udev/rules.d/eth1.rules` file, and add the following line:

   ```
   ACTION=="add", SUBSYSTEM=="net", ENV{ID_NET_DRIVER}=="ixgbe",
   ATTR{device/sriov_numvfs}="2"
   ```

   This ensures that the two VFs that use the **ixgbe** driver will automatically be available for the **eth1** interface when the host starts. If you do not require persistent SR-IOV devices, skip this step.
Currently, the setting described above does not work correctly when attempting to make VFs persistent on Broadcom NetXtreme II BCM57810 adapters. In addition, attaching VFs based on these adapters to Windows VMs is currently not reliable.

5. Hot-plug one of the newly added VF interface devices to a running VM.

```
# virsh attach-interface testguest1 hostdev 0000:82:10.0 --managed --live --config
```

Verification

- If the procedure is successful, the guest operating system detects a new network interface card.

13.7.3. Supported devices for SR-IOV assignment

Not all devices can be used for SR-IOV. The following devices have been tested and verified as compatible with SR-IOV in RHEL 9.

**Networking devices**

- Intel 82599ES 10 Gigabit Ethernet Controller - uses the `ixgbe` driver
- Intel Ethernet Controller XL710 Series - uses the `i40e` driver
- Mellanox ConnectX-5 Ethernet Adapter Cards - use the `mlx5_core` driver
- Intel Ethernet Network Adapter XXV710 - uses the `i40e` driver
- Intel 82576 Gigabit Ethernet Controller - uses the `igb` driver
- Broadcom NetXtreme II BCM57810 - uses the `bnx2x` driver

13.8. ATTACHING DASD DEVICES TO VIRTUAL MACHINES ON IBM Z

Using the `vfio-ccw` feature, you can assign direct-access storage devices (DASDs) as mediated devices to your virtual machines (VMs) on IBM Z hosts. This for example makes it possible for the VM to access a z/OS dataset, or to provide the assigned DASDs to a z/OS machine.

**Prerequisites**

- Your host system is using the IBM Z hardware architecture and supports the FICON protocol.
- The target VM is using a Linux guest operating system.
- The `mdevctl` package is installed.

```
# dnf install mdevctl
```
The `driverctl` package is installed.

```bash
# dnf install driverctl
```

The necessary kernel modules have been loaded on the host. To verify, use:

```bash
# lsmod | grep vfio
```

The output should contain the following modules:

- `vfio_ccw`
- `vfio_mdev`
- `vfio_iommu_type1`

You have a spare DASD device for exclusive use by the VM, and you know the device’s identifier.

This procedure uses `0.0.002c` as an example. When performing the commands, replace `0.0.002c` with the identifier of your DASD device.

### Procedure

1. Obtain the subchannel identifier of the DASD device.

   ```bash
   # lscss -d 0.0.002c
   ``
   
   
<table>
<thead>
<tr>
<th>Device</th>
<th>Subchan.</th>
<th>DevType CU Type Use</th>
<th>PIM</th>
<th>PAM</th>
<th>POM</th>
<th>CHPIDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.002c</td>
<td>0.0.29a8</td>
<td>3390/0c 3990/e9 yes</td>
<td>f0</td>
<td>f0</td>
<td>ff</td>
<td>02112210000000</td>
</tr>
</tbody>
</table>
   
   In this example, the subchannel identifier is detected as `0.0.29a8`. In the following commands of this procedure, replace `0.0.29a8` with the detected subchannel identifier of your device.

2. If the `lscss` command in the previous step only displayed the header output and no device information, perform the following steps:

   a. Remove the device from the `cio Ignore` list.

      ```bash
      # cio Ignore -r 0.0.002c
      ```

   b. In the guest OS, edit the kernel command line of the VM and add the device identifier with a `!` mark to the line that starts with `cio Ignore=`, if it is not present already.

      ```bash
      cio Ignore=all,!condev,!0.0.002c
      ```

   c. Repeat step 1 on the host to obtain the subchannel identifier.

3. Bind the subchannel to the `vfio_ccw` passthrough driver.

   ```bash
   # driverctl -b css set-override 0.0.29a8 vfio_ccw
   ```
NOTE
This binds the 0.0.29a8 subchannel to \texttt{vfio_ccw} persistently, which means the DASD will not be usable on the host. If you need to use the device on the host, you must first remove the automatic binding to ‘vfio_ccw’ and rebind the subchannel to the default driver:

```bash
# driverctl -b css unset-override 0.0.29a8
```

4. Generate an UUID.

```bash
# uuidgen
30820a6f-b1a5-4503-91ca-0c10ba12345a
```

5. Create the DASD mediated device using the generated UUID.

```bash
# mdevctl start --uuid 30820a6f-b1a5-4503-91ca-0c10ba12345a --parent 0.0.29a8 --type vfio_ccw-io
```

6. Make the mediated device persistent.

```bash
# mdevctl define --auto --uuid 30820a6f-b1a5-4503-91ca-0c10ba12345a
```

7. Shut down the VM, if it is running.

8. Attach the mediated device to the VM. To do so, use the \texttt{virsh edit} utility to edit the XML configuration of the VM, add the following section to the XML, and replace the \texttt{uuid} value with the UUID you generated in the previous step.

```xml
<hostdev mode='subsystem' type='mdev' model='vfio-ccw'>
  <source>
    <uuid>30820a6f-b1a5-4503-91ca-0c10ba12345a</uuid>
  </source>
</hostdev>
```

Verification

1. Obtain the identifier that \texttt{libvirt} assigned to the mediated DASD device. To do so, display the XML configuration of the VM and look for a \texttt{vfio-ccw} device.

```bash
# virsh dumpxml vm-name
```

```xml
<domain>
  [...]  
  <hostdev mode='subsystem' type='mdev' managed='no' model='vfio-ccw'>
    <source>
      <address uuid='10620d2f-ed4d-437b-8aff-beda461541f9'/>
    </source>
    <alias name='hostdev0'/>
    <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0009'/>
  </hostdev>
  [...]  
</domain>
```
In this example, the assigned identifier of the device is 0.0.0009.

2. Start the VM and log in to its guest OS.

3. In the guest OS, confirm that the DASD device is listed. For example:

   ```bash
   # lscss | grep 0.0.0009
   0.0.0009 0.0.0007 3390/0c 3990/e9  f0 f0 ff 12212231 00000000
   ```

4. In the guest OS, set the device online. For example:

   ```bash
   # chccwdev -e 0.0009
   Setting device 0.0.0009 online
   Done
   ```

Additional resources

- IBM documentation on `cio_ignore`
- Configuring kernel parameters at runtime
CHAPTER 14. MANAGING STORAGE FOR VIRTUAL MACHINES

A virtual machine (VM), just like a physical machine, requires storage for data, program, and system files. As a VM administrator, you can assign physical or network-based storage to your VMs as virtual storage. You can also modify how the storage is presented to a VM regardless of the underlying hardware.

The following sections provide information about the different types of VM storage, how they work, and how you can manage them using the CLI or the web console.

14.1. UNDERSTANDING VIRTUAL MACHINE STORAGE

If you are new to virtual machine (VM) storage, or are unsure about how it works, the following sections provide a general overview about the various components of VM storage, how it works, management basics, and the supported solutions provided by Red Hat.

You can find information about:

- Storage pools
- Storage volumes
- Managing storage using libvirt
- Overview of VM storage
- Supported and unsupported storage pool types

14.1.1. Introduction to storage pools

A storage pool is a file, directory, or storage device, managed by libvirt to provide storage for virtual machines (VMs). You can divide storage pools into storage volumes, which store VM images or are attached to VMs as additional storage.

Furthermore, multiple VMs can share the same storage pool, allowing for better allocation of storage resources.

- Storage pools can be persistent or transient:
  - A persistent storage pool survives a system restart of the host machine. You can use the `virsh pool-define` command to create a persistent storage pool.
  - A transient storage pool only exists until the host reboots. You can use the `virsh pool-create` command to create a transient storage pool.

Storage pool storage types

Storage pools can be either local or network-based (shared):

- **Local storage pools**
  Local storage pools are attached directly to the host server. They include local directories, directly attached disks, physical partitions, and Logical Volume Management (LVM) volume groups on local devices.

  Local storage pools are useful for development, testing, and small deployments that do not require migration or have a large number of VMs.
Networked (shared) storage pools

Networked storage pools include storage devices shared over a network using standard protocols.

14.1.2. Introduction to storage volumes

Storage pools are divided into storage volumes. Storage volumes are abstractions of physical partitions, LVM logical volumes, file-based disk images, and other storage types handled by libvirt. Storage volumes are presented to VMs as local storage devices, such as disks, regardless of the underlying hardware.

On the host machine, a storage volume is referred to by its name and an identifier for the storage pool from which it derives. On the virsh command line, this takes the form --pool storage_pool volume_name.

For example, to display information about a volume named firstimage in the guest_images pool.

```
# virsh vol-info --pool guest_images firstimage
Name:             firstimage
Type:             block
Capacity:         20.00 GB
Allocation:       20.00 GB
```

14.1.3. Storage management using libvirt

Using the libvirt remote protocol, you can manage all aspects of VM storage. These operations can also be performed on a remote host. Consequently, a management application that uses libvirt, such as the RHEL web console, can be used to perform all the required tasks of configuring the storage of a VM.

You can use the libvirt API to query the list of volumes in a storage pool or to get information regarding the capacity, allocation, and available storage in that storage pool. For storage pools that support it, you can also use the libvirt API to create, clone, resize, and delete storage volumes. Furthermore, you can use the libvirt API to upload data to storage volumes, download data from storage volumes, or wipe data from storage volumes.

14.1.4. Overview of storage management

To illustrate the available options for managing storage, the following example talks about a sample NFS server that uses mount -t nfs nfs.example.com:/path/to/share /path/to/data.

As a storage administrator:

- You can define an NFS storage pool on the virtualization host to describe the exported server path and the client target path. Consequently, libvirt can mount the storage either automatically when libvirt is started or as needed while libvirt is running.

- You can simply add the storage pool and storage volume to a VM by name. You do not need to add the target path to the volume. Therefore, even if the target client path changes, it does not affect the VM.

- You can configure storage pools to autostart. When you do so, libvirt automatically mounts the NFS shared disk on the directory which is specified when libvirt is started. libvirt mounts the share on the specified directory, similar to the command mount nfs.example.com:/path/to/share /vmda.
You can query the storage volume paths using the libvirt API. These storage volumes are basically the files present in the NFS shared disk. You can then copy these paths into the section of a VM’s XML definition that describes the source storage for the VM’s block devices.

In the case of NFS, you can use an application that uses the libvirt API to create and delete storage volumes in the storage pool (files in the NFS share) up to the limit of the size of the pool (the storage capacity of the share).

Note that, not all storage pool types support creating and deleting volumes.

You can stop a storage pool when no longer required. Stopping a storage pool (pool-destroy) undoes the start operation, in this case, unmounting the NFS share. The data on the share is not modified by the destroy operation, despite what the name of the command suggests. For more information, see man virsh.

14.1.5. Supported and unsupported storage pool types

Supported storage pool types

The following is a list of storage pool types supported by RHEL:

- Directory-based storage pools
- Disk-based storage pools
- Partition-based storage pools
- iSCSI-based storage pools
- LVM-based storage pools
- NFS-based storage pools
- SCSI-based storage pools with vHBA devices
- Multipath-based storage pools
- RBD-based storage pools

Unsupported storage pool types

The following is a list of libvirt storage pool types not supported by RHEL:

- Sheepdog-based storage pools
- Vstorage-based storage pools
- ZFS-based storage pools
- iSCSI-direct storage pools
- GlusterFS storage pools

14.2. MANAGING VIRTUAL MACHINE STORAGE POOLS USING THE CLI

You can use the CLI to manage the following aspects of your storage pools to assign storage to your virtual machines (VMs):
- View storage pool information
- Create storage pools
  - Create directory-based storage pools using the CLI
  - Create disk-based storage pools using the CLI
  - Create filesystem-based storage pools using the CLI
  - Create iSCSI-based storage pools using the CLI
  - Create LVM-based storage pools using the CLI
  - Create NFS-based storage pools using the CLI
  - Create SCSI-based storage pools with vHBA devices using the CLI
- Remove storage pools

14.2.1. Viewing storage pool information using the CLI

Using the CLI, you can view a list of all storage pools with limited or full details about the storage pools. You can also filter the storage pools listed.

**Procedure**

- Use the `virsh pool-list` command to view storage pool information.

  ```bash
  # virsh pool-list --all --details
  Name                State    Autostart  Persistent  Capacity  Allocation  Available
  default              running  yes      yes          48.97 GiB   23.93 GiB  25.03 GiB
  Downloads              running  yes      yes         175.62 GiB   62.02 GiB  113.60 GiB
  RHEL-Storage-Pool       running  yes      yes         214.62 GiB   93.02 GiB  168.60 GiB
  ```

**Additional resources**

- The `virsh pool-list --help` command

14.2.2. Creating directory-based storage pools using the CLI

A directory-based storage pool is based on a directory in an existing mounted file system. This is useful, for example, when you want to use the remaining space on the file system for other purposes. You can use the `virsh` utility to create directory-based storage pools.

**Prerequisites**

- Ensure your hypervisor supports directory storage pools:

  ```bash
  # virsh pool-capabilities | grep "dir supported='yes'"
  ```

  If the command displays any output, directory pools are supported.

**Procedure**
1. Create a storage pool
Use the `virsh pool-define-as` command to define and create a directory-type storage pool. For example, to create a storage pool named `guest_images_dir` that uses the `/guest_images` directory:

```
# virsh pool-define-as guest_images_dir dir --target "/guest_images"
Pool guest_images_dir defined
```

If you already have an XML configuration of the storage pool you want to create, you can also define the pool based on the XML. For details, see Directory-based storage pool parameters.

2. Create the storage pool target path
Use the `virsh pool-build` command to create a storage pool target path for a pre-formatted file system storage pool, initialize the storage source device, and define the format of the data.

```
# virsh pool-build guest_images_dir
Pool guest_images_dir built
```

```
# ls -la /guest_images
 total 8
 drwx------.  2 root root 4096 May 31 19:38 .
 dr-xr-xr-x. 25 root root 4096 May 31 19:38 ..
```

3. Verify that the pool was created
Use the `virsh pool-list` command to verify that the pool was created.

```
# virsh pool-list --all
Name                State      Autostart
-----------------------------------------
default              active     yes
guest_images_dir     inactive   no
```

4. Start the storage pool
Use the `virsh pool-start` command to mount the storage pool.

```
# virsh pool-start guest_images_dir
Pool guest_images_dir started
```

NOTE
The `virsh pool-start` command is only necessary for persistent storage pools. Transient storage pools are automatically started when they are created.

5. [Optional] Turn on autostart
By default, a storage pool defined with the `virsh` command is not set to automatically start each time virtualization services start. Use the `virsh pool-autostart` command to configure the storage pool to autostart.

```
# virsh pool-autostart guest_images_dir
Pool guest_images_dir marked as autostarted
```
Verification

- Use the `virsh pool-info` command to verify that the storage pool is in the **running** state. Check if the sizes reported are as expected and if autostart is configured correctly.

```bash
# virsh pool-info guest_images_dir
Name:           guest_images_dir
UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
State:          running
Persistent:     yes
Autostart:      yes
Capacity:       458.39 GB
Allocation:     197.91 MB
Available:      458.20 GB
```

14.2.3. Creating disk-based storage pools using the CLI

In a disk-based storage pool, the pool is based on a disk partition. This is useful, for example, when you want to have an entire disk partition dedicated as virtual machine (VM) storage. You can use the `virsh` utility to create disk-based storage pools.

Prerequisites

- Ensure your hypervisor supports disk-based storage pools:

  ```bash
  # virsh pool-capabilities | grep "'disk' supported='yes'"
  ```

  If the command displays any output, disk-based pools are supported.

- Prepare a device on which you will base the storage pool. For this purpose, prefer partitions (for example, `/dev/sdb1`) or LVM volumes. If you provide a VM with write access to an entire disk or block device (for example, `/dev/sdb`), the VM will likely partition it or create its own LVM groups on it. This can result in system errors on the host.

  However, if you require using an entire block device for the storage pool, Red Hat recommends protecting any important partitions on the device from GRUB’s `os-prober` function. To do so, edit the `/etc/default/grub` file and apply one of the following configurations:

  - Disable `os-prober`.

    ```bash
    GRUB_DISABLE_OS_PROBER=true
    ```

  - Prevent `os-prober` from discovering a specific partition. For example:

    ```bash
    GRUB_OS_PROBER_SKIP_LIST="5ef6313a-257c-4d43@/dev/sdb1"
    ```

- Back up any data on the selected storage device before creating a storage pool. Depending on the version of `libvirt` being used, dedicating a disk to a storage pool may reformat and erase all data currently stored on the disk device.

Procedure

1. Create a storage pool
Use the `virsh pool-define-as` command to define and create a disk-type storage pool. The following example creates a storage pool named `guest_images_disk` that uses the `/dev/sdb` device and is mounted on the `/dev` directory.

```
# virsh pool-define-as guest_images_disk disk --source-format=gpt --source-dev=/dev/sdb --target /dev
Pool guest_images_disk defined
```

If you already have an XML configuration of the storage pool you want to create, you can also define the pool based on the XML. For details, see Disk-based storage pool parameters.

2. Create the storage pool target path

   Use the `virsh pool-build` command to create a storage pool target path for a pre-formatted file-system storage pool, initialize the storage source device, and define the format of the data.

   ```
   # virsh pool-build guest_images_disk
   Pool guest_images_disk built
   ```

   **NOTE**

   Building the target path is only necessary for disk-based, file system-based, and logical storage pools. If `libvirt` detects that the source storage device's data format differs from the selected storage pool type, the build fails, unless the `overwrite` option is specified.

3. Verify that the pool was created

   Use the `virsh pool-list` command to verify that the pool was created.

   ```
   # virsh pool-list --all
   Name                 State      Autostart
   -----------------------------------------
   default              active     yes
   guest_images_disk    inactive   no
   ```

4. Start the storage pool

   Use the `virsh pool-start` command to mount the storage pool.

   ```
   # virsh pool-start guest_images_disk
   Pool guest_images_disk started
   ```

   **NOTE**

   The `virsh pool-start` command is only necessary for persistent storage pools. Transient storage pools are automatically started when they are created.

5. [Optional] Turn on autostart

   By default, a storage pool defined with the `virsh` command is not set to automatically start each time virtualization services start. Use the `virsh pool-autostart` command to configure the storage pool to autostart.
# virsh pool-autostart guest_images_disk
Pool guest_images_disk marked as autostarted

Verification

- Use the `virsh pool-info` command to verify that the storage pool is in the `running` state. Check if the sizes reported are as expected and if autostart is configured correctly.

```bash
# virsh pool-info guest_images_disk
Name:           guest_images_disk
UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
State:          running
Persistent:     yes
Autostart:      yes
Capacity:       458.39 GB
Allocation:     197.91 MB
Available:      458.20 GB
```

14.2.4. Creating filesystem-based storage pools using the CLI

When you want to create a storage pool on a file system that is not mounted, use the filesystem-based storage pool. This storage pool is based on a given file-system mountpoint. You can use the `virsh` utility to create filesystem-based storage pools.

Prerequisites

- Ensure your hypervisor supports filesystem-based storage pools:

```bash
# virsh pool-capabilities | grep "'fs' supported='yes'"
```

If the command displays any output, file-based pools are supported.

- Prepare a device on which you will base the storage pool. For this purpose, prefer partitions (for example, `/dev/sdb1`) or LVM volumes. If you provide a VM with write access to an entire disk or block device (for example, `/dev/sdb`), the VM will likely partition it or create its own LVM groups on it. This can result in system errors on the host. However, if you require using an entire block device for the storage pool, Red Hat recommends protecting any important partitions on the device from GRUB’s `os-prober` function. To do so, edit the `/etc/default/grub` file and apply one of the following configurations:

  - Disable `os-prober`.

    ```bash
    GRUB_DISABLE_OS_PROBER=true
    ```

  - Prevent `os-prober` from discovering a specific partition. For example:

    ```bash
    GRUB_OS_PROBER_SKIP_LIST="5ef6313a-257c-4d43@/dev/sdb1"
    ```

Procedure

1. Create a storage pool
Use the `virsh pool-define-as` command to define and create a filesystem-type storage pool. For example, to create a storage pool named `guest_images_fs` that uses the `/dev/sdc1` partition, and is mounted on the `/guest_images` directory:

```
# virsh pool-define-as guest_images_fs fs --source-dev /dev/sdc1 --target /guest_images
Pool guest_images_fs defined
```

If you already have an XML configuration of the storage pool you want to create, you can also define the pool based on the XML. For details, see `Filesystem-based storage pool parameters`.

2. **Define the storage pool target path**
   Use the `virsh pool-build` command to create a storage pool target path for a pre-formatted file-system storage pool, initialize the storage source device, and define the format of the data.

```
# virsh pool-build guest_images_fs
Pool guest_images_fs built
```

```
# ls -la /guest_images
total 8
  drwx------.  2 root root 4096 May 31 19:38 .
  dr-xr-xr-x. 25 root root 4096 May 31 19:38 ..
```

3. **Verify that the pool was created**
   Use the `virsh pool-list` command to verify that the pool was created.

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images_fs      inactive   no
```

4. **Start the storage pool**
   Use the `virsh pool-start` command to mount the storage pool.

```
# virsh pool-start guest_images_fs
Pool guest_images_fs started
```

```
NOTE
The `virsh pool-start` command is only necessary for persistent storage pools. Transient storage pools are automatically started when they are created.
```

5. **[Optional] Turn on autostart**
   By default, a storage pool defined with the `virsh` command is not set to automatically start each time virtualization services start. Use the `virsh pool-autostart` command to configure the storage pool to autostart.

```
# virsh pool-autostart guest_images_fs
Pool guest_images_fs marked as autostarted
```
Verification

1. Use the `virsh pool-info` command to verify that the storage pool is in the **running** state. Check if the sizes reported are as expected and if autostart is configured correctly.

   ```
   # virsh pool-info guest_images_fs
   Name:           guest_images_fs
   UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
   State:          running
   Persistent:     yes
   Autostart:      yes
   Capacity:       458.39 GB
   Allocation:     197.91 MB
   Available:      458.20 GB
   ```

2. Verify there is a **lost+found** directory in the target path on the file system, indicating that the device is mounted.

   ```
   # mount | grep /guest_images
   /dev/sdc1 on /guest_images type ext4 (rw)
   # ls -la /guest_images
   total 24
   drwxr-xr-x.  3 root root  4096 May 31 19:47 .
   dr-xr-xr-x. 25 root root  4096 May 31 19:38 ..
   drwx------.  2 root root 16384 May 31 14:18 lost+found
   ```

14.2.5. Creating iSCSI-based storage pools using the CLI

Internet Small Computer Systems Interface (iSCSI) is an IP-based storage networking standard for linking data storage facilities. If you want to have a storage pool on an iSCSI server, you can use the `virsh` utility to create iSCSI-based storage pools.

Prerequisites

- Ensure your hypervisor supports iSCSI-based storage pools:

  ```
  # virsh pool-capabilities | grep "'iscsi' supported='yes'"
  ```

  If the command displays any output, iSCSI-based pools are supported.

Procedure

1. Create a storage pool

   Use the `virsh pool-define-as` command to define and create an iSCSI-type storage pool. For example, to create a storage pool named `guest_images_iscsi` that uses the `iqn.2010-05.com.example.server1:iscsi` IQN on the `server1.example.com`, and is mounted on the `/dev/disk/by-path` path:

   ```
   # virsh pool-define-as --name guest_images_iscsi --type iscsi --source-host server1.example.com --source-dev iqn.2010-05.com.example.server1:iscsi --target /dev/disk/by-path
   Pool guest_images_iscsi defined
   ```
If you already have an XML configuration of the storage pool you want to create, you can also define the pool based on the XML. For details, see iSCSI-based storage pool parameters.

2. **Verify that the pool was created**
   Use the `virsh pool-list` command to verify that the pool was created.

   ```
   # virsh pool-list --all
   Name               State  Autostart
   -----------------------------------------
   default            active  yes
   guest_images_iscsi inactive no
   ```

3. **Start the storage pool**
   Use the `virsh pool-start` command to mount the storage pool.

   ```
   # virsh pool-start guest_images_iscsi
   Pool guest_images_iscsi started
   ```

   **NOTE**
   The `virsh pool-start` command is only necessary for persistent storage pools. Transient storage pools are automatically started when they are created.

4. **[Optional] Turn on autostart**
   By default, a storage pool defined with the `virsh` command is not set to automatically start each time virtualization services start. Use the `virsh pool-autostart` command to configure the storage pool to autostart.

   ```
   # virsh pool-autostart guest_images_iscsi
   Pool guest_images_iscsi marked as autostarted
   ```

**Verification**

- Use the `virsh pool-info` command to verify that the storage pool is in the `running` state. Check if the sizes reported are as expected and if autostart is configured correctly.

   ```
   # virsh pool-info guest_images_iscsi
   Name:           guest_images_iscsi
   UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
   State:          running
   Persistent:     yes
   Autostart:      yes
   Capacity:       458.39 GB
   Allocation:     197.91 MB
   Available:      458.20 GB
   ```

### 14.2.6. Creating LVM-based storage pools using the CLI

If you want to have a storage pool that is part of an LVM volume group, you can use the `virsh` utility to create LVM-based storage pools.
**Recommendations**

Be aware of the following before creating an LVM-based storage pool:

- LVM-based storage pools do not provide the full flexibility of LVM.
- **libvirt** supports thin logical volumes, but does not provide the features of thin storage pools.
- LVM-based storage pools are volume groups. You can create volume groups using the **virsh** utility, but this way you can only have one device in the created volume group. To create a volume group with multiple devices, use the LVM utility instead, see [How to create a volume group in Linux with LVM](https://docs.rhcn.redhat.com/rhel/9/html/enterprise-linux-lvm-guide/ch-lvm-volume-groups.html).
  
  For more detailed information about volume groups, refer to the **Red Hat Enterprise Linux Logical Volume Manager Administration Guide**.
- LVM-based storage pools require a full disk partition. If you activate a new partition or device using **virsh** commands, the partition will be formatted and all data will be erased. If you are using a host’s existing volume group, as in these procedures, nothing will be erased.

**Prerequisites**

- Ensure your hypervisor supports LVM-based storage pools:

  ```
  # virsh pool-capabilities | grep "'logical' supported='yes'"
  ```

  If the command displays any output, LVM-based pools are supported.

**Procedure**

1. **Create a storage pool**

   Use the **virsh pool-define-as** command to define and create an LVM-type storage pool. For example, the following command creates a storage pool named `guest_images_lvm` that uses the `lvm_vg` volume group and is mounted on the `/dev/lvm_vg` directory:

   ```
   # virsh pool-define-as guest_images_lvm logical --source-name lvm_vg --target /dev/lvm_vg
   Pool guest_images_lvm defined
   ```

   If you already have an XML configuration of the storage pool you want to create, you can also define the pool based on the XML. For details, see [LVM-based storage pool parameters](https://docs.rhcn.redhat.com/rhel/9/html/enterprise-linux-lvm-guide/ch-lvm-storage-pool-parameters.html).

2. **Verify that the pool was created**

   Use the **virsh pool-list** command to verify that the pool was created.

   ```
   # virsh pool-list --all
   Name                State    Autostart
   -----------------------------------
   default             active    yes
   guest_images_lvm    inactive  no
   ```

3. **Start the storage pool**

   Use the **virsh pool-start** command to mount the storage pool.
# virsh pool-start guest_images_lvm
Pool guest_images_lvm started

**NOTE**

The `virsh pool-start` command is only necessary for persistent storage pools. Transient storage pools are automatically started when they are created.

4. **[Optional] Turn on autostart**
   
   By default, a storage pool defined with the `virsh` command is not set to automatically start each time virtualization services start. Use the `virsh pool-autostart` command to configure the storage pool to autostart.

   ```bash
   # virsh pool-autostart guest_images_lvm
   Pool guest_images_lvm marked as autostarted
   ```

**Verification**

- Use the `virsh pool-info` command to verify that the storage pool is in the **running** state. Check if the sizes reported are as expected and if autostart is configured correctly.

   ```bash
   # virsh pool-info guest_images_lvm
   Name:           guest_images_lvm
   UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
   State:          running
   Persistent:     yes
   Autostart:      yes
   Capacity:       458.39 GB
   Allocation:     197.91 MB
   Available:      458.20 GB
   ```

### 14.2.7. Creating NFS-based storage pools using the CLI

If you want to have a storage pool on a Network File System (NFS) server, you can use the `virsh` utility to create NFS-based storage pools.

**Prerequisites**

- Ensure your hypervisor supports NFS-based storage pools:

   ```bash
   # virsh pool-capabilities | grep "<value>nfs</value>"
   ```

   If the command displays any output, NFS-based pools are supported.

**Procedure**

1. **Create a storage pool**
   
   Use the `virsh pool-define-as` command to define and create an NFS-type storage pool. For example, to create a storage pool named `guest_images_netfs` that uses a NFS server with IP `111.222.111.222` mounted on the server directory `/home/net_mount` using the target directory `/var/lib/libvirt/images/nfspool:`
If you already have an XML configuration of the storage pool you want to create, you can also define the pool based on the XML. For details, see NFS-based storage pool parameters.

2. Verify that the pool was created
   Use the `virsh pool-list` command to verify that the pool was created.

   ```bash
   # virsh pool-list --all
   Name                  State  Autostart
   -----------------------------------------
   default               active yes
   guest_images_netfs    inactive no
   ```

3. Start the storage pool
   Use the `virsh pool-start` command to mount the storage pool.

   ```bash
   # virsh pool-start guest_images_netfs
   Pool guest_images_netfs started
   ```

   **NOTE**
   The `virsh pool-start` command is only necessary for persistent storage pools. Transient storage pools are automatically started when they are created.

4. [Optional] Turn on autostart
   By default, a storage pool defined with the `virsh` command is not set to automatically start each time virtualization services start. Use the `virsh pool-autostart` command to configure the storage pool to autostart.

   ```bash
   # virsh pool-autostart guest_images_netfs
   Pool guest_images_netfs marked as autostarted
   ```

Verification

- Use the `virsh pool-info` command to verify that the storage pool is in the `running` state. Check if the sizes reported are as expected and if autostart is configured correctly.

   ```bash
   # virsh pool-info guest_images_netfs
   Name: guest_images_netfs
   UUID: c7466869-e82a-a66c-2187-dc9d6f0877d0
   State: running
   Persistent: yes
   Autostart: yes
   Capacity: 458.39 GB
   Allocation: 197.91 MB
   Available: 458.20 GB
   ```

14.2.8. Creating SCSI-based storage pools with vHBA devices using the CLI
If you want to have a storage pool on a Small Computer System Interface (SCSI) device, your host must be able to connect to the SCSI device using a virtual host bus adapter (vHBA). You can then use the `virsh` utility to create SCSI-based storage pools.

**Prerequisites**

- Ensure your hypervisor supports SCSI-based storage pools:

  ```
  # virsh pool-capabilities | grep "'scsi' supported='yes'"
  ```

  If the command displays any output, SCSI-based pools are supported.

- Before creating a SCSI-based storage pools with vHBA devices, create a vHBA. For more information, see [Creating vHBAs](#).

**Procedure**

1. **Create a storage pool**

   Use the `virsh pool-define-as` command to define and create SCSI storage pool using a vHBA. For example, the following creates a storage pool named `guest_images_vhba` that uses a vHBA identified by the `scsi_host3` parent adapter, world-wide port number `5001a4ace3ee047d`, and world-wide node number `5001a4a93526d0a1`. The storage pool is mounted on the `/dev/disk/` directory:

   ```
   # virsh pool-define-as guest_images_vhba scsi --adapter-parent scsi_host3 --adapter-wwnn 5001a4ace3ee047d --adapter-wwpn 5001a4a93526d0a1 --target /dev/disk/
   Pool guest_images_vhba defined
   ```

   If you already have an XML configuration of the storage pool you want to create, you can also define the pool based on the XML. For details, see [Parameters for SCSI-based storage pools with vHBA devices](#).

2. **Verify that the pool was created**

   Use the `virsh pool-list` command to verify that the pool was created.

   ```
   # virsh pool-list --all
   Name                 State      Autostart
   -----------------------------------------
   default              active     yes
   guest_images_vhba    inactive   no
   ```

3. **Start the storage pool**

   Use the `virsh pool-start` command to mount the storage pool.

   ```
   # virsh pool-start guest_images_vhba
   Pool guest_images_vhba started
   ```

   **NOTE**

   The `virsh pool-start` command is only necessary for persistent storage pools. Transient storage pools are automatically started when they are created.
4. [Optional] Turn on autostart
   By default, a storage pool defined with the `virsh` command is not set to automatically start each
time virtualization services start. Use the `virsh pool-autostart` command to configure the
storage pool to autostart.

   ```
   # virsh pool-autostart guest_images_vhba
   Pool guest_images_vhba marked as autostarted
   ```

   Verification

   * Use the `virsh pool-info` command to verify that the storage pool is in the `running` state. Check
     if the sizes reported are as expected and if autostart is configured correctly.

   ```
   # virsh pool-info guest_images_vhba
   Name: guest_images_vhba
   UUID: c7466869-e82a-a66c-2187-dc9d6f0877d0
   State: running
   Persistent: yes
   Autostart: yes
   Capacity: 458.39 GB
   Allocation: 197.91 MB
   Available: 458.20 GB
   ```

14.2.9. Deleting storage pools using the CLI

To remove a storage pool from your host system, you must stop the pool and remove its XML definition.

Procedure

1. List the defined storage pools using the `virsh pool-list` command.

   ```
   # virsh pool-list --all
   Name          State      Autostart
   ------------------------------
   default       active     yes
   Downloads      active     yes
   RHEL-Storage-Pool  active  yes
   ```

2. Stop the storage pool you want to delete using the `virsh pool-destroy` command.

   ```
   # virsh pool-destroy Downloads
   Pool Downloads destroyed
   ```

3. Optional: For some types of storage pools, you can remove the directory where the storage
   pool resides using the `virsh pool-delete` command. Note that to do so, the directory must be
   empty.

   ```
   # virsh pool-delete Downloads
   Pool Downloads deleted
   ```

4. Delete the definition of the storage pool using the `virsh pool-undefine` command.
# virsh pool-undefine Downloads
Pool Downloads has been undefined

Verification

- Confirm that the storage pool was deleted.

```bash
# virsh pool-list --all
Name         State  Autostart
--------------
default      active yes
RHEL-Storage-Pool active yes
```

### 14.3. MANAGING VIRTUAL MACHINE STORAGE POOLS USING THE WEB CONSOLE

Using the RHEL web console, you can manage the storage pools to assign storage to your virtual machines (VMs).

You can use the web console to:

- View storage pool information.
- Create storage pools:
  - Create directory-based storage pools.
  - Create NFS-based storage pools.
  - Create iSCSI-based storage pools.
  - Create LVM-based storage pools.
- Remove storage pools.
- Deactivate storage pools.

#### 14.3.1. Viewing storage pool information using the web console

Using the web console, you can view detailed information about storage pools available on your system. Storage pools can be used to create disk images for your virtual machines.

**Prerequisites**

- The web console VM plug-in is installed on your system.

**Procedure**

1. Click **Storage Pools** at the top of the **Virtual Machines** interface. The Storage Pools window appears, showing a list of configured storage pools.
The information includes the following:

- **Name** - The name of the storage pool.
- **Size** - The current allocation and the total capacity of the storage pool.
- **Connection** - The connection used to access the storage pool.
- **State** - The state of the storage pool.

2. Click the arrow next to the storage pool whose information you want to see. The row expands to reveal the Overview pane with detailed information about the selected storage pool.

The information includes:

- **Target path** - The source for the types of storage pools backed by directories, such as `dir` or `netfs`.
- **Persistent** - Indicates whether or not the storage pool has a persistent configuration.
- **Autostart** - Indicates whether or not the storage pool starts automatically when the system boots up.
- **Type** - The type of the storage pool.

3. To view a list of storage volumes associated with the storage pool, click **Storage Volumes**. The Storage Volumes pane appears, showing a list of configured storage volumes.

The information includes:
- **Name** - The name of the storage volume.
- **Used by** - The VM that is currently using the storage volume.
- **Size** - The size of the volume.

**Additional resources**

- Viewing virtual machine information using the web console

**14.3.2. Creating directory-based storage pools using the web console**

A directory-based storage pool is based on a directory in an existing mounted file system. This is useful, for example, when you want to use the remaining space on the file system for other purposes.

**Prerequisites**

- The web console VM plug-in is installed on your system.

**Procedure**

1. In the RHEL web console, click **Storage pools** in the **Virtual Machines** tab. The **Storage pools** window appears, showing a list of configured storage pools, if any.

2. Click **Create storage pool**. The **Create storage pool** dialog appears.

3. Enter a name for the storage pool.

4. In the **Type** drop down menu, select **Filesystem directory**.
NOTE

If you do not see the **Filesystem directory** option in the drop down menu, then your hypervisor does not support directory-based storage pools.

5. Enter the following information:
   - **Target path** - The source for the types of storage pools backed by directories, such as *dir* or *netfs*.
   - **Startup** - Whether or not the storage pool starts when the host boots.

6. Click **Create**.
   The storage pool is created, the **Create Storage Pool** dialog closes, and the new storage pool appears in the list of storage pools.

Additional resources
- Understanding storage pools
- Viewing storage pool information using the web console

14.3.3. Creating NFS-based storage pools using the web console

An NFS-based storage pool is based on a file system that is hosted on a server.

**Prerequisites**
- The web console VM plug-in is installed on your system.

**Procedure**

1. In the RHEL web console, click **Storage pools** in the **Virtual Machines** tab.
   The **Storage pools** window appears, showing a list of configured storage pools, if any.
2. Click **Create storage pool**. The **Create storage pool** dialog appears.

3. Enter a name for the storage pool.

4. In the **Type** drop down menu, select **Network file system**.

5. Enter the rest of the information:

   - **Target path** – The path specifying the target. This will be the path used for the storage pool.
   - **Host** – The hostname of the network server where the mount point is located. This can be a hostname or an IP address.
   - **Source path** – The directory used on the network server.
   - **Startup** – Whether or not the storage pool starts when the host boots.

   **NOTE**

   If you do not see the **Network file system** option in the drop down menu, then your hypervisor does not support nfs-based storage pools.
6. Click **Create**. 
The storage pool is created. The **Create storage pool** dialog closes, and the new storage pool appears in the list of storage pools.

**Additional resources**

- Understanding storage pools
- Viewing storage pool information using the web console

**14.3.4. Creating iSCSI-based storage pools using the web console**

An iSCSI-based storage pool is based on the Internet Small Computer Systems Interface (iSCSI), an IP-based storage networking standard for linking data storage facilities.

**Prerequisites**

- The web console VM plug-in is installed on your system.

**Procedure**

1. In the RHEL web console, click **Storage pools** in the **Virtual Machines** tab. 
   The **Storage pools** window appears, showing a list of configured storage pools, if any.

2. Click **Create storage pool**. 
   The **Create storage pool** dialog appears.

3. Enter a name for the storage pool.

4. In the **Type** drop down menu, select **iSCSI target**.
5. Enter the rest of the information:

- **Target Path** - The path specifying the target. This will be the path used for the storage pool.
- **Host** - The hostname or IP address of the iSCSI server.
- **Source path** - The unique iSCSI Qualified Name (IQN) of the iSCSI target.
- **Startup** - Whether or not the storage pool starts when the host boots.

6. Click **Create**.
   The storage pool is created. The **Create storage pool** dialog closes, and the new storage pool appears in the list of storage pools.

Additional resources

- **Understanding storage pools**
- **Viewing storage pool information using the web console**

14.3.5. Creating disk-based storage pools using the web console

A disk-based storage pool uses entire disk partitions.
WARNING

- Depending on the version of libvirt being used, dedicating a disk to a storage pool may reformat and erase all data currently stored on the disk device. It is strongly recommended that you back up the data on the storage device before creating a storage pool.

- When whole disks or block devices are passed to the VM, the VM will likely partition it or create its own LVM groups on it. This can cause the host machine to detect these partitions or LVM groups and cause errors. These errors can also occur when you manually create partitions or LVM groups and pass them to the VM.

To avoid these errors, use file-based storage pools instead.

Prerequisites

- The web console VM plug-in is installed on your system.

Procedure

1. In the RHEL web console, click Storage pools in the Virtual Machines tab. The Storage pools window appears, showing a list of configured storage pools, if any.

2. Click Create storage pool.

3. Enter a name for the storage pool.

4. In the Type drop down menu, select Physical disk device.
NOTE

If you do not see the Physical disk device option in the drop down menu, then your hypervisor does not support disk-based storage pools.

5. Enter the rest of the information:

- **Target Path** - The path specifying the target device. This will be the path used for the storage pool.

- **Source path** - The path specifying the storage device. For example, `/dev/sdb`.

- **Format** - The type of the partition table.

- **Startup** - Whether or not the storage pool starts when the host boots.

6. Click **Create**.

The storage pool is created. The **Create storage pool** dialog closes, and the new storage pool appears in the list of storage pools.

**Additional resources**

- Understanding storage pools

- Viewing storage pool information using the web console

**14.3.6. Creating LVM-based storage pools using the web console**

An LVM-based storage pool is based on volume groups, which you can manage using the Logical Volume Manager (LVM). A volume group is a combination of multiple physical volumes that creates a single storage structure.
NOTE

- LVM-based storage pools do not provide the full flexibility of LVM.
- *libvirt* supports thin logical volumes, but does not provide the features of thin storage pools.
- LVM-based storage pools require a full disk partition. If you activate a new partition or device using *virsh* commands, the partition will be formatted and all data will be erased. If you are using a host’s existing volume group, as in these procedures, nothing will be erased.
- To create a volume group with multiple devices, use the LVM utility instead, see How to create a volume group in Linux with LVM. For more detailed information about volume groups, refer to the *Red Hat Enterprise Linux Logical Volume Manager Administration Guide*.

Prerequisites

- The web console VM plug-in is installed on your system.

Procedure

1. In the RHEL web console, click **Storage pools** in the **Virtual Machines** tab. The **Storage pools** window appears, showing a list of configured storage pools, if any.

2. Click **Create storage pool**. The **Create storage pool** dialog appears.

3. Enter a name for the storage pool.

4. In the **Type** drop down menu, select **LVM volume group**.
NOTE
If you do not see the LVM volume group option in the drop down menu, then your hypervisor does not support LVM-based storage pools.

5. Enter the rest of the information:

   - **Source volume group** - The name of the LVM volume group that you wish to use.
   - **Startup** - Whether or not the storage pool starts when the host boots.

6. Click **Create**.
   The storage pool is created. The Create storage pool dialog closes, and the new storage pool appears in the list of storage pools.

Additional resources

- Understanding storage pools
- Viewing storage pool information using the web console

14.3.7. Removing storage pools using the web console

You can remove storage pools to free up resources on the host or on the network to improve system performance. Deleting storage pools also frees up resources that can then be used by other virtual machines (VMs).

**IMPORTANT**

Unless explicitly specified, deleting a storage pool does not simultaneously delete the storage volumes inside that pool.

To delete a storage pool using the RHEL web console, see the following procedure.

**NOTE**

If you want to temporarily deactivate a storage pool instead of deleting it, see Deactivating storage pools using the web console
Prerequisites

- The web console VM plug-in is installed on your system.
- If you want to delete a storage volume along with the pool, you must first detach the disk from the VM.

Procedure

1. Click **Storage Pools** on the **Virtual Machines** tab. The **Storage Pools** window appears, showing a list of configured storage pools.

![Storage Pools window](image)

2. In the **Storage Pools** window, click the storage pool you want to delete. The row expands to reveal the Overview pane with basic information about the selected storage pool and controls for deactivating or deleting the storage pool.

![Overview pane](image)

3. Click the Menu button ⋄ and click **Delete**. A confirmation dialog appears.

![Delete dialog](image)

4. Optional: To delete the storage volumes inside the pool, select the check box in the dialog.

5. Click **Delete**. The storage pool is deleted. If you had selected the checkbox in the previous step, the associated storage volumes are deleted as well.

Additional resources

- Understanding storage pools
14.3.8. Deactivating storage pools using the web console

If you do not want to permanently delete a storage pool, you can temporarily deactivate it instead.

When you deactivate a storage pool, no new volumes can be created in that pool. However, any virtual machines (VMs) that have volumes in that pool will continue to run. This is useful for a number of reasons, for example, you can limit the number of volumes that can be created in a pool to increase system performance.

To deactivate a storage pool using the RHEL web console, see the following procedure.

Prerequisites

- The web console VM plug-in is installed on your system.

Procedure

1. Click **Storage Pools** at the top of the Virtual Machines tab. The Storage Pools window appears, showing a list of configured storage pools.

2. In the **Storage Pools** window, click the storage pool you want to deactivate. The row expands to reveal the Overview pane with basic information about the selected storage pool and controls for deactivating and deleting the VM.

3. Click **Deactivate**. The storage pool is deactivated.

Additional resources

- Understanding storage pools
- Viewing storage pool information using the web console

14.4. PARAMETERS FOR CREATING STORAGE POOLS
Based on the type of storage pool you require, you can modify its XML configuration file and define a specific type of storage pool. This section provides information about the XML parameters required for creating various types of storage pools along with examples.

### 14.4.1. Directory-based storage pool parameters

When you want to create or modify a directory-based storage pool using an XML configuration file, you must include certain required parameters. See the following table for more information about these parameters.

You can use the `virsh pool-define` command to create a storage pool based on the XML configuration in a specified file. For example:

```bash
# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_dir
```

#### Parameters

The following table provides a list of required parameters for the XML file for a directory-based storage pool.

<table>
<thead>
<tr>
<th>Description</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>The type of storage pool</td>
<td><code>&lt;pool type='dir'&gt;</code></td>
</tr>
<tr>
<td>The name of the storage pool</td>
<td><code>&lt;name&gt;</code>&lt;name/&gt;&lt;/name&gt;</td>
</tr>
<tr>
<td>The path specifying the target. This will be the path used for the storage</td>
<td><code>&lt;target&gt;</code>&lt;path&gt;<code>target_path'&lt;/path&gt;</code>&lt;path/&gt;</td>
</tr>
<tr>
<td>pool.</td>
<td>&lt;/target&gt;</td>
</tr>
</tbody>
</table>

#### Example

The following is an example of an XML file for a storage pool based on the `/guest_images` directory:

```xml
<pool type='dir'>
  <name>dirpool</name>
  <target>
    <path>/guest_images</path>
  </target>
</pool>
```

#### Additional resources

- Creating directory-based storage pools using the CLI

### 14.4.2. Disk-based storage pool parameters

When you want to create or modify a disk-based storage pool using an XML configuration file, you must include certain required parameters. See the following table for more information about these parameters.
You can use the `virsh pool-define` command to create a storage pool based on the XML configuration in a specified file. For example:

```bash
# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_disk
```

### Parameters

The following table provides a list of required parameters for the XML file for a disk-based storage pool.

**Table 14.2. Disk-based storage pool parameters**

<table>
<thead>
<tr>
<th>Description</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>The type of storage pool</td>
<td><code>&lt;pool type='disk'&gt;</code></td>
</tr>
<tr>
<td>The name of the storage pool</td>
<td><code>&lt;name&gt;name&lt;/name&gt;</code></td>
</tr>
<tr>
<td>The path specifying the storage device. For example, <code>/dev/sdb</code></td>
<td><code>&lt;source&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>  &lt;path&gt;source_path&lt;/path&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/source&gt;</code></td>
</tr>
<tr>
<td>The path specifying the target device. This will be the path used for the storage pool.</td>
<td><code>&lt;target&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>  &lt;path&gt;target_path&lt;/path&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/target&gt;</code></td>
</tr>
</tbody>
</table>

**Example**

The following is an example of an XML file for a disk-based storage pool:

```xml
<pool type='disk'>
  <name>phy_disk</name>
  <source>
    <device path='/dev/sdb'/>
    <format type='gpt'/>
  </source>
  <target>
    <path>/dev</path>
  </target>
</pool>
```

### Additional resources

- Creating disk-based storage pools using the CLI

**14.4.3. Filesystem-based storage pool parameters**

When you want to create or modify a filesystem-based storage pool using an XML configuration file, you must include certain required parameters. See the following table for more information about these parameters.

You can use the `virsh pool-define` command to create a storage pool based on the XML configuration in a specified file. For example:
# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_fs

Parameters

The following table provides a list of required parameters for the XML file for a filesystem-based storage pool.

Table 14.3. Filesystem-based storage pool parameters

<table>
<thead>
<tr>
<th>Description</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>The type of storage pool</td>
<td>&lt;pool type='fs'&gt;</td>
</tr>
<tr>
<td>The name of the storage pool</td>
<td>&lt;name&gt;name&lt;/name&gt;</td>
</tr>
<tr>
<td>The path specifying the partition. For example, /dev/sdc1</td>
<td>&lt;source&gt;&lt;device path=device_path /&gt;&lt;/source&gt;</td>
</tr>
<tr>
<td>The file system type, for example ext4.</td>
<td>&lt;format type=fs_type /&gt;</td>
</tr>
<tr>
<td>The path specifying the target. This will be the path used for the storage pool.</td>
<td>&lt;target&gt;&lt;path&gt;path-to-pool&lt;/path&gt;&lt;/target&gt;</td>
</tr>
</tbody>
</table>

Example

The following is an example of an XML file for a storage pool based on the /dev/sdc1 partition:

```xml
<pool type='fs'>
  <name>guest_images_fs</name>
  <source>
    <device path='/dev/sdc1'/>
    <format type='auto'/>
  </source>
  <target>
    <path>/guest_images</path>
  </target>
</pool>
```

Additional resources

- Creating filesystem-based storage pools using the CLI

14.4.4. iSCSI-based storage pool parameters

When you want to create or modify an iSCSI-based storage pool using an XML configuration file, you must include certain required parameters. See the following table for more information about these parameters.
You can use the `virsh pool-define` command to create a storage pool based on the XML configuration in a specified file. For example:

```bash
# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_iscsi
```

**Parameters**

The following table provides a list of required parameters for the XML file for an iSCSI-based storage pool.

**Table 14.4. iSCSI-based storage pool parameters**

<table>
<thead>
<tr>
<th>Description</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>The type of storage pool</td>
<td><code>&lt;pool type='iscsi'&gt;</code></td>
</tr>
<tr>
<td>The name of the storage pool</td>
<td><code>&lt;name&gt;name&lt;/name&gt;</code></td>
</tr>
<tr>
<td>The name of the host</td>
<td><code>&lt;source&gt;</code>&lt;br&gt;<code>&lt;host name='hostname'/&gt;</code></td>
</tr>
<tr>
<td>The iSCSI IQN</td>
<td><code>&lt;device path='iSCSI_IQN'/&gt;</code>&lt;br&gt;<code>&lt;/source&gt;</code></td>
</tr>
<tr>
<td>The path specifying the target. This will be the path used for the storage pool.</td>
<td><code>&lt;target&gt;</code>&lt;br&gt;<code>&lt;path&gt;/dev/disk/by-path&lt;/path&gt;</code></td>
</tr>
<tr>
<td>[Optional] The IQN of the iSCSI initiator. This is only needed when the ACL restricts the LUN to a particular initiator.</td>
<td><code>&lt;initiator&gt;</code>&lt;br&gt;<code>&lt;iqn name='initiator0' /&gt;</code>&lt;br&gt;<code>&lt;/initiator&gt;</code></td>
</tr>
</tbody>
</table>

**NOTE**

The IQN of the iSCSI initiator can be determined using the `virsh find-storage-pool-sources-as iscsi` command.

**Example**

The following is an example of an XML file for a storage pool based on the specified iSCSI device:

```xml
<pool type='iscsi'>
  <name>iSCSI_pool</name>
  <source>
    <host name='server1.example.com'/>
    <device path='iqn.2010-05.com.example.server1:iscsrhel7guest'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```
Additional resources

- Creating iSCSI-based storage pools using the CLI

14.4.5. LVM-based storage pool parameters

When you want to create or modify an LVM-based storage pool using an XML configuration file, you must include certain required parameters. See the following table for more information about these parameters.

You can use the `virsh pool-define` command to create a storage pool based on the XML configuration in a specified file. For example:

```
# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_logical
```

Parameters

The following table provides a list of required parameters for the XML file for a LVM-based storage pool.

Table 14.5. LVM-based storage pool parameters

<table>
<thead>
<tr>
<th>Description</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>The type of storage pool</td>
<td><code>&lt;pool type='logical'&gt;</code></td>
</tr>
<tr>
<td>The name of the storage pool</td>
<td><code>&lt;name&gt;name&lt;/name&gt;</code></td>
</tr>
</tbody>
</table>
| The path to the device for the storage pool      | `<source>
  <device path='device_path'/>`
| The name of the volume group                     | `<name>VG-name</name>` |
| The virtual group format                         | `<format type='lvm2' />
  </source>` |
| The target path                                  | `<target>
  <path=target_path />
  </target>` |

NOTE

If the logical volume group is made of multiple disk partitions, there may be multiple source devices listed. For example:

```
<source>
  <device path='/dev/sda1'/>
  <device path='/dev/sdb3'/>
  <device path='/dev/sdc2'/>
...
</source>
```
Example

The following is an example of an XML file for a storage pool based on the specified LVM:

```xml
<pool type='logical'>
  <name>guest_images_lvm</name>
  <source>
    <device path='/dev/sdc'/>
    <name>libvirt_lvm</name>
    <format type='lvm2'/>
  </source>
  <target>
    <path>/dev/libvirt_lvm</path>
  </target>
</pool>
```

Additional resources

- Creating LVM-based storage pools using the CLI

14.4.6. NFS-based storage pool parameters

When you want to create or modify an NFS-based storage pool using an XML configuration file, you must include certain required parameters. See the following table for more information about these parameters.

You can use the `virsh pool-define` command to create a storage pool based on the XML configuration in a specified file. For example:

```
# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_netfs
```

Parameters

The following table provides a list of required parameters for the XML file for an NFS-based storage pool.

Table 14.6. NFS-based storage pool parameters

<table>
<thead>
<tr>
<th>Description</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>The type of storage pool</td>
<td><code>&lt;pool type='netfs'&gt;</code></td>
</tr>
<tr>
<td>The name of the storage pool</td>
<td><code>&lt;name&gt;name&lt;/name&gt;</code></td>
</tr>
<tr>
<td>The hostname of the network server where the mount point is located. This</td>
<td><code>&lt;source&gt;</code></td>
</tr>
<tr>
<td>can be a hostname or an IP address.</td>
<td><code>&lt;host name=hostname/&gt;</code></td>
</tr>
<tr>
<td>Description</td>
<td>XML</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td>The format of the storage pool</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>&lt;format type='nfs' /&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;format type='cifs' /&gt;</td>
</tr>
<tr>
<td>The directory used on the network server</td>
<td>&lt;dir path=source_path/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/source&gt;</td>
</tr>
<tr>
<td>The path specifying the target. This will be the path used for the storage pool.</td>
<td>&lt;target&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;path&gt;target_path&lt;/path&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/target&gt;</td>
</tr>
</tbody>
</table>

**Example**

The following is an example of an XML file for a storage pool based on the `/home/net_mount` directory of the file_server NFS server:

```xml
<pool type='netfs'>
  <name>nfspool</name>
  <source>
    <host name='file_server'/>
    <format type='nfs'/>
    <dir path='/home/net_mount'/>
  </source>
  <target>
    <path>/var/lib/libvirt/images/nfspool</path>
  </target>
</pool>
```

**Additional resources**

- Creating NFS-based storage pools using the CLI

**14.4.7. Parameters for SCSI-based storage pools with vHBA devices**

To create or modify an XML configuration file for a SCSI-based storage pool that uses a virtual host adapter bus (vHBA) device, you must include certain required parameters in the XML configuration file. See the following table for more information about the required parameters.

You can use the `virsh pool-define` command to create a storage pool based on the XML configuration in a specified file. For example:

```bash
# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_vhba
```

**Parameters**

The following table provides a list of required parameters for the XML file for a SCSI-based storage pool with vHBA.
Table 14.7. Parameters for SCSI-based storage pools with vHBA devices

<table>
<thead>
<tr>
<th>Description</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>The type of storage pool</td>
<td><code>&lt;pool type='scsi'&gt;</code></td>
</tr>
<tr>
<td>The name of the storage pool</td>
<td><code>&lt;name&gt;</code>name&lt;/name&gt;`</td>
</tr>
<tr>
<td>The identifier of the vHBA. The <code>parent</code> attribute is optional.</td>
<td><code>&lt;source&gt;</code>&lt;adapter type='fc_host' [parent='parent_scsi_device'] wwnn='WWNN' wwpn='WWPN' /&gt;&lt;/source&gt;`</td>
</tr>
<tr>
<td>The target path. This will be the path used for the storage pool.</td>
<td><code>&lt;target&gt;</code>&lt;path=target_path /&gt;`</td>
</tr>
</tbody>
</table>

**IMPORTANT**

When the `<path>` field is `/dev/`, `libvirt` generates a unique short device path for the volume device path. For example, `/dev/sdc`. Otherwise, the physical host path is used. For example, `/dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016044602198-lun-0`. The unique short device path allows the same volume to be listed in multiple virtual machines (VMs) by multiple storage pools. If the physical host path is used by multiple VMs, duplicate device type warnings may occur.

**NOTE**

The `parent` attribute can be used in the `<adapter>` field to identify the physical HBA parent from which the NPIV LUNs by varying paths can be used. This field, `scsi_hostN`, is combined with the `vports` and `max_vports` attributes to complete the parent identification. The `parent`, `parent_wwnn`, `parent_wwpn`, or `parent_fabric_wwn` attributes provide varying degrees of assurance that after the host reboots the same HBA is used.

- If no `parent` is specified, `libvirt` uses the first `scsi_hostN` adapter that supports NPIV.
- If only the `parent` is specified, problems can arise if additional SCSI host adapters are added to the configuration.
- If `parent_wwnn` or `parent_wwpn` is specified, after the host reboots the same HBA is used.
- If `parent_fabric_wwn` is used, after the host reboots an HBA on the same fabric is selected, regardless of the `scsi_hostN` used.

**Examples**

The following are examples of XML files for SCSI-based storage pools with vHBA.
A storage pool that is the only storage pool on the HBA:

```xml
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <source>
    <adapter type='fc_host' wwnn='5001a4a93526d0a1' wwpn='5001a4ace3ee047d'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

A storage pool that is one of several storage pools that use a single vHBA and uses the `parent` attribute to identify the SCSI host device:

```xml
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <source>
    <adapter type='fc_host' parent='scsi_host3' wwnn='5001a4a93526d0a1' wwpn='5001a4ace3ee047d'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

Additional resources

- Creating SCSI-based storage pools with vHBA devices using the CLI

### 14.5. MANAGING VIRTUAL MACHINE STORAGE VOLUMES USING THE CLI

You can use the CLI to manage the following aspects of your storage volumes to assign storage to your virtual machines (VMs):

- View storage volume information
- Create storage volumes
- Delete storage volumes

#### 14.5.1. Viewing storage volume information using the CLI

Using the command line, you can view a list of all storage pools available on your host, as well as details about a specified storage pool

**Procedure**

1. Use the `virsh vol-list` command to list the storage volumes in a specified storage pool.

```bash
# virsh vol-list --pool RHEL-Storage-Pool --details
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Path</th>
<th>Type</th>
<th>Capacity</th>
<th>Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Use the **virsh vol-info** command to list the storage volumes in a specified storage pool.

```bash
# vol-info --pool RHEL-Storage-Pool --vol RHEL_Volume.qcow2
Name: RHEL_Volume.qcow2
Type: file
Capacity: 60.00 GiB
Allocation: 13.93 GiB
```

### 14.5.2. Creating and assigning storage volumes using the CLI

To obtain a disk image and attach it to a virtual machine (VM) as a virtual disk, create a storage volume and assign its XML configuration to a the VM.

#### Prerequisites

- A storage pool with unallocated space is present on the host.
  - To verify, list the storage pools on the host:

```bash
# virsh pool-list --details
```

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
<th>Autostart</th>
<th>Persistent</th>
<th>Capacity</th>
<th>Allocation</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>running</td>
<td>yes</td>
<td>yes</td>
<td>48.97 GiB</td>
<td>36.34 GiB</td>
<td>12.63 GiB</td>
</tr>
<tr>
<td>Downloads</td>
<td>running</td>
<td>yes</td>
<td>yes</td>
<td>175.92 GiB</td>
<td>121.20 GiB</td>
<td>54.72 GiB</td>
</tr>
<tr>
<td>VM-disks</td>
<td>running</td>
<td>yes</td>
<td>yes</td>
<td>175.92 GiB</td>
<td>121.20 GiB</td>
<td>54.72 GiB</td>
</tr>
</tbody>
</table>

- If you do not have an existing storage pool, create one. For more information, see [Managing storage for virtual machines](#).

#### Procedure

1. Create a storage volume using the **virsh vol-create-as** command. For example, to create a 20 GB qcow2 volume based on the **guest-images-fs** storage pool:

   ```bash
   # virsh vol-create-as --pool guest-images-fs --name vm-disk1 --capacity 20 --format qcow2
   ```

   **Important**: Specific storage pool types do not support the **virsh vol-create-as** command and instead require specific processes to create storage volumes:

   - **iSCSI-based** - Prepare the iSCSI LUNs in advance on the iSCSI server.

   - **Multipath-based** - Use the **multipathd** command to prepare or manage the multipath.
- **vHBA-based** - Prepare the fibre channel card in advance.

2. Create an XML file, and add the following lines in it. This file will be used to add the storage volume as a disk to a VM.

   ```xml
   <disk type='volume' device='disk'>
     <driver name='qemu' type='qcow2'/>
     <source pool='guest-images-fs' volume='vm-disk1'/>
     <target dev='hdk' bus='ide'/>
   </disk>
   
   This example specifies a virtual disk that uses the `vm-disk1` volume, created in the previous step, and sets the volume to be set up as disk `hdk` on an `ide` bus. Modify the respective parameters as appropriate for your environment.

   **Important**: With specific storage pool types, you must use different XML formats to describe a storage volume disk.

   - For **multipath-based** pools:
     ```xml
     <disk type='block' device='disk'>
     <driver name='qemu' type='raw'/>
     <source dev='/dev/mapper/mpatha'/>
     <target dev='sda' bus='scsi'/>
     </disk>
     
     - For **RBD-based storage pools**:
     ```xml
     <disk type='network' device='disk'>
     <driver name='qemu' type='raw'/>
     <source protocol='rbd' name='pool/image'>
       <host name='mon1.example.org' port='6321'/>
     </source>
     <target dev='vdc' bus='virtio'/>
     </disk>
     ```

3. Use the XML file to assign the storage volume as a disk to a VM. For example, to assign a disk defined in `~/vm-disk1.xml` to the `testguest1` VM:

   ```bash
   # attach-device --config testguest1 ~/vm-disk1.xml
   ```

**Verification**

- In the guest operating system of the VM, confirm that the disk image has become available as an un-formatted and un-allocated disk.

### 14.5.3. Deleting storage volumes using the CLI

To remove a storage volume from your host system, you must stop the pool and remove its XML definition.

**Prerequisites**

- Any virtual machine that uses the storage volume you want to delete is shut down.
Procedure

1. Use the `virsh vol-list` command to list the storage volumes in a specified storage pool.

   ```
   # virsh vol-list --pool RHEL-SP
   Name                  Path
   -----------------------------------------------
   .bash_history         /home/VirtualMachines/.bash_history
   .bash_logout          /home/VirtualMachines/.bash_logout
   .bash_profile         /home/VirtualMachines/.bash_profile
   .bashrc               /home/VirtualMachines/.bashrc
   .git-prompt.sh        /home/VirtualMachines/.git-prompt.sh
   .gitconfig            /home/VirtualMachines/.gitconfig
   vm-disk1              /home/VirtualMachines/vm-disk1
   ```

2. Optional: Use the `virsh vol-wipe` command to wipe a storage volume. For example, to wipe a storage volume named `vm-disk1` associated with the storage pool `RHEL-SP`:

   ```
   # virsh vol-wipe --pool RHEL-SP vm-disk1
   Vol vm-disk1 wiped
   ```

3. Use the `virsh vol-delete` command to delete a storage volume. For example, to delete a storage volume named `vm-disk1` associated with the storage pool `RHEL-SP`:

   ```
   # virsh vol-delete --pool RHEL-SP vm-disk1
   Vol vm-disk1 deleted
   ```

Verification

- Use the `virsh vol-list` command again to verify that the storage volume was deleted.

   ```
   # virsh vol-list --pool RHEL-SP
   Name                  Path
   -----------------------------------------------
   .bash_history         /home/VirtualMachines/.bash_history
   .bash_logout          /home/VirtualMachines/.bash_logout
   .bash_profile         /home/VirtualMachines/.bash_profile
   .bashrc               /home/VirtualMachines/.bashrc
   .git-prompt.sh        /home/VirtualMachines/.git-prompt.sh
   .gitconfig            /home/VirtualMachines/.gitconfig
   ```

14.6. MANAGING VIRTUAL MACHINE STORAGE VOLUMES USING THE WEB CONSOLE

Using the RHEL, you can manage the storage volumes used to allocate storage to your virtual machines (VMs).

You can use the RHEL web console to:

- Create storage volumes.
- Remove storage volumes.
14.6.1. Creating storage volumes using the web console

To create a functioning virtual machine (VM) you require a local storage device assigned to the VM that can store the VM image and VM-related data. You can create a storage volume in a storage pool and assign it to a VM as a storage disk.

To create storage volumes using the web console, see the following procedure.

Prerequisites

- The web console VM plug-in is installed on your system.

Procedure

1. Click **Storage Pools** at the top of the Virtual Machines tab. The Storage Pools window appears, showing a list of configured storage pools.

2. In the **Storage Pools** window, click the storage pool from which you want to create a storage volume.
   The row expands to reveal the Overview pane with basic information about the selected storage pool.

3. Click **Storage Volumes** next to the Overview tab in the expanded row.
   The Storage Volume tab appears with basic information about existing storage volumes, if any.

4. Click **Create Volume**.
   The Create Storage Volume dialog appears.
5. Enter the following information in the Create Storage Volume dialog:

- **Name** - The name of the storage volume.
- **Size** - The size of the storage volume in MiB or GiB.
- **Format** - The format of the storage volume. The supported types are `qcow2` and `raw`.

6. Click **Create**.
   The storage volume is created, the Create Storage Volume dialog closes, and the new storage volume appears in the list of storage volumes.

### Additional resources

- [Understanding storage volumes](#)
- [Adding new disks to virtual machines using the web console](#)

#### 14.6.2. Removing storage volumes using the web console

You can remove storage volumes to free up space in the storage pool, or to remove storage items associated with defunct virtual machines (VMs).

To remove storage volumes using the RHEL web console, see the following procedure.

### Prerequisites

- The web console VM plug-in is installed on your system.
- Any virtual machine that uses the storage volume you want to delete is shut down.

### Procedure

1. Click **Storage Pools** at the top of the Virtual Machines tab. The Storage Pools window appears, showing a list of configured storage pools.
2. In the **Storage Pools** window, click the storage pool from which you want to remove a storage volume. The row expands to reveal the Overview pane with basic information about the selected storage pool.

3. Click **Storage Volumes** next to the Overview tab in the expanded row. The Storage Volume tab appears with basic information about existing storage volumes, if any.

4. Select the storage volume you want to remove.

5. Click **Delete 1 Volume**

**Additional resources**

- Understanding storage volumes
14.7. MANAGING VIRTUAL MACHINE STORAGE DISKS USING THE WEB CONSOLE

Using RHEL, you can manage the storage disks that are attached to your virtual machines (VMs).

You can use the RHEL web console to:

- View VM disk information.
- Add new disks to a VM.
- Attach disks to a VM.
- Detach disks from a VM.

14.7.1. Viewing virtual machine disk information in the web console

Using the web console, you can view detailed information about disks assigned to a selected virtual machine (VM).

Prerequisites

- The web console VM plug-in is installed on your system.

Procedure

1. Click the VM whose information you want to see.
   
   A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM’s graphical interface.

2. Scroll to Disks.
   
   The Disks section displays information about the disks assigned to the VM as well as options to Add, Remove, or Edit disks.

<table>
<thead>
<tr>
<th>Disks</th>
<th>Add disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>Used</td>
</tr>
<tr>
<td>disk 1</td>
<td>8.9 GB</td>
</tr>
<tr>
<td>disk 2</td>
<td>0 GB</td>
</tr>
</tbody>
</table>

The information includes the following:

- **Device** - The device type of the disk.
- **Used** - The amount of disk currently allocated.
- **Capacity** - The maximum size of the storage volume.
- **Bus** - The type of disk device that is emulated.
- **Access** - Whether the disk is **Writeable** or **Read-only**. For **raw** disks, you can also set the access to **Writeable and shared**.
- **Source** - The disk device or file.
Additional resources

- Viewing virtual machine information using the web console

14.7.2. Adding new disks to virtual machines using the web console

You can add new disks to virtual machines (VMs) by creating a new storage volume and attaching it to a VM using the RHEL 9 web console.

Prerequisites

- The web console VM plug-in is installed on your system.

Procedure

1. In the Virtual Machines interface, click the VM for which you want to create and attach a new disk.
   A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM’s graphical interface.

2. Scroll to Disks.
   The Disks section displays information about the disks assigned to the VM as well as options to Add, Remove, or Edit disks.

3. Click Add Disk.
   The Add Disk dialog appears.

4. Select the Create New option.
5. Configure the new disk.
   - **Pool** - Select the storage pool from which the virtual disk will be created.
   - **Name** - Enter a name for the virtual disk that will be created.
   - **Size** - Enter the size and select the unit (MiB or GiB) of the virtual disk that will be created.
   - **Format** - Select the format for the virtual disk that will be created. The supported types are *qcow2* and *raw*.
   - **Persistence** - If checked, the virtual disk is persistent. If not checked, the virtual disk is transient.

   **NOTE**
   Transient disks can only be added to VMs that are running.

   - **Additional Options** - Set additional configurations for the virtual disk.
     - **Cache** - Select the cache mechanism.
     - **Bus** - Select the type of disk device to emulate.

6. Click **Add**.
   The virtual disk is created and connected to the VM.

**Additional resources**
- Viewing virtual machine disk information in the web console
- Attaching existing disks to virtual machines using the web console
- Detaching disks from virtual machines using the web console

**14.7.3. Attaching existing disks to virtual machines using the web console**

Using the web console, you can attach existing storage volumes as disks to a virtual machine (VM).

**Prerequisites**
- The web console VM plug-in is installed on your system.

**Procedure**

1. In the **Virtual Machines** interface, click the VM for which you want to create and attach a new disk.
   A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM's graphical interface.

2. Scroll to **Disks**.
   The Disks section displays information about the disks assigned to the VM as well as options to **Add**, **Remove**, or **Edit** disks.
3. Click **Add Disk**.
   The Add Disk dialog appears.

4. Click the **Use Existing** radio button.
   The appropriate configuration fields appear in the Add Disk dialog.

5. Configure the disk for the VM.
   - **Pool** - Select the storage pool from which the virtual disk will be attached.
• **Volume** - Select the storage volume that will be attached.

• **Persistence** - Available when the VM is running. Select the **Always attach** checkbox to make the virtual disk persistent. Clear the checkbox to make the virtual disk transient.

• **Additional Options** - Set additional configurations for the virtual disk.
  
  o **Cache** - Select the cache mechanism.

  o **Bus** - Select the type of disk device to emulate.

6. Click **Add**
   The selected virtual disk is attached to the VM.

**Additional resources**

- Viewing virtual machine disk information in the web console
- Adding new disks to virtual machines using the web console
- Detaching disks from virtual machines using the web console

**14.7.4. Detaching disks from virtual machines using the web console**

Using the web console, you can detach disks from virtual machines (VMs).

**Prerequisites**

- The web console VM plug-in is installed on your system.

**Procedure**

1. In the **Virtual Machines** interface, click the VM from which you want to detach a disk. A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM’s graphical interface.

2. Scroll to **Disks**. The Disks section displays information about the disks assigned to the VM as well as options to Add, Remove, or Edit disks.

3. Click the **Remove** button next to the disk you want to detach from the VM. A **Remove Disk** confirmation dialog box appears.

4. In the confirmation dialog box, click **Remove**. The virtual disk is detached from the VM.

**Additional resources**
14.8. SECURING ISCSI STORAGE POOLS WITH LIBVIRT SECRETS

User name and password parameters can be configured with `virsh` to secure an iSCSI storage pool. You can configure this before or after you define the pool, but the pool must be started for the authentication settings to take effect.

The following provides instructions for securing iSCSI-based storage pools with `libvirt` secrets.

NOTE

This procedure is required if a `user_ID` and `password` were defined when creating the iSCSI target.

Prerequisites

- Ensure that you have created an iSCSI-based storage pool. For more information, see Creating iSCSI-based storage pools using the CLI.

Procedure

1. Create a libvirt secret file with a challenge-handshake authentication protocol (CHAP) user name. For example:

   ```xml
   <secret ephemeral='no' private='yes'>
     <description>Passphrase for the iSCSI example.com server</description>
     <usage type='iscsi'>
       <target>iscsirhel7secret</target>
     </usage>
   </secret>
   ``

2. Define the libvirt secret with the `virsh secret-define` command.

   ```
   # virsh secret-define secret.xml
   ```

3. Verify the UUID with the `virsh secret-list` command.

   ```
   # virsh secret-list
   UUID Usage
   ---------------------------------------------
   2d7891af-20be-4e5e-af83-190e8a922360 iscsi iscsirhel7secret
   ```

4. Assign a secret to the UUID in the output of the previous step using the `virsh secret-set-value` command. This ensures that the CHAP username and password are in a libvirt-controlled secret list. For example:

   ```
   # virsh secret-set-value --interactive 2d7891af-20be-4e5e-af83-190e8a922360
   Enter new value for secret:
   Secret value set
   ```
5. Add an authentication entry in the storage pool’s XML file using the `virsh edit` command, and add an `<auth>` element, specifying `authentication type`, `username`, and `secret usage`. For example:

   `<pool type='iscsi'>
   <name>iscsirhel7pool</name>
   <source>
     <host name='192.168.122.1'/>
     <device path='iqn.2010-05.com.example.server1:iscsirhel7guest'/>
     <auth type='chap' username='redhat'>
       <secret usage='iscsirhel7secret'/>
     </auth>
   </source>
   <target>
     <path>/dev/disk/by-path</path>
   </target>
   </pool>

   NOTE

   The `<auth>` sub-element exists in different locations within the virtual machine’s `<pool>` and `<disk>` XML elements. For a `<pool>`, `<auth>` is specified within the `<source>` element, as this describes where to find the pool sources, since authentication is a property of some pool sources (iSCSI and RBD). For a `<disk>`, which is a sub-element of a domain, the authentication to the iSCSI or RBD disk is a property of the disk. In addition, the `<auth>` sub-element for a disk differs from that of a storage pool.

   `<auth username='redhat'>
   <secret type='iscsi' usage='iscsirhel7secret'/>
   </auth>

6. To activate the changes, activate the storage pool. If the pool has already been started, stop and restart the storage pool:

   # virsh pool-destroy iscsirhel7pool
   # virsh pool-start iscsirhel7pool

14.9. CREATING VHBAS

A virtual host bus adapter (vHBA) device connects the host system to an SCSI device and is required for creating an SCSI-based storage pool.

You can create a vHBA device by defining it in an XML configuration file.

Procedure

1. Locate the HBAs on your host system, using the `virsh nodedev-list --cap vports` command. The following example shows a host that has two HBAs that support vHBA:
2. View the HBA’s details, using the `virsh nodedev-dumpxml HBA_device` command.

```bash
# virsh nodedev-list --cap vports
scsi_host3
scsi_host4
```

The output from the command lists the `<name>`, `<wwnn>`, and `<wwpn>` fields, which are used to create a vHBA. `<max_vports>` shows the maximum number of supported vHBAs. For example:

```xml
<device>
  <name>scsi_host3</name>
  <path>/sys/devices/pci0000:00/0000:00:04.0/0000:10:00.0/host3</path>
  <parent>pci_0000_10_00_0</parent>
  <capability type='scsi_host'>
    <host>3</host>
    <unique_id>0</unique_id>
    <capability type='fc_host'>
      <wwnn>20000000c9848140</wwnn>
      <wwpn>10000000c9848140</wwpn>
      <fabric_wwn>2002000573de9a81</fabric_wwn>
    </capability>
    <capability type='vport_ops'>
      <max_vports>127</max_vports>
      <vports>0</vports>
    </capability>
  </capability>
</device>
```

In this example, the `<max_vports>` value shows there are a total 127 virtual ports available for use in the HBA configuration. The `<vports>` value shows the number of virtual ports currently being used. These values update after creating a vHBA.

3. Create an XML file similar to one of the following for the vHBA host. In these examples, the file is named `vhba_host3.xml`. This example uses `scsi_host3` to describe the parent vHBA.

```xml
<device>
  <parent>scsi_host3</parent>
  <capability type='scsi_host'>
    <capability type='fc_host'>
      <wwnn>20000000c9848140</wwnn>
      <wwpn>10000000c9848140</wwpn>
      <fabric_wwn>2002000573de9a81</fabric_wwn>
    </capability>
</device>
```

This example uses a WWNN/WWPN pair to describe the parent vHBA.

```xml
<device>
  <name>vhba</name>
  <parent wwnn='20000000c9848140' wwpn='10000000c9848140'/>
  <capability type='scsi_host'>
    <capability type='fc_host'>
```
NOTE

The WWNN and WWPN values must match those in the HBA details seen in the previous step.

The <parent> field specifies the HBA device to associate with this vHBA device. The details in the <device> tag are used in the next step to create a new vHBA device for the host. For more information on the nodedev XML format, see the libvirt upstream pages.

NOTE

The virsh command does not provide a way to define the parent_wwnn, parent_wwpn, or parent_fabric_wwn attributes.

4. Create a VHBA based on the XML file created in the previous step using the virsh nodedev-create command.

```
# virsh nodedev-create vhba_host3
Node device scsi_host5 created from vhba_host3.xml
```

Verification

- Verify the new vHBA’s details (scsi_host5) using the virsh nodedev-dumpxml command:

```
# virsh nodedev-dumpxml scsi_host5
<device>
  <name>scsi_host5</name>
  <path>/sys/devices/pci0000:00/0000:00:04.0/0000:10:00.0/host3/vport-3:0-0/host5</path>
  <parent>scsi_host3</parent>
  <capability type='scsi_host'>
    <host>5</host>
    <unique_id>2</unique_id>
    <capability type='fc_host'>
      <wwnn>5001a4a93526d0a1</wwnn>
      <wwpn>5001a4ace3ee047d</wwpn>
      <fabric_wwn>2002000573de9a81</fabric_wwn>
    </capability>
  </capability>
</device>
```

Additional resources

- Creating SCSI-based storage pools with vHBA devices using the CLI
CHAPTER 15. MANAGING GPU DEVICES IN VIRTUAL MACHINES

To enhance the graphical performance of your virtual machine (VMs) on a RHEL 9 host, you can assign a host GPU to a VM.

- You can detach the GPU from the host and pass full control of the GPU directly to the VM.
- You can create multiple mediated devices from a physical GPU, and assign these devices as virtual GPUs (vGPUs) to multiple guests. This is currently only supported on selected NVIDIA GPUs, and only one mediated device can be assigned to a single guest.

15.1. ASSIGNING A GPU TO A VIRTUAL MACHINE

To access and control GPUs that are attached to the host system, you must configure the host system to pass direct control of the GPU to the virtual machine (VM).

NOTE

If you are looking for information about assigning a virtual GPU, see Managing NVIDIA vGPU devices.

Prerequisites

- You must enable IOMMU support on the host machine kernel.
  - On an Intel host, you must enable VT-d:
    1. Regenerate the GRUB configuration with the `intel_iommu=on` and `iommu=pt` parameters:

       ```bash
       # grubby --args="intel_iommu=on iommu_pt" --update-kernel DEFAULT
       ```
    2. Reboot the host.
  - On an AMD host, you must enable AMD-Vi.
    Note that on AMD hosts, IOMMU is enabled by default, you can add `iommu=pt` to switch it to pass-through mode:
    1. Regenerate the GRUB configuration with the `iommu=pt` parameter:

       ```bash
       # grubby --args="iommu=pt" --update-kernel DEFAULT
       ```

       NOTE

       The `pt` option only enables IOMMU for devices used in pass-through mode and provides better host performance. However, not all hardware supports the option. You can still assign devices irrespective of whether this option is enabled.

    2. Reboot the host.

Procedure
1. Prevent the driver from binding to the GPU.

   a. Identify the PCI bus address to which the GPU is attached.

      # lspci -Dnn | grep VGA
      0000:02:00.0 VGA compatible controller [0300]: NVIDIA Corporation GK106GL [Quadro K4000] [10de:11fa] (rev a1)

   b. Prevent the host’s graphics driver from using the GPU. To do so, use the GPU’s PCI ID with the pci-stub driver.

      For example, the following command prevents the driver from binding to the GPU attached at the 10de:11fa bus:

      # grubby --args="pci-stub.ids=10de:11fa" --update-kernel DEFAULT

   c. Reboot the host.

2. Optional: If certain GPU functions, such as audio, cannot be passed through to the VM due to support limitations, you can modify the driver bindings of the endpoints within an IOMMU group to pass through only the necessary GPU functions.

   a. Convert the GPU settings to XML and note the PCI address of the endpoints that you want to prevent from attaching to the host drivers.

      To do so, convert the GPU’s PCI bus address to a libvirt-compatible format by adding the pci__ prefix to the address, and converting the delimiters to underscores.

      For example, the following command displays the XML configuration of the GPU attached at the 0000:02:00.0 bus address.

      # virsh nodedev-dumpxml pci_0000_02_00_0

      <device>
        <name>pci_0000_02_00_0</name>
        <path>/sys/devices/pci0000:00/0000:00:03.0/0000:02:00.0</path>
        <parent>pci_0000_00_03_0</parent>
        <driver>
          <name>pci-stub</name>
        </driver>
        <capability type="pci">
          <domain>0</domain>
          <bus>2</bus>
          <slot>0</slot>
          <function>0</function>
          <product id='0x11fa'>GK106GL [Quadro K4000]</product>
          <vendor id='0x10de'>NVIDIA Corporation</vendor>
          <iommuGroup number='13'>
            <address domain='0x0000' bus='0x02' slot='0x00' function='0x0'/>
            <address domain='0x0000' bus='0x02' slot='0x00' function='0x1'/>
          </iommuGroup>
          <pci-express>
            <link validity='cap' port='0' speed='8' width='16'/>
            <link validity='sta' speed='2.5' width='16'/>
          </pci-express>
        </capability>
      </device>
b. Prevent the endpoints from attaching to the host driver.
   In this example, to assign the GPU to a VM, prevent the endpoints that correspond to the audio function, <address domain='0x0000' bus='0x02' slot='0x00' function='0x1'/>, from attaching to the host audio driver, and instead attach the endpoints to VFIO-PCI.

   # driverctl set-override 0000:02:00.1 vfio-pci

3. Attach the GPU to the VM

   a. Create an XML configuration file for the GPU by using the PCI bus address.
      For example, you can create the following XML file, GPU-Assign.xml, by using parameters from the GPU’s bus address.

      <hostdev mode='subsystem' type='pci' managed='yes'>
         <driver name='vfio'/>
         <source>
            <address domain='0x0000' bus='0x02' slot='0x00' function='0x0'/>
         </source>
      </hostdev>

   b. Save the file on the host system.

   c. Merge the file with the VM’s XML configuration.
      For example, the following command merges the GPU XML file, GPU-Assign.xml, with the XML configuration file of the System1 VM.

      # virsh attach-device System1 --file /home/GPU-Assign.xml --persistent
      Device attached successfully.

   NOTE

   The GPU is attached as a secondary graphics device to the VM. Assigning a GPU as the primary graphics device is not supported, and Red Hat does not recommend removing the primary emulated graphics device in the VM’s XML configuration.

Verification

- The device appears under the <devices> section in VM’s XML configuration. For more information, see Sample virtual machine XML configuration.

Known Issues

- Attaching an NVIDIA GPU device to a VM that uses a RHEL 9 guest operating system currently disables the Wayland session on that VM, and loads an Xorg session instead. This is because of incompatibilities between NVIDIA drivers and Wayland.

15.2. MANAGING NVIDIA VGPU DEVICES

The vGPU feature makes it possible to divide a physical NVIDIA GPU device into multiple virtual devices, referred to as mediated devices. These mediated devices can then be assigned to multiple virtual machines (VMs) as virtual GPUs. As a result, these VMs can share the performance of a single
Assigning a physical GPU to VMs, with or without using mediated devices, makes it impossible for the host to use the GPU.

### 15.2.1. Setting up NVIDIA vGPU devices

To set up the NVIDIA vGPU feature, you need to download NVIDIA vGPU drivers for your GPU device, create mediated devices, and assign them to the intended virtual machines. For detailed instructions, see below.

#### Prerequisites

- Your GPU supports vGPU mediated devices. For an up-to-date list of NVIDIA GPUs that support creating vGPUs, see the NVIDIA vGPU software documentation.
  - If you do not know which GPU your host is using, install the `lshw` package and use the `lshw -C display` command. The following example shows the system is using an NVIDIA Tesla P4 GPU, compatible with vGPU.

```
# lshw -C display
*-display
  description: 3D controller
  product: GP104GL [Tesla P4]
  vendor: NVIDIA Corporation
  physical id: 0
  bus info: pci@0000:01:00.0
  version: a1
  width: 64 bits
  clock: 33MHz
  capabilities: pm msi pciexpress cap_list
  configuration: driver=vfio-pci latency=0
  resources: irq:16 memory:f6000000-f6ffffff memory:e0000000-efffffff memory:f0000000-f1ffffff
```

#### Procedure

1. Download the NVIDIA vGPU drivers and install them on your system. For instructions, see the NVIDIA documentation.

2. If the NVIDIA software installer did not create the `/etc/modprobe.d/nvidia-installer-disable-nouveau.conf` file, create a `conf` file of any name in `/etc/modprobe.d/`, and add the following lines in the file:

```
blacklist nouveau
options nouveau modeset=0
```

3. Regenerate the initial ramdisk for the current kernel, then reboot.

```
# dracut --force
# reboot
```
4. Check that the kernel has loaded the `nvidia_vgpu_vfio` module and that the `nvidia-vgpu-mgr.service` service is running.

```bash
# lsmod | grep nvidia_vgpu_vfio
nvidia_vgpu_vfio 45011 0
nvidia 14333621 10 nvidia_vgpu_vfio
mdev 20414 2 vfio_mdev,nvidia_vgpu_vfio
vfio 32695 3 vfio_mdev,nvidia_vgpu_vfio,vfio_iommu_type1

# systemctl status nvidia-vgpu-mgr.service
nvidia-vgpu-mgr.service - NVIDIA vGPU Manager Daemon
   Loaded: loaded (/usr/lib/systemd/system/nvidia-vgpu-mgr.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2018-03-16 10:17:36 CET; 5h 8min ago
     Main PID: 1553 (nvidia-vgpu-mgr)
     [...]
```

In addition, if creating vGPU based on an NVIDIA Ampere GPU device, ensure that virtual functions are enable for the physical GPU. For instructions, see the NVIDIA documentation.

5. Generate a device UUID.

```bash
# uuidgen
30820a6f-b1a5-4503-91ca-0c10ba58692a
```

6. Prepare an XML file with a configuration of the mediated device, based on the detected GPU hardware. For example, the following configures a mediated device of the `nvidia-63` vGPU type on an NVIDIA Tesla P4 card that runs on the 0000:01:00.0 PCI bus and uses the UUID generated in the previous step.

```xml
<device>
    <parent>pci_0000_01_00_0</parent>
    <capability type="mdev">
        <type id="nvidia-63"/>
        <uuid>30820a6f-b1a5-4503-91ca-0c10ba58692a</uuid>
    </capability>
</device>
```

7. Define a vGPU mediated device based on the XML file you prepared. For example:

```bash
# virsh nodedev-define vgpu-test.xml
Node device mdev_30820a6f_b1a5_4503-91ca-0c10ba58692a_0000_01_00_0 created from vgpu-test.xml
```

8. Optional: Verify that the mediated device is listed as inactive.

```bash
# virsh nodedev-list --cap mdev --inactive
mdev_30820a6f_b1a5_4503-91ca-0c10ba58692a_0000_01_00_0
```

9. Start the vGPU mediated device you created.

```bash
# virsh nodedev-start mdev_30820a6f_b1a5_4503-91ca-0c10ba58692a_0000_01_00_0
Device mdev_30820a6f_b1a5_4503-91ca-0c10ba58692a_0000_01_00_0 started
```
10. Optional: Ensure that the mediated device is listed as active.

```bash
# virsh nodedev-list --cap mdev
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
```

11. Set the vGPU device to start automatically after the host reboots

```bash
# virsh nodedev-autostart
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Device mdev_d196754e_d8ed_4f43_bf22_684ed698b08b_0000_9b_00_0 marked as autostarted
```

12. Attach the mediated device to a VM that you want to share the vGPU resources. To do so, add the following lines, along with the previously generated UUID, to the `<devices/>` sections in the XML configuration of the VM.

```xml
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci' display='on'>
  <source>
    <address uuid='30820a6f-b1a5-4503-91ca-0c10ba58692a'/>
  </source>
</hostdev>
```

Note that each UUID can only be assigned to one VM at a time. In addition, if the VM does not have QEMU video devices, such as `virtio-vga`, add also the `ramfb='on'` parameter on the `<hostdev>` line.

13. For full functionality of the vGPU mediated devices to be available on the assigned VMs, set up NVIDIA vGPU guest software licensing on the VMs. For further information and instructions, see the [NVIDIA Virtual GPU Software License Server User Guide](#).

**Verification**

1. Query the capabilities of the vGPU you created, and ensure it is listed as active and persistent.

```bash
# virsh nodedev-info mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Name:    virsh nodedev-autostart
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Parent:  pci_0000_01_00_0
Active:  yes
Persistent: yes
Autostart: yes
```

2. Start the VM and verify that the guest operating system detects the mediated device as an NVIDIA GPU. For example, if the VM uses Linux:

```bash
# lspci -d 10de: -k
07:00.0 VGA compatible controller: NVIDIA Corporation GV100GL [Tesla V100 SXM2 32GB]
  (rev a1)
    Subsystem: NVIDIA Corporation Device 12ce
    Kernel driver in use: nvidia
    Kernel modules: nouveau, nvidia_drm, nvidia
```

**Known Issues**

...
Assigning an NVIDIA vGPU mediated device to a VM that uses a RHEL 9 guest operating system currently disables the Wayland session on that VM, and loads an Xorg session instead. This is because of incompatibilities between NVIDIA drivers and Wayland.

Additional resources

- NVIDIA vGPU software documentation
- The `man virsh` command

15.2.2. Removing NVIDIA vGPU devices

To change the configuration of assigned vGPU mediated devices, you need to remove the existing devices from the assigned VMs. For instructions, see below:

Prerequisites

- The VM from which you want to remove the device is shut down.

Procedure

1. Obtain the ID of the mediated device that you want to remove.

   ```
   # virsh nodedev-list --cap mdev
   mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
   ```

2. Stop the running instance of the vGPU mediated device.

   ```
   # virsh nodedev-destroy
   mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
   Destroyed node device 'mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0'
   ```

3. Optional: Ensure the mediated device has been deactivated.

   ```
   # virsh nodedev-info mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
   Name:        virsh nodedev-autostart
   mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
   Parent:     pci_0000_01_00_0
   Active:      no
   Persistent:  yes
   Autostart:   yes
   ```

4. Remove the device from the XML configuration of the VM. To do so, use the `virsh edit` utility to edit the XML configuration of the VM, and remove the mdev’s configuration segment. The segment will look similar to the following:

   ```xml
   <hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci'>
   <source>
   <address uuid="30820a6f-b1a5-4503-91ca-0c10ba58692a"/>
   </source>
   </hostdev>
   ```

Note that stopping and detaching the mediated device does not delete it, but rather keeps it as defined. As such, you can `restart` and `attach` the device to a different VM.
5. **Optional:** To delete the stopped mediated device, remove its definition.

```bash
# virsh nodedev-undefine
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Undefined node device 'mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0'
```

**Verification**

- If you only stopped and detached the device, ensure the mediated device is listed as inactive.

```bash
# virsh nodedev-list --cap mdev --inactive
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
```

- If you also deleted the device, ensure the following command does not display it.

```bash
# virsh nodedev-list --cap mdev
```

**Additional resources**

- The `man virsh` command

### 15.2.3. Obtaining NVIDIA vGPU information about your system

To evaluate the capabilities of the vGPU features available to you, you can obtain additional information about the mediated devices on your system, such as:

- How many mediated devices of a given type can be created
- What mediated devices are already configured on your system.

**Procedure**

- To see the available GPUs devices on your host that can support vGPU mediated devices, use the `virsh nodedev-list --cap mdev_types` command. For example, the following shows a system with two NVIDIA Quadro RTX6000 devices.

```bash
# virsh nodedev-list --cap mdev_types
pci_0000_5b_00_0
pci_0000_9b_00_0
```

- To display vGPU types supported by a specific GPU device, as well as additional metadata, use the `virsh nodedev-dumpxml` command.

```xml
# virsh nodedev-dumpxml pci_0000_9b_00_0
<device>
  <name>pci_0000_9b_00_0</name>
  <path>/sys/devices/pci0000:9a/0000:9a:00.0/0000:9b:00.0</path>
  <parent>pci_0000_9a_00_0</parent>
  <driver>
    <name>nvidia</name>
  </driver>
  <capability type='pci'>
    <class>0x030000</class>
```

---

**CHAPTER 15. MANAGING GPU DEVICES IN VIRTUAL MACHINES**

---

173
Additional resources

- The **man virsh** command

15.2.4. Remote desktop streaming services for NVIDIA vGPU

The following remote desktop streaming services are supported on the RHEL 9 hypervisor with NVIDIA vGPU or NVIDIA GPU passthrough enabled:

- HP ZCentral Remote Boost/Teradici
- NICE DCV
• Mechdyne TGX

For support details, see the appropriate vendor support matrix.

15.2.5. Additional resources

• NVIDIA vGPU software documentation
CHAPTER 16. CONFIGURING VIRTUAL MACHINE NETWORK CONNECTIONS

For your virtual machines (VMs) to connect over a network to your host, to other VMs on your host, and to locations on an external network, the VM networking must be configured accordingly. To provide VM networking, the RHEL 9 hypervisor and newly created VMs have a default network configuration, which can also be modified further. For example:

- You can enable the VMs on your host to be discovered and connected to by locations outside the host, as if the VMs were on the same network as the host.
- You can partially or completely isolate a VM from inbound network traffic to increase its security and minimize the risk of any problems with the VM impacting the host.

The following sections explain the various types of VM network configuration and provide instructions for setting up selected VM network configurations.

16.1. UNDERSTANDING VIRTUAL NETWORKING

The connection of virtual machines (VMs) to other devices and locations on a network has to be facilitated by the host hardware. The following sections explain the mechanisms of VM network connections and describe the default VM network setting.

16.1.1. How virtual networks work

Virtual networking uses the concept of a virtual network switch. A virtual network switch is a software construct that operates on a host machine. VMs connect to the network through the virtual network switch. Based on the configuration of the virtual switch, a VM can use an existing virtual network managed by the hypervisor, or a different network connection method.

The following figure shows a virtual network switch connecting two VMs to the network:

From the perspective of a guest operating system, a virtual network connection is the same as a physical network connection. Host machines view virtual network switches as network interfaces. When the virtnetwordk service is first installed and started, it creates virbr0, the default network interface for VMs.

To view information about this interface, use the ip utility on the host.
By default, all VMs on a single host are connected to the same **NAT-type** virtual network, named **default**, which uses the **virbr0** interface. For details, see Virtual networking default configuration.

For basic outbound-only network access from VMs, no additional network setup is usually needed, because the default network is installed along with the **libvirt-daemon-config-network** package, and is automatically started when the **virtnetworkd** service is started.

If a different VM network functionality is needed, you can create additional virtual networks and network interfaces and configure your VMs to use them. In addition to the default NAT, these networks and interfaces can be configured to use one of the following modes:

- Routed mode
- Bridged mode
- Isolated mode
- Open mode

### 16.1.2. Virtual networking default configuration

When the **virtnetworkd** service is first installed on a virtualization host, it contains an initial virtual network configuration in network address translation (NAT) mode. By default, all VMs on the host are connected to the same **libvirt** virtual network, named **default**. VMs on this network can connect to locations both on the host and on the network beyond the host, but with the following limitations:

- VMs on the network are visible to the host and other VMs on the host, but the network traffic is affected by the firewalls in the guest operating system’s network stack and by the **libvirt** network filtering rules attached to the guest interface.

  - VMs on the network can connect to locations outside the host but are not visible to them. Outbound traffic is affected by the NAT rules, as well as the host system’s firewall.

The following diagram illustrates the default VM network configuration:
16.2. USING THE WEB CONSOLE FOR MANAGING VIRTUAL MACHINE NETWORK INTERFACES

Using the RHEL 9 web console, you can manage the virtual network interfaces for the virtual machines to which the web console is connected. You can:

- View information about network interfaces and edit them.
- Add network interfaces to virtual machines, and disconnect or delete the interfaces.

16.2.1. Viewing and editing virtual network interface information in the web console

Using the RHEL 9 web console, you can view and modify the virtual network interfaces on a selected virtual machine (VM):

**Prerequisites**

- The web console VM plug-in is installed on your system.

**Procedure**

1. In the Virtual Machines interface, click the VM whose information you want to see. A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM’s graphical interface.

2. Scroll to Network Interfaces. The Networks Interfaces section displays information about the virtual network interface configured for the VM as well as options to Add, Delete, Edit, or Unplug network interfaces.
The information includes the following:

- **Type** - The type of network interface for the VM. The types include virtual network, bridge to LAN, and direct attachment.

  **NOTE**
  
  Generic Ethernet connection is not supported in RHEL 9 and later.

- **Model type** - The model of the virtual network interface.

- **MAC Address** - The MAC address of the virtual network interface.

- **IP Address** - The IP address of the virtual network interface.

- **Source** - The source of the network interface. This is dependent on the network type.

- **State** - The state of the virtual network interface.

3. To edit the virtual network interface settings, Click **Edit**. The Virtual Network Interface Settings dialog opens.

4. Change the interface type, source, model, or MAC address.

5. Click **Save**. The network interface is modified.

**NOTE**

Changes to the virtual network interface settings take effect only after restarting the VM.

Additionally, MAC address can only be modified when the VM is shut off.

Additional resources
16.2.2. Adding and connecting virtual network interfaces in the web console

Using the RHEL 9 web console, you can create a virtual network interface and connect a virtual machine (VM) to it.

Prerequisites

- The web console VM plug-in is installed on your system.

Procedure

1. In the Virtual Machines interface, click the VM whose information you want to see. A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM's graphical interface.

2. Scroll to Network Interfaces. The Networks Interfaces section displays information about the virtual network interface configured for the VM as well as options to Add, Delete, Edit, or Plug network interfaces.

3. Click Plug in the row of the virtual network interface you want to connect. The selected virtual network interface connects to the VM.

16.2.3. Disconnecting and removing virtual network interfaces in the web console

Using the RHEL 9 web console, you can disconnect the virtual network interfaces connected to a selected virtual machine (VM).

Prerequisites

- The web console VM plug-in is installed on your system.

Procedure

1. In the Virtual Machines interface, click the VM whose information you want to see. A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM's graphical interface.

2. Scroll to Network Interfaces. The Networks Interfaces section displays information about the virtual network interface configured for the VM as well as options to Add, Delete, Edit, or Unplug network interfaces.

3. Click Unplug in the row of the virtual network interface you want to disconnect. The selected virtual network interface disconnects from the VM.

16.3. RECOMMENDED VIRTUAL MACHINE NETWORKING CONFIGURATIONS USING THE COMMAND-LINE INTERFACE
In many scenarios, the default VM networking configuration is sufficient. However, if adjusting the configuration is required, you can use the command-line interface (CLI) to do so. The following sections describe selected VM network setups for such situations.

16.3.1. Configuring externally visible virtual machines using the command-line interface

By default, a newly created VM connects to a NAT-type network that uses `virbr0`, the default virtual bridge on the host. This ensures that the VM can use the host’s network interface controller (NIC) for connecting to outside networks, but the VM is not reachable from external systems.

If you require a VM to appear on the same external network as the hypervisor, you must use bridged mode instead. To do so, attach the VM to a bridge device connected to the hypervisor’s physical network device. To use the command-line interface for this, follow the instructions below.

Prerequisites

- A shut-down existing VM with the default NAT setup.
- The IP configuration of the hypervisor. This varies depending on the network connection of the host. As an example, this procedure uses a scenario where the host is connected to the network using an ethernet cable, and the hosts’ physical NIC MAC address is assigned to a static IP on a DHCP server. Therefore, the ethernet interface is treated as the hypervisor IP. To obtain the IP configuration of the ethernet interface, use the `ip addr` utility:

```bash
# ip addr
[...]
enp0s25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
  link/ether 54:ee:75:49:dc:46 brd ff:ff:ff:ff:ff:ff
  inet 10.0.0.148/24 brd 10.0.0.255 scope global dynamic noprefixroute
enp0s25
```

Procedure

1. Create and set up a bridge connection for the physical interface on the host. For instructions, see the Configuring a network bridge.
   Note that in a scenario where static IP assignment is used, you must move the IPv4 setting of the physical ethernet interface to the bridge interface.

2. Modify the VM’s network to use the created bridged interface. For example, the following sets `testguest` to use `bridge0`.

```bash
# virt-xml testguest --edit --network bridge=bridge0
Domain 'testguest' defined successfully.
```

3. Start the VM.

```bash
# virsh start testguest
```

4. In the guest operating system, adjust the IP and DHCP settings of the system’s network interface as if the VM was another physical system in the same network as the hypervisor. The specific steps for this will differ depending on the guest OS used by the VM. For example, if the guest OS is RHEL 9, see Configuring an Ethernet connection.
Verification

1. Ensure the newly created bridge is running and contains both the host’s physical interface and the interface of the VM.

```bash
# ip link show master bridge0
2: enp0s25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master bridge0 state UP mode DEFAULT group default qlen 1000
   link/ether 54:ee:75:49:dc:46 brd ff:ff:ff:ff:ff:ff
10: vnet0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master bridge0 state UNKNOWN mode DEFAULT group default qlen 1000
   link/ether fe:54:00:89:15:40 brd ff:ff:ff:ff:ff:ff
```

2. Ensure the VM appears on the same external network as the hypervisor:

   a. In the guest operating system, obtain the network ID of the system. For example, if it is a Linux guest:

```bash
# ip addr
[...]
enp0s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
   link/ether 52:54:00:09:15:46 brd ff:ff:ff:ff:ff:ff
   inet 10.0.0.150/24 brd 10.0.0.255 scope global dynamic noprefixroute enp0s0
```

   b. From an external system connected to the local network, connect to the VM using the obtained ID.

```bash
# ssh root@10.0.0.150
root@10.0.0.150's password:
Last login: Mon Sep 24 12:05:36 2019
root~#
```

If the connection works, the network has been configured successfully.

Troubleshooting

- In certain situations, such as when using a client-to-site VPN while the VM is hosted on the client, using bridged mode for making your VMs available to external locations is not possible. To work around this problem, you can set destination NAT using nftables for the VM.

Additional resources

- Configuring externally visible virtual machines using the web console
- Virtual networking in bridged mode

16.3.2. Isolating virtual machines from each other using the command-line interface

To prevent a virtual machine (VM) from communicating with other VMs on your host, for example to avoid data sharing or to increase system security, you can completely isolate the VM from host-side network traffic.

By default, a newly created VM connects to a NAT-type network that uses virbr0, the default virtual bridge on the host. This ensures that the VM can use the host’s NIC for connecting to outside networks,
as well as to other VMs on the host. This is a generally secure connection, but in some cases, connectivity
to the other VMs may be a security or data privacy hazard. In such situations, you can isolate the VM by
using direct `macvtap` connection in private mode instead of the default network.

In private mode, the VM is visible to external systems and can receive a public IP on the host’s subnet,
but the VM and the host cannot access each other, and the VM is also not visible to other VMs on the
host.

For instructions to set up `macvtap` private mode on your VM using the CLI, see below.

**Prerequisites**

- An existing VM with the default NAT setup.
- The name of the host interface that you want to use for the `macvtap` connection. The interface
  you must select will vary depending on your use case and the network configuration on your
  host. As an example, this procedure uses the host’s physical ethernet interface.

To obtain the name of the targeted interface:

```
$ ip addr
[...]
2: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel
state DOWN group default qlen 1000
  link/ether 54:e1:ad:42:70:45 brd ff:ff:ff:ff:ff:ff
[...]
```

**Procedure**

- Use the selected interface to set up private `macvtap` on the selected VM. The following
  example configures `macvtap` in private mode on the `enp0s31f6` interface for the VM named
  `panic-room`.

```
# virt-xml panic-room --edit --network
type=direct,source=enp0s31f6,source.mode=private
Domain 'panic-room' XML defined successfully
```

**Verification**

1. Start the updated VM.

```
# virsh start panic-room
Domain 'panic-room' started
```

2. List the interface statistics for the VM.

```
# virsh domstats panic-room --interface
Domain: 'panic-room'
  net.count=1
  net.0.name=macvtap0
  net.0.rx.bytes=0
  net.0.rx.pkts=0
  net.0.rx.errs=0
  net.0.rx.drop=0
  net.0.tx.bytes=0
```
Additional resources

- Isolating virtual machines from each other using the web console
- Using macvtap in private mode
- Securing virtual machines

16.4. RECOMMENDED VIRTUAL MACHINE NETWORKING CONFIGURATIONS USING THE WEB CONSOLE

In many scenarios, the default VM networking configuration is sufficient. However, if adjusting the configuration is required, you can use the RHEL 9 web console to do so. The following sections describe selected VM network setups for such situations.

16.4.1. Configuring externally visible virtual machines using the web console

By default, a newly created VM connects to a NAT-type network that uses `virbr0`, the default virtual bridge on the host. This ensures that the VM can use the host's network interface controller (NIC) for connecting to outside networks, but the VM is not reachable from external systems.

If you require a VM to appear on the same external network as the hypervisor, you must use **bridged mode** instead. To do so, attach the VM to a bridge device connected to the hypervisor's physical network device. To use the RHEL 9 web console for this, follow the instructions below.

Prerequisites

- The web console VM plug-in is installed on your system.
- A shut-down existing VM with the default NAT setup.
- The IP configuration of the hypervisor. This varies depending on the network connection of the host. As an example, this procedure uses a scenario where the host is connected to the network using an ethernet cable, and the hosts' physical NIC MAC address is assigned to a static IP on a DHCP server. Therefore, the ethernet interface is treated as the hypervisor IP.

To obtain the IP configuration of the ethernet interface, go to the **Networking** tab in the web console, and see the **Interfaces** section.

Procedure

1. Create and set up a bridge connection for the physical interface on the host. For instructions, see **Configuring network bridges in the web console**.
   Note that in a scenario where static IP assignment is used, you must move the IPv4 setting of the physical ethernet interface to the bridge interface.

2. Modify the VM's network to use the bridged interface. In the **Network Interfaces** tab of the VM:
   a. Click **Add Network Interface**
b. In the **Add Virtual Network Interface** dialog, set:
   - **Interface Type** to **Bridge to LAN**
   - Source to the newly created bridge, for example **bridge0**

c. Click **Add**

d. **Optional**: Click **Unplug** for all the other interfaces connected to the VM.

3. Click **Run** to start the VM.

4. In the guest operating system, adjust the IP and DHCP settings of the system’s network interface as if the VM was another physical system in the same network as the hypervisor. The specific steps for this will differ depending on the guest OS used by the VM. For example, if the guest OS is RHEL 9, see [Configuring an Ethernet connection](#).

**Verification**

1. In the **Networking** tab of the host’s web console, click the row with the newly created bridge to ensure it is running and contains both the host’s physical interface and the interface of the VM.

2. Ensure the VM appears on the same external network as the hypervisor.

   a. In the guest operating system, obtain the network ID of the system. For example, if it is a Linux guest:

      ```
      # ip addr
      [...] enp0s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
      link/ether 52:54:00:09:15:46 brd ff:ff:ff:ff:ff:ff
      inet 10.0.0.150/24 brd 10.0.0.255 scope global dynamic noprefixroute enp0s0
      ```

   b. From an external system connected to the local network, connect to the VM using the obtained ID.

      ```
      # ssh root@10.0.0.150
      root@110.34.5.18's password: 
      Last login: Mon Sep 24 12:05:36 2019
      root~#*
      ```

      If the connection works, the network has been configured successfully.

**Troubleshooting**

- In certain situations, such as when using a client-to-site VPN while the VM is hosted on the client, using bridged mode for making your VMs available to external locations is not possible.

**Additional resources**

- [Configuring externally visible virtual machines using the command-line interface](#)
- [Virtual networking in bridged mode](#)
16.4.2. Isolating virtual machines from each other using the web console

To prevent a virtual machine (VM) from communicating with other VMs on your host, for example to avoid data sharing or to increase system security, you can completely isolate the VM from host-side network traffic.

By default, a newly created VM connects to a NAT-type network that uses `virbr0`, the default virtual bridge on the host. This ensures that the VM can use the host’s NIC for connecting to outside networks, as well as to other VMs on the host. This is a generally secure connection, but in some cases, connectivity to the other VMs may be a security or data privacy hazard. In such situations, you can isolate the VM by using direct `macvtap` connection in private mode instead of the default network.

In private mode, the VM is visible to external systems and can receive a public IP on the host’s subnet, but the VM and the host cannot access each other, and the VM is also not visible to other VMs on the host.

For instructions to set up `macvtap` private mode on your VM using the web console, see below.

Prerequisites

- The web console VM plug-in is installed on your system.
- An existing VM with the default NAT setup.

Procedure

1. In the Virtual Machines pane, click the row with the virtual machine you want to isolate. A pane with the basic information about the VM opens.
2. Click the Network Interfaces tab.
3. Click Edit. The Virtual Machine Interface Settings dialog opens.
4. Set Interface Type to Direct Attachment.
5. Set Source to the host interface of your choice. Note that the interface you select will vary depending on your use case and the network configuration on your host.

Verification

1. Start the VM by clicking Run.
2. In the Terminal pane of the web console, list the interface statistics for the VM. For example, to view the network interface traffic for the `panic-room` VM:

   ```
   # virsh domstats panic-room --interface
   Domain: 'panic-room'
   net.count=1
   net.0.name=macvtap0
   net.0.rx.bytes=0
   net.0.rx.pkts=0
   net.0.rx.errs=0
   net.0.rx.drop=0
   net.0.tx.bytes=0
   ```

Red Hat Enterprise Linux 9 Configuring and managing virtualization

186
Additional resources

- Isolating virtual machines from each other using the command-line interface
- Using macvtap in private mode
- Securing virtual machines

16.5. TYPES OF VIRTUAL MACHINE NETWORK CONNECTIONS

To modify the networking properties and behavior of your VMs, change the type of virtual network or interface the VMs use. The following sections describe the connection types available to VMs in RHEL 9.

16.5.1. Virtual networking with network address translation

By default, virtual network switches operate in network address translation (NAT) mode. They use IP masquerading rather than Source-NAT (SNAT) or Destination-NAT (DNAT). IP masquerading enables connected VMs to use the host machine’s IP address for communication with any external network. When the virtual network switch is operating in NAT mode, computers external to the host cannot communicate with the VMs inside the host.

**WARNING**

Virtual network switches use NAT configured by firewall rules. Editing these rules while the switch is running is not recommended, because incorrect rules may result in the switch being unable to communicate.
16.5.2. Virtual networking in routed mode

When using *Routed* mode, the virtual switch connects to the physical LAN connected to the host machine, passing traffic back and forth without the use of NAT. The virtual switch can examine all traffic and use the information contained within the network packets to make routing decisions. When using this mode, the virtual machines (VMs) are all in a single subnet, separate from the host machine. The VM subnet is routed through a virtual switch, which exists on the host machine. This enables incoming connections, but requires extra routing-table entries for systems on the external network.

Routed mode uses routing based on the IP address:

![Diagram showing virtual networking in routed mode](image)

Common topologies that use routed mode include DMZs and virtual server hosting.

**DMZ**

You can create a network where one or more nodes are placed in a controlled sub-network for security reasons. Such a sub-network is known as a demilitarized zone (DMZ).

![Diagram showing DMZ topology](image)

Host machines in a DMZ typically provide services to WAN (external) host machines as well as LAN (internal) host machines. Since this requires them to be accessible from multiple locations, and considering that these locations are controlled and operated in different ways based on their security and trust level, routed mode is the best configuration for this environment.

**Virtual server hosting**
A virtual server hosting provider may have several host machines, each with two physical network connections. One interface is used for management and accounting, the other for the VMs to connect through. Each VM has its own public IP address, but the host machines use private IP addresses so that only internal administrators can manage the VMs.

16.5.3. Virtual networking in bridged mode

In most VM networking modes, VMs automatically create and connect to the virbr0 virtual bridge. In contrast, in bridged mode, the VM connects to an existing Linux bridge on the host. As a result, the VM is directly visible on the physical network. This enables incoming connections, but does not require any extra routing-table entries.

Bridged mode uses connection switching based on the MAC address:

In bridged mode, the VM appear within the same subnet as the host machine. All other physical machines on the same physical network can detect the VM and access it.

Bridged network bonding

It is possible to use multiple physical bridge interfaces on the hypervisor by joining them together with a bond. The bond can then be added to a bridge, after which the VMs can be added to the bridge as well. However, the bonding driver has several modes of operation, and not all of these modes work with a bridge where VMs are in use.
The following bonding modes are usable:

- mode 1
- mode 2
- mode 4

In contrast, using modes 0, 3, 5, or 6 is likely to cause the connection to fail. Also note that media-independent interface (MII) monitoring should be used to monitor bonding modes, as Address Resolution Protocol (ARP) monitoring does not work correctly.

For more information on bonding modes, refer to the Red Hat Knowledgebase.

Common scenarios
The most common use cases for bridged mode include:

- Deploying VMs in an existing network alongside host machines, making the difference between virtual and physical machines invisible to the end user.
- Deploying VMs without making any changes to existing physical network configuration settings.
- Deploying VMs that must be easily accessible to an existing physical network. Placing VMs on a physical network where they must access DHCP services.
- Connecting VMs to an existing network where virtual LANs (VLANs) are used.

Additional resources

- Configuring externally visible virtual machines using the command-line interface
- Configuring externally visible virtual machines using the web console
- Explanation of bridge_opts parameters

16.5.4. Virtual networking in isolated mode

When using isolated mode, virtual machines connected to the virtual switch can communicate with each other and with the host machine, but their traffic will not pass outside of the host machine, and they cannot receive traffic from outside the host machine. Using dnsmasq in this mode is required for basic functionality such as DHCP.
16.5.5. Virtual networking in open mode

When using open mode for networking, libvirt does not generate any firewall rules for the network. As a result, libvirt does not overwrite firewall rules provided by the host, and the user can therefore manually manage the VM’s firewall rules.

16.5.6. Direct attachment of the virtual network device

You can use the macvtap driver to attach a virtual machine’s NIC directly to a specified physical interface of the host machine. The macvtap connection has a number of modes, including **private mode**.

In this mode, all packets are sent to the external switch and will only be delivered to a target VM on the same host machine if they are sent through an external router or gateway and these send them back to the host. Private mode can be used to prevent the individual VMs on a single host from communicating with each other.

---

**Additional resources**

- Isolating virtual machines from each other using the command-line interface
- Isolating virtual machines from each other using the web console

**16.5.7. Comparison of virtual machine connection types**
The following table provides information about the locations to which selected types of virtual machine (VM) network configurations can connect, and to which they are visible.

Table 16.1. Virtual machine connection types

<table>
<thead>
<tr>
<th></th>
<th>Connection to the host</th>
<th>Connection to other VMs on the host</th>
<th>Connection to outside locations</th>
<th>Visible to outside locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bridged mode</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>NAT</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>no</td>
</tr>
<tr>
<td>Routed mode</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Isolated mode</td>
<td>YES</td>
<td>YES</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Private mode</td>
<td>no</td>
<td>no</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Open mode</td>
<td></td>
<td>Depend on the host’s firewall rules</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

16.6. BOOTING VIRTUAL MACHINES FROM A PXE SERVER

Virtual machines (VMs) that use Preboot Execution Environment (PXE) can boot and load their configuration from a network. This chapter describes how to use `libvirt` to boot VMs from a PXE server on a virtual or bridged network.

**WARNING**

These procedures are provided only as an example. Ensure that you have sufficient backups before proceeding.

16.6.1. Setting up a PXE boot server on a virtual network

This procedure describes how to configure a `libvirt` virtual network to provide Preboot Execution Environment (PXE). This enables virtual machines on your host to be configured to boot from a boot image available on the virtual network.

**Prerequisites**

- A local PXE server (DHCP and TFTP), such as:
  - `libvirt` internal server
  - manually configured `dhcpd` and `tftpd`
  - `dnsmasq`
Cobbler server

- PXE boot images, such as PXELINUX configured by Cobbler or manually.

Procedure

1. Place the PXE boot images and configuration in `/var/lib/tftpboot` folder.

2. Set folder permissions:

   ```bash
   # chmod -R a+r /var/lib/tftpboot
   ```

3. Set folder ownership:

   ```bash
   # chown -R nobody: /var/lib/tftpboot
   ```

4. Update SELinux context:

   ```bash
   # chcon -R --reference /usr/sbin/dnsmasq /var/lib/tftpboot
   # chcon -R --reference /usr/libexec/libvirt_leaseshelper /var/lib/tftpboot
   ```

5. Shut down the virtual network:

   ```bash
   # virsh net-destroy default
   ```

6. Open the virtual network configuration file in your default editor:

   ```bash
   # virsh net-edit default
   ```

7. Edit the `<ip>` element to include the appropriate address, network mask, DHCP address range, and boot file, where `BOOT_FILENAME` is the name of the boot image file.

   ```xml
   <ip address='192.168.122.1' netmask='255.255.255.0'>
   <tftp root='/var/lib/tftpboot' />
   <dhcp>
   <range start='192.168.122.2' end='192.168.122.254' />
   <bootp file='BOOT_FILENAME' />
   </dhcp>
   </ip>
   ```

8. Start the virtual network:

   ```bash
   # virsh net-start default
   ```

Verification

- Verify that the `default` virtual network is active:

   ```bash
   # virsh net-list
   Name         State Autostart Persistent
   ------------------------------
   default      active no no
   ```
Additional resources

- Configuring TFTP and DHCP on a PXE server

16.6.2. Booting virtual machines using PXE and a virtual network

To boot virtual machines (VMs) from a Preboot Execution Environment (PXE) server available on a virtual network, you must enable PXE booting.

Prerequisites

- A PXE boot server is set up on the virtual network as described in Setting up a PXE boot server on a virtual network.

Procedure

- Create a new VM with PXE booting enabled. For example, to install from a PXE, available on the default virtual network, into a new 10 GB qcow2 image file:

  ```
  # virt-install --pxe --network network=default --memory 2048 --vcpus 2 --disk size=10
  ```

  Alternatively, you can manually edit the XML configuration file of an existing VM:

  i. Ensure the `<os>` element has a `<boot dev='network'/>` element inside:

  ```
  <os>
    <type arch='x86_64' machine='pc-i440fx-rhel7.0.0'>hvm</type>
    <boot dev='network'/>
    <boot dev='hd'/>
  </os>
  ```

  ii. Ensure the guest network is configured to use your virtual network:

  ```
  <interface type='network'>
    <mac address='52:54:00:66:79:14'/>
    <source network='default'/>
    <target dev='vnet0'/>
    <alias name='net0'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
  </interface>
  ```

Verification

- Start the VM using the `virsh start` command. If PXE is configured correctly, the VM boots from a boot image available on the PXE server.

16.6.3. Booting virtual machines using PXE and a bridged network

To boot virtual machines (VMs) from a Preboot Execution Environment (PXE) server available on a bridged network, you must enable PXE booting.

Prerequisites

- Network bridging is enabled.
• A PXE boot server is available on the bridged network.

Procedure

• Create a new VM with PXE booting enabled. For example, to install from a PXE, available on the `breth0` bridged network, into a new 10 GB qcow2 image file:

```bash
# virt-install --pxe --network bridge=breth0 --memory 2048 --vcpus 2 --disk size=10
```

• Alternatively, you can manually edit the XML configuration file of an existing VM:

  i. Ensure the `<os>` element has a `<boot dev='network'/>` element inside:

```xml
<os>
  <type arch='x86_64' machine='pc-i440fx-rhel7.0.0'>hvm</type>
  <boot dev='network'/>
  <boot dev='hd'/>
</os>
```

  ii. Ensure the VM is configured to use your bridged network:

```xml
<interface type='bridge'>
  <mac address='52:54:00:5a:ad:cb'/>
  <source bridge='breth0'/>
  <target dev='vnet0'/>
  <alias name='net0'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

Verification

• Start the VM using the `virsh start` command. If PXE is configured correctly, the VM boots from a boot image available on the PXE server.

Additional resources

• Configuring a network bridge

16.7. ADDITIONAL RESOURCES

• Configuring and managing networking

• Attach specific network interface cards as SR-IOV devices to increase VM performance.
CHAPTER 17. OPTIMIZING VIRTUAL MACHINE PERFORMANCE

Virtual machines (VMs) always experience some degree of performance deterioration in comparison to the host. The following sections explain the reasons for this deterioration and provide instructions on how to minimize the performance impact of virtualization in RHEL 9, so that your hardware infrastructure resources can be used as efficiently as possible.

17.1. WHAT INFLUENCES VIRTUAL MACHINE PERFORMANCE

VMs are run as user-space processes on the host. The hypervisor therefore needs to convert the host’s system resources so that the VMs can use them. As a consequence, a portion of the resources is consumed by the conversion, and the VM therefore cannot achieve the same performance efficiency as the host.

The impact of virtualization on system performance

More specific reasons for VM performance loss include:

- Virtual CPUs (vCPUs) are implemented as threads on the host, handled by the Linux scheduler.
- VMs do not automatically inherit optimization features, such as NUMA or huge pages, from the host kernel.
- Disk and network I/O settings of the host might have a significant performance impact on the VM.
- Network traffic typically travels to a VM through a software-based bridge.
- Depending on the host devices and their models, there might be significant overhead due to emulation of particular hardware.

The severity of the virtualization impact on the VM performance is influenced by a variety factors, which include:

- The number of concurrently running VMs.
- The amount of virtual devices used by each VM.
- The device types used by the VMs.

Reducing VM performance loss

RHEL 9 provides a number of features you can use to reduce the negative performance effects of virtualization. Notably:

- The **tuned** service can automatically optimize the resource distribution and performance of your VMs.
- **Block I/O tuning** can improve the performances of the VM’s block devices, such as disks.
- **NUMA tuning** can increase vCPU performance.
- **Virtual networking** can be optimized in various ways.

**IMPORTANT**

Tuning VM performance can have adverse effects on other virtualization functions. For example, it can make migrating the modified VM more difficult.
17.2. OPTIMIZING VIRTUAL MACHINE PERFORMANCE USING TUNED

The tuned utility is a tuning profile delivery mechanism that adapts RHEL for certain workload characteristics, such as requirements for CPU-intensive tasks or storage-network throughput responsiveness. It provides a number of tuning profiles that are pre-configured to enhance performance and reduce power consumption in a number of specific use cases. You can edit these profiles or create new profiles to create performance solutions tailored to your environment, including virtualized environments.

To optimize RHEL 9 for virtualization, use the following profiles:

- For RHEL 9 virtual machines, use the virtual-guest profile. It is based on the generally applicable throughput-performance profile, but also decreases the swappiness of virtual memory.
- For RHEL 9 virtualization hosts, use the virtual-host profile. This enables more aggressive writeback of dirty memory pages, which benefits the host performance.

Prerequisites

- The tuned service is installed and enabled.

Procedure

To enable a specific tuned profile:

1. List the available tuned profiles.

   ```bash
   # tuned-adm list
   ``

   Available profiles:
   - balanced - General non-specialized tuned profile
   - desktop - Optimize for the desktop use-case
   [...]
   - virtual-guest - Optimize for running inside a virtual guest
   - virtual-host - Optimize for running KVM guests
   Current active profile: balanced

2. Optional: Create a new tuned profile or edit an existing tuned profile. For more information, see Customizing tuned profiles.

3. Activate a tuned profile.

   ```bash
   # tuned-adm profile selected-profile
   ```

   - To optimize a virtualization host, use the virtual-host profile.

     ```bash
     # tuned-adm profile virtual-host
     ```

   - On a RHEL guest operating system, use the virtual-guest profile.

     ```bash
     # tuned-adm profile virtual-guest
     ```

Additional resources
17.3. OPTIMIZING LIBVIRT DAEMONS

The **libvirt** virtualization suite works as a management layer for the RHEL hypervisor, and your **libvirt** configuration significantly impacts your virtualization host. Notably, RHEL 9 contains two different types of **libvirt** daemons, monolithic or modular, and which type of daemons you use affects how granularly you can configure individual virtualization drivers.

17.3.1. Types of libvirt daemons

RHEL 9 supports the following **libvirt** daemon types:

**Monolithic libvirt**

The traditional **libvirt** daemon, **libvirtd**, controls a wide variety of virtualization drivers, using a single configuration file - `/etc/libvirt/libvirtd.conf`. As such, **libvirtd** allows for centralized hypervisor configuration, but may use system resources inefficiently. Therefore, **libvirtd** will become unsupported in a future major release of RHEL.

However, if you updated to RHEL 9 from RHEL 8, your host still uses **libvirtd** by default.

**Modular libvirt**

Newly introduced in RHEL 9, modular **libvirt** provides a specific daemon for each virtualization driver. These include the following:

- **virtqemud** - A primary daemon for hypervisor management
- **virtinterfaced** - A secondary daemon for host NIC management
- **virtnetworkd** - A secondary daemon for virtual network management
- **virtnodedevd** - A secondary daemon for host physical device management
- **virtnwfilterd** - A secondary daemon for host firewall management
- **virtsecretd** - A secondary daemon for host secret management
- **virtstoraged** - A secondary daemon for storage management

Each of the daemons has a separate configuration file - for example `/etc/libvirt/virtqemud.conf`. As such, modular **libvirt** daemons provide better options for fine-tuning **libvirt** resource management.

If you performed a fresh install of RHEL 9, modular **libvirt** is configured by default.

**Next steps**

- If your RHEL 9 uses **libvirtd**, Red Hat recommends switching to modular daemons. For instructions, see [Enabling modular libvirt daemons](#).

17.3.2. Enabling modular libvirt daemons

In RHEL 9, the **libvirt** library uses modular daemons that handle individual virtualization driver sets on your host. For example, the **virtqemud** daemon handles QEMU drivers.
If you performed a fresh install of a RHEL 9 host, your hypervisor uses modular libvirt daemons by default. However, if you upgraded your host from RHEL 8 to RHEL 9, your hypervisor uses the monolithic libvirtd daemon, which is the default in RHEL 8.

If that is the case, Red Hat recommends enabling the modular libvirt daemons instead, because they provide better options for fine-tuning libvirt resource management. In addition, libvirtd will become unsupported in a future major release of RHEL.

Prerequisites

- Your hypervisor is using the monolithic libvirtd service. To learn whether this is the case:
  
  ```
  # systemctl is-active libvirtd.service
  active
  
  If this command displays active, you are using libvirtd.
  ```

- Your virtual machines are shut down.

Procedure

1. Stop libvirtd and its sockets.
   
   ```
   # systemctl stop libvirtd.service
   # systemctl stop libvirtd{-ro,-admin,-tcp,-tls}.socket
   ```

2. Disable libvirtd to prevent it from starting on boot.
   
   ```
   $ systemctl disable libvirtd.service
   $ systemctl disable libvirtd{-ro,-admin,-tcp,-tls}.socket
   ```

3. Enable the modular libvirt daemons.
   
   ```
   # for drv in qemu interface network nodetool nwfilter secret storage do systemctl unmask virt${drv}d.service systemctl unmask virt${drv}d{-ro,-admin}.socket systemctl enable virt${drv}d.service systemctl enable virt${drv}d{-ro,-admin}.socket done
   ```

4. Start the sockets for the modular daemons.
   
   ```
   # for drv in qemu network nodetool nwfilter secret storage do systemctl start virt${drv}d{-ro,-admin}.socket done
   ```

5. Optional: If you require connecting to your host from remote hosts, enable and start the virtualization proxy daemon.
   
   ```
   # systemctl unmask virtproxyd.service
   # systemctl unmask virtproxyd{-ro,-admin,-tls}.socket
   # systemctl enable virtproxyd.service
   # systemctl enable virtproxyd{-ro,-admin,-tls}.socket
   # systemctl start virtproxyd{-ro,-admin,tls}.socket
   ```

Verification
1. Activate the enabled virtualization daemons.

   ```shell
   # virsh uri
   qemu:///system
   ```

2. Ensure your host is using the `virtqemud` modular daemon.

   ```shell
   # systemctl is-active virtqemud.service
   active
   ```

   If this command displays `active`, you have successfully enabled modular `libvirt` daemons.

### 17.4. CONFIGURING VIRTUAL MACHINE MEMORY

To improve the performance of a virtual machine (VM), you can assign additional host RAM to the VM. Similarly, you can decrease the amount of memory allocated to a VM so the host memory can be allocated to other VMs or tasks.

To perform these actions, you can use the web console or the command-line interface.

#### 17.4.1. Adding and removing virtual machine memory using the web console

To improve the performance of a virtual machine (VM) or to free up the host resources it is using, you can use the web console to adjust amount of memory allocated to the VM.

**Prerequisites**

- The guest OS is running the memory balloon drivers. To verify this is the case:
  1. Ensure the VM’s configuration includes the `memballoon` device:

     ```shell
     # virsh dumpxml testguest | grep membaloone
     <memballoon model='virtio'>
     </memballoon>
     ```

     If this commands displays any output and the model is not set to `none`, the `memballoon` device is present.

  2. Ensure the balloon drivers are running in the guest OS.

     - In Windows guests, the drivers are installed as a part of the `virtio-win` driver package. For instructions, see [Installing KVM paravirtualized drivers for Windows virtual machines](#).

     - In Linux guests, the drivers are generally included by default and activate when the `memballoon` device is present.

- The web console VM plug-in is installed on your system.

**Procedure**

1. **Optional**: Obtain the information about the maximum memory and currently used memory for a VM. This will serve as a baseline for your changes, and also for verification.
2. In the **Virtual Machines** interface, click the VM whose information you want to see.
   A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM’s graphical interface.

3. Click **edit** next to the **Memory** line in the Overview pane.
   The **Memory Adjustment** dialog appears.

4. Configure the virtual CPUs for the selected VM.
   - **Maximum allocation** - Sets the maximum amount of host memory that the VM can use for its processes. You can specify the maximum memory when creating the VM or increase it later. You can specify memory as multiples of MiB or GiB.
     Adjusting maximum memory allocation is only possible on a shut-off VM.
   
   - **Current allocation** - Sets the actual amount of memory allocated to the VM. This value can be less than the Maximum allocation but cannot exceed it. You can adjust the value to regulate the memory available to the VM for its processes. You can specify memory as multiples of MiB or GiB.
     If you do not specify this value, the default allocation is the **Maximum allocation** value.

5. Click **Save**.
   The memory allocation of the VM is adjusted.

Additional resources

- Adding and removing virtual machine memory using the command-line interface
- Optimizing virtual machine CPU performance

### 17.4.2. Adding and removing virtual machine memory using the command-line interface

To improve the performance of a virtual machine (VM) or to free up the host resources it is using, you can use the CLI to adjust amount of memory allocated to the VM.

**Prerequisites**

- The guest OS is running the memory balloon drivers. To verify this is the case:
1. Ensure the VM’s configuration includes the `memballoon` device:

```
# virsh dumpxml testguest | grep membaloos
<memballoon model='virtio'>
</memballoon>
```

If this command displays any output and the model is not set to `none`, the `memballoon` device is present.

2. Ensure the balloon drivers are running in the guest OS.

- In Windows guests, the drivers are installed as a part of the `virtio-win` driver package. For instructions, see [Installing KVM paravirtualized drivers for Windows virtual machines](#).
- In Linux guests, the drivers are generally included by default and activate when the `memballoon` device is present.

### Procedure

1. **Optional:** Obtain the information about the maximum memory and currently used memory for a VM. This will serve as a baseline for your changes, and also for verification.

```
# virsh dominfo testguest
Max memory:     2097152 KiB
Used memory:    2097152 KiB
```

2. Adjust the maximum memory allocated to a VM. Increasing this value improves the performance potential of the VM, and reducing the value lowers the performance footprint the VM has on your host. Note that this change can only be performed on a shut-off VM, so adjusting a running VM requires a reboot to take effect.

For example, to change the maximum memory that the `testguest` VM can use to 4096 MiB:

```
# virt-xml testguest --edit --memory memory=4096,currentMemory=4096
Domain 'testguest' defined successfully.
Changes will take effect after the domain is fully powered off.
```

To increase the maximum memory of a running VM, you can attach a memory device to the VM. This is also referred to as `memory hot plug`. For details, see [Attaching memory devices to virtual machines](#).

**WARNING**

Removing memory devices from a running VM (also referred as a memory hot unplug) is not supported, and highly discouraged by Red Hat.

3. **Optional:** You can also adjust the memory currently used by the VM, up to the maximum allocation. This regulates the memory load that the VM has on the host until the next reboot, without changing the maximum VM allocation.
# virsh setmem testguest --current 2048

Verification

1. Confirm that the memory used by the VM has been updated:

   # virsh dominfo testguest
   Max memory: 4194304 KiB
   Used memory: 2097152 KiB

2. **Optional**: If you adjusted the current VM memory, you can obtain the memory balloon statistics of the VM to evaluate how effectively it regulates its memory use.

   # virsh domstats --balloon testguest
   Domain: 'testguest'
   balloon.current=365624
   balloon.maximum=4194304
   balloon.swap_in=0
   balloon.swap_out=0
   balloon.major_fault=306
   balloon.minor_fault=156117
   balloon.unused=3834448
   balloon.available=4035008
   balloon.usable=3746340
   balloon.last-update=1587971682
   balloon.disk_caches=75444
   balloon.hugetlb_pgalloc=0
   balloon.hugetlb_pgfail=0
   balloon.rss=1005456

Additional resources

- Adding and removing virtual machine memory using the web console
- Optimizing virtual machine CPU performance

17.4.3. Additional resources

- Attaching devices to virtual machines.

17.5. OPTIMIZING VIRTUAL MACHINE I/O PERFORMANCE

The input and output (I/O) capabilities of a virtual machine (VM) can significantly limit the VM’s overall efficiency. To address this, you can optimize a VM’s I/O by configuring block I/O parameters.

17.5.1. Tuning block I/O in virtual machines

When multiple block devices are being used by one or more VMs, it might be important to adjust the I/O priority of specific virtual devices by modifying their I/O weights.

Increasing the I/O weight of a device increases its priority for I/O bandwidth, and therefore provides it with more host resources. Similarly, reducing a device’s weight makes it consume less host resources.
NOTE

Each device’s weight value must be within the 100 to 1000 range. Alternatively, the value can be 0, which removes that device from per-device listings.

Procedure

To display and set a VM’s block I/O parameters:

1. Display the current <blkio> parameters for a VM:
   
   ```
   # virsh dumpxml VM-name
   ```

   ```xml
   <domain>
   [...]
   <blkiotune>
   <weight>800</weight>
   <device>
   <path>/dev/sda</path>
   <weight>1000</weight>
   </device>
   <device>
   <path>/dev/sdb</path>
   <weight>500</weight>
   </device>
   </blkiotune>
   [...]  
   </domain>
   ```

2. Edit the I/O weight of a specified device:

   ```
   # virsh blkiotune VM-name --device-weights device, I/O-weight
   ```
   
   For example, the following changes the weight of the /dev/sda device in the liftrul VM to 500.

   ```
   # virsh blkiotune liftrul --device-weights /dev/sda, 500
   ```

17.5.2. Disk I/O throttling in virtual machines

When several VMs are running simultaneously, they can interfere with system performance by using excessive disk I/O. Disk I/O throttling in KVM virtualization provides the ability to set a limit on disk I/O requests sent from the VMs to the host machine. This can prevent a VM from over-utilizing shared resources and impacting the performance of other VMs.

To enable disk I/O throttling, set a limit on disk I/O requests sent from each block device attached to VMs to the host machine.

Procedure

1. Use the `virsh domblklist` command to list the names of all the disk devices on a specified VM.

   ```
   # virsh domblklist VM-name
   ```
   
   ```
   Target     Source
   --------------------------
   ```
vda /var/lib/libvirt/images/rollin-coal.qcow2  
sda -  
sdb /home/horridly-demanding-processes.iso

2. Find the host block device where the virtual disk that you want to throttle is mounted.  
For example, if you want to throttle the sdb virtual disk from the previous step, the following output shows that the disk is mounted on the /dev/nvme0n1p3 partition.

```
$ lsblk
NAME                                MAJ:MIN  RM  SIZE RO TYPE  MOUNTPOINT
zram0                                252:0   0   4G  0 disk  [SWAP]  
nvme0n1                               259:0   0 238.5G 0 disk  
  └─nvme0n1p1                           259:1   0 600M  0 part  /boot/efi  
  └─nvme0n1p2                           259:2   0  1G  0 part  /boot  
  └─nvme0n1p3                           259:3   0 236.9G 0 part  
    └─luks-a1123911-6f37-463c-b4eb-fxzy1ac12fe 253:0   0 236.9G 0 crypt  /home
```

3. Set I/O limits for the block device using the `virsh blkiotune` command.

```
# virsh blkiotune VM-name --parameter device,limit
```

The following example throttles the sdb disk on the rollin-coal VM to 1000 read and write I/O operations per second and to 50 MB per second read and write throughput.

```
# virsh blkiotune rollin-coal --device-read-iops-sec /dev/nvme0n1p3,1000 --device-write-iops-sec /dev/nvme0n1p3,1000 --device-write-bytes-sec /dev/nvme0n1p3,52428800 --device-read-bytes-sec /dev/nvme0n1p3,52428800
```

Additional information

- Disk I/O throttling can be useful in various situations, for example when VMs belonging to different customers are running on the same host, or when quality of service guarantees are given for different VMs. Disk I/O throttling can also be used to simulate slower disks.

- I/O throttling can be applied independently to each block device attached to a VM and supports limits on throughput and I/O operations.

- Red Hat does not support using the `virsh blkdeviotune` command to configure I/O throttling in VMs. For more information on unsupported features when using RHEL 9 as a VM host, see [Unsupported features in RHEL 9 virtualization](#).

17.5.3. Enabling multi-queue virtio-scsi

When using virtio-scsi storage devices in your virtual machines (VMs), the multi-queue virtio-scsi feature provides improved storage performance and scalability. It enables each virtual CPU (vCPU) to have a separate queue and interrupt to use without affecting other vCPUs.

Procedure

- To enable multi-queue virtio-scsi support for a specific VM, add the following to the VM’s XML configuration, where N is the total number of vCPU queues:
Much like physical CPUs in host machines, vCPUs are critical to virtual machine (VM) performance. As a result, optimizing vCPUs can have a significant impact on the resource efficiency of your VMs. To optimize your vCPU:

1. Adjust how many host CPUs are assigned to the VM. You can do this using the CLI or the web console.

2. Ensure that the vCPU model is aligned with the CPU model of the host. For example, to set the testguest1 VM to use the CPU model of the host:

```
# virt-xml testguest1 --edit --cpu host-model
```

3. Manage kernel same-page merging (KSM).

4. If your host machine uses Non-Uniform Memory Access (NUMA), you can also configure NUMA for its VMs. This maps the host’s CPU and memory processes onto the CPU and memory processes of the VM as closely as possible. In effect, NUMA tuning provides the vCPU with a more streamlined access to the system memory allocated to the VM, which can improve the vCPU processing effectiveness. For details, see Configuring NUMA in a virtual machine and Sample vCPU performance tuning scenario.

17.6.1. Adding and removing virtual CPUs using the command-line interface

To increase or optimize the CPU performance of a virtual machine (VM), you can add or remove virtual CPUs (vCPUs) assigned to the VM.

When performed on a running VM, this is also referred to as vCPU hot plugging and hot unplugging. However, note that vCPU hot unplug is not supported in RHEL 9, and Red Hat highly discourages its use.

**Prerequisites**

- **Optional:** View the current state of the vCPUs in the targeted VM. For example, to display the number of vCPUs on the testguest VM:

```
# virsh vcpucount testguest
maximum config 4
maximum live 2
current config 2
current live 1
```

This output indicates that testguest is currently using 1 vCPU, and 1 more vCPU can be hot plugged to it to increase the VM’s performance. However, after reboot, the number of vCPUs testguest uses will change to 2, and it will be possible to hot plug 2 more vCPUs.

**Procedure**
1. Adjust the maximum number of vCPUs that can be attached to a VM, which takes effect on the VM's next boot. For example, to increase the maximum vCPU count for the testguest VM to 8:

```
# virsh setvcpus testguest 8 --maximum --config
```

Note that the maximum may be limited by the CPU topology, host hardware, the hypervisor, and other factors.

2. Adjust the current number of vCPUs attached to a VM, up to the maximum configured in the previous step. For example:

- To increase the number of vCPUs attached to the running testguest VM to 4:

  ```
  # virsh setvcpus testguest 4 --live
  ```

  This increases the VM's performance and host load footprint of testguest until the VM's next boot.

- To permanently decrease the number of vCPUs attached to the testguest VM to 1:

  ```
  # virsh setvcpus testguest 1 --config
  ```

  This decreases the VM's performance and host load footprint of testguest after the VM's next boot. However, if needed, additional vCPUs can be hot plugged to the VM to temporarily increase its performance.

**Verification**

- Confirm that the current state of vCPU for the VM reflects your changes.

  ```
  # virsh vcpucount testguest
  maximum      config         8
  maximum      live           4
  current      config         1
  current      live           4
  ```

**Additional resources**

- Managing virtual CPUs using the web console

**17.6.2. Managing virtual CPUs using the web console**

Using the RHEL 9 web console, you can review and configure virtual CPUs used by virtual machines (VMs) to which the web console is connected.

**Prerequisites**

- The web console VM plug-in is installed on your system.

**Procedure**

1. In the Virtual Machines interface, click the VM whose information you want to see.
A new page opens with an Overview section with basic information about the selected VM and a Console section to access the VM’s graphical interface.

2. Click **edit** next to the number of vCPUs in the Overview pane. The vCPU details dialog appears.

   ![Grid_v2 vCPU details](image)

1. Configure the virtual CPUs for the selected VM.

   - **vCPU Count** - The number of vCPUs currently in use.

   ![NOTE](image)

     The vCPU count cannot be greater than the vCPU Maximum.

   - **vCPU Maximum** - The maximum number of virtual CPUs that can be configured for the VM. If this value is higher than the **vCPU Count**, additional vCPUs can be attached to the VM.

   - **Sockets** - The number of sockets to expose to the VM.

   - **Cores per socket** - The number of cores for each socket to expose to the VM.

   - **Threads per core** - The number of threads for each core to expose to the VM.

     Note that the **Sockets**, **Cores per socket**, and **Threads per core** options adjust the CPU topology of the VM. This may be beneficial for vCPU performance and may impact the functionality of certain software in the guest OS. If a different setting is not required by your deployment, keep the default values.

2. Click **Apply**.

   The virtual CPUs for the VM are configured.

   ![NOTE](image)

   Changes to virtual CPU settings only take effect after the VM is restarted.

**Additional resources**

- Adding and removing virtual CPUs using the command-line interface

**17.6.3. Configuring NUMA in a virtual machine**
The following methods can be used to configure Non-Uniform Memory Access (NUMA) settings of a virtual machine (VM) on a RHEL 9 host.

**Prerequisites**

- The host is a NUMA-compatible machine. To detect whether this is the case, use the `virsh nodeinfo` command and see the NUMA cell(s) line:

  ```
  # virsh nodeinfo
  CPU model:           x86_64
  CPU(s):              48
  CPU frequency:       1200 MHz
  CPU socket(s):       1
  Core(s) per socket:  12
  Thread(s) per core:  2
  NUMA cell(s):        2
  Memory size:         67012964 KiB
  ```

  If the value of the line is 2 or greater, the host is NUMA-compatible.

**Procedure**

For ease of use, you can set up a VM's NUMA configuration using automated utilities and services. However, manual NUMA setup is more likely to yield a significant performance improvement.

**Automatic methods**

- Set the VM’s NUMA policy to **Preferred**. For example, to do so for the `testguest5` VM:

  ```
  # virt-xml testguest5 --edit --vcpus placement=auto
  # virt-xml testguest5 --edit --numatune mode=preferred
  ```

- Enable automatic NUMA balancing on the host:

  ```
  # echo 1 > /proc/sys/kernel/numa_balancing
  ```

- Use the `numad` command to automatically align the VM CPU with memory resources.

  ```
  # numad
  ```

**Manual methods**

1. Pin specific vCPU threads to a specific host CPU or range of CPUs. This is also possible on non-NUMA hosts and VMs, and is recommended as a safe method of vCPU performance improvement.

   For example, the following commands pin vCPU threads 0 to 5 of the `testguest6` VM to host CPUs 1, 3, 5, 7, 9, and 11, respectively:

   ```
   # virsh vcpupin testguest6 0 1
   # virsh vcpupin testguest6 1 3
   # virsh vcpupin testguest6 2 5
   # virsh vcpupin testguest6 3 7
   # virsh vcpupin testguest6 4 9
   # virsh vcpupin testguest6 5 11
   ```
Afterwards, you can verify whether this was successful:

```bash
# virsh vcpupin testguest6
VCPU   CPU Affinity
------------
0      1
1      3
2      5
3      7
4      9
5      11
```

2. After pinning vCPU threads, you can also pin QEMU process threads associated with a specified VM to a specific host CPU or range of CPUs. For example, the following commands pin the QEMU process thread of `testguest6` to CPUs 13 and 15, and verify this was successful:

```bash
# virsh emulatorpin testguest6 13,15
# virsh emulatorpin testguest6
emulator: CPU Affinity
-----------
*: 13,15
```

3. Finally, you can also specify which host NUMA nodes will be assigned specifically to a certain VM. This can improve the host memory usage by the VM’s vCPU. For example, the following commands set `testguest6` to use host NUMA nodes 3 to 5, and verify this was successful:

```bash
# virsh numatune testguest6 --nodeset 3-5
# virsh numatune testguest6
```

**NOTE**

For best performance results, it is recommended to use all of the manual tuning methods listed above.

**Known issues**

- NUMA tuning currently cannot be performed on IBM Z hosts.

**Additional resources**

- Sample vCPU performance tuning scenario
- View the current NUMA configuration of your system using the `numastat` utility

**17.6.4. Sample vCPU performance tuning scenario**

To obtain the best vCPU performance possible, Red Hat recommends using manual `vcpupin`, `emulatorpin`, and `numatune` settings together, for example like in the following scenario.

**Starting scenario**

- Your host has the following hardware specifics:
  - 2 NUMA nodes
• 3 CPU cores on each node
• 2 threads on each core

The output of `virsh nodeinfo` of such a machine would look similar to:

```bash
# virsh nodeinfo
CPU model:           x86_64
CPU(s):              12
CPU frequency:       3661 MHz
CPU socket(s):       2
Core(s) per socket:  3
Thread(s) per core:  2
NUMA cell(s):        2
Memory size:         31248692 KiB
```

• You intend to modify an existing VM to have 8 vCPUs, which means that it will not fit in a single NUMA node. Therefore, you should distribute 4 vCPUs on each NUMA node and make the vCPU topology resemble the host topology as closely as possible. This means that vCPUs that run as sibling threads of a given physical CPU should be pinned to host threads on the same core. For details, see the Solution below:

**Solution**

1. Obtain the information on the host topology:

```bash
# virsh capabilities
```

The output should include a section that looks similar to the following:

```xml
<topology>
  <cells num="2">
    <cell id="0">
      <memory unit="KiB">15624346</memory>
      <pages unit="KiB" size="4">3906086</pages>
      <pages unit="KiB" size="2048">0</pages>
      <pages unit="KiB" size="1048576">0</pages>
      <distances>
        <sibling id="0" value="10" />  
        <sibling id="1" value="21" />
      </distances>
      <cpus num="6">
        <cpu id="0" socket_id="0" core_id="0" siblings="0,3" />  
        <cpu id="1" socket_id="0" core_id="1" siblings="1,4" />  
        <cpu id="2" socket_id="0" core_id="2" siblings="2,5" />  
        <cpu id="3" socket_id="0" core_id="0" siblings="0,3" />  
        <cpu id="4" socket_id="0" core_id="1" siblings="1,4" />  
        <cpu id="5" socket_id="0" core_id="2" siblings="2,5" />
      </cpus>
    </cell>
    <cell id="1">
      <memory unit="KiB">15624346</memory>
      <pages unit="KiB" size="4">3906086</pages>
      <pages unit="KiB" size="2048">0</pages>
    </cell>
  </cells>
</topology>
```
2. **Optional:** Test the performance of the VM using the applicable tools and utilities.

3. Set up and mount 1 GiB huge pages on the host:
   a. Add the following line to the host’s kernel command line:
      
      ```
      default_hugepagesz=1G hugepagesz=1G
      ```
   b. Create the `/etc/systemd/system/hugetlb-gigantic-pages.service` file with the following content:
      
      ```
      [Unit]
      Description=HugeTLB Gigantic Pages Reservation
      DefaultDependencies=no
      Before=dev-hugepages.mount
      ConditionPathExists=/sys/devices/system/node
      ConditionKernelCommandLine=hugepagesz=1G

      [Service]
      Type=oneshot
      RemainAfterExit=yes
      ExecStart=/etc/systemd/hugetlb-reserve-pages.sh

      [Install]
      WantedBy=sysinit.target
      ```
   c. Create the `/etc/systemd/hugetlb-reserve-pages.sh` file with the following content:
      
      ```
      #!/bin/sh

      nodes_path=/sys/devices/system/node/
      if [ ! -d $nodes_path ]; then
        echo "ERROR: $nodes_path does not exist"
        exit 1
      fi

      reserve_pages()
      ```
echo $1 > $nodes_path/$2/hugepages/hugepages-1048576kB/nr_hugepages
}
reserve_pages 4 node1
reserve_pages 4 node2

This reserves four 1GiB huge pages from node1 and four 1GiB huge pages from node2.

d. Make the script created in the previous step executable:

```
# chmod +x /etc/systemd/hugetlb-reserve-pages.sh
```

e. Enable huge page reservation on boot:

```
# systemctl enable hugetlb-gigantic-pages
```

4. Use the `virsh edit` command to edit the XML configuration of the VM you wish to optimize, in this example super-VM:

```
# virsh edit super-VM
```

5. Adjust the XML configuration of the VM in the following way:

a. Set the VM to use 8 static vCPUs. Use the `<vcpu/>` element to do this.

b. Pin each of the vCPU threads to the corresponding host CPU threads that it mirrors in the topology. To do so, use the `<vcpupin/>` elements in the `<cputune>` section.

Note that, as shown by the `virsh capabilities` utility above, host CPU threads are not ordered sequentially in their respective cores. In addition, the vCPU threads should be pinned to the highest available set of host cores on the same NUMA node. For a table illustration, see the Sample topology section below.

The XML configuration for steps a. and b. can look similar to:

```
<cpuset>
    <vcpupin vcpu='0' cpuset='1'/>
    <vcpupin vcpu='1' cpuset='4'/>
    <vcpupin vcpu='2' cpuset='2'/>
    <vcpupin vcpu='3' cpuset='5'/>
    <vcpupin vcpu='4' cpuset='7'/>
    <vcpupin vcpu='5' cpuset='10'/>
    <vcpupin vcpu='6' cpuset='8'/>
    <vcpupin vcpu='7' cpuset='11'/>
    <emulatorpin cpuset='6,9'/>
</cpuset>
```

c. Set the VM to use 1 GiB huge pages:

```
<memoryBacking>
    <hugepages>
        <page size='1' unit='GiB'/>
    </hugepages>
</memoryBacking>
```
d. Configure the VM's NUMA nodes to use memory from the corresponding NUMA nodes on the host. To do so, use the `<memnode/>` elements in the `<numatune/>` section:

```
<numatune>
  <memory mode="preferred" nodeset="1"/>
  <memnode cellid="0" mode="strict" nodeset="0"/>
  <memnode cellid="1" mode="strict" nodeset="1"/>
</numatune>
```

e. Ensure the CPU mode is set to **host-passthrough**, and that the CPU uses cache in **passthrough** mode:

```
<cpu mode="host-passthrough">
  <topology sockets="2" cores="2" threads="2"/>
  <cache mode="passthrough"/>
</cpu>
```

**Verification**

1. Confirm that the resulting XML configuration of the VM includes a section similar to the following:

```
[...]
<memoryBacking>
  <hugepages>
    <page size="1" unit="GiB"/>
  </hugepages>
</memoryBacking>
<vcpu placement='static'>8</vcpu>
<brutune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='4'/>
  <vcpupin vcpu='2' cpuset='2'/>
  <vcpupin vcpu='3' cpuset='5'/>
  <vcpupin vcpu='4' cpuset='7'/>
  <vcpupin vcpu='5' cpuset='10'/>
  <vcpupin vcpu='6' cpuset='8'/>
  <vcpupin vcpu='7' cpuset='11'/>
  <emulatorpin cpuset='6,9'/>
</brutune>
<brutune>
  <memory mode="preferred" nodeset="1"/>
  <memnode cellid="0" mode="strict" nodeset="0"/>
  <memnode cellid="1" mode="strict" nodeset="1"/>
</numatune>
<cpu mode="host-passthrough">
  <topology sockets="2" cores="2" threads="2"/>
  <cache mode="passthrough"/>
  <numa>
    <cell id="0" cpus="0-3" memory="2" unit="GiB">
      <distances>
        <sibling id="0" value="10"/>
        <sibling id="1" value="21"/>
      </distances>
    </cell>
    <cell id="1" cpus="4-7" memory="2" unit="GiB">
```

2. **Optional:** Test the performance of the VM using the applicable tools and utilities to evaluate the impact of the VM’s optimization.

### Sample topology

- The following tables illustrate the connections between the vCPUs and the host CPUs they should be pinned to:

#### Table 17.1. Host topology

<table>
<thead>
<tr>
<th>CPU threads</th>
<th>0</th>
<th>3</th>
<th>1</th>
<th>4</th>
<th>2</th>
<th>5</th>
<th>6</th>
<th>9</th>
<th>7</th>
<th>10</th>
<th>8</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sockets</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMA nodes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Table 17.2. VM topology

<table>
<thead>
<tr>
<th>vCPU threads</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sockets</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NUMA nodes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

#### Table 17.3. Combined host and VM topology

<table>
<thead>
<tr>
<th>vCPU threads</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host CPU threads</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Cores</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sockets</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NUMA nodes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
In this scenario, there are 2 NUMA nodes and 8 vCPUs. Therefore, 4 vCPU threads should be pinned to each node.

In addition, Red Hat recommends leaving at least a single CPU thread available on each node for host system operations.

Because in this example, each NUMA node houses 3 cores, each with 2 host CPU threads, the set for node 0 translates as follows:

```xml
<vcpupin vcpu='0' cpuset='1'/>
<vcpupin vcpu='1' cpuset='4'/>
<vcpupin vcpu='2' cpuset='2'/>
<vcpupin vcpu='3' cpuset='5'/>
```

17.6.5. Managing kernel same-page merging

Kernel Same-Page Merging (KSM) improves memory density by sharing identical memory pages between virtual machines (VMs). However, enabling KSM increases CPU utilization, and might adversely affect overall performance depending on the workload.

Depending on your requirements, you can either enable or disable KSM for a single session or persistently.

**NOTE**

In RHEL 9 and later, KSM is disabled by default.

**Prerequisites**

- Root access to your host system.

**Procedure**

- Disable KSM:
  - To deactivate KSM for a single session, use the `systemctl` utility to stop `ksm` and `ksmtuned` services.
    ```bash
    # systemctl stop ksm
    # systemctl stop ksmtuned
    ```
  - To deactivate KSM persistently, use the `systemctl` utility to disable `ksm` and `ksmtuned` services.
    ```bash
    # systemctl disable ksm
    Removed /etc/systemd/system/multi-user.target.wants/ksm.service.
    # systemctl disable ksmtuned
    Removed /etc/systemd/system/multi-user.target.wants/ksmtuned.service.
    ```
NOTE

Memory pages shared between VMs before deactivating KSM will remain shared. To stop sharing, delete all the **PageKSM** pages in the system using the following command:

```
# echo 2 > /sys/kernel/mm/ksm/run
```

After anonymous pages replace the KSM pages, the **khugepaged** kernel service will rebuild transparent hugepages on the VM’s physical memory.

- Enable KSM:

**WARNING**

Enabling KSM increases CPU utilization and affects overall CPU performance.

1. Install the **ksmtuned** service:

   ```
   # yum install ksmtuned
   ```

2. Start the service:

   - To enable KSM for a single session, use the **systemctl** utility to start the **ksm** and **ksmtuned** services.

     ```
     # systemctl start ksm
     # systemctl start ksmtuned
     ```

   - To enable KSM persistently, use the **systemctl** utility to enable the **ksm** and **ksmtuned** services.

     ```
     # systemctl enable ksm
     Created symlink /etc/systemd/system/multi-user.target.wants/ksm.service → /usr/lib/systemd/system/ksm.service

     # systemctl enable ksmtuned
     Created symlink /etc/systemd/system/multi-user.target.wants/ksmtuned.service → /usr/lib/systemd/system/ksmtuned.service
     ```

17.7. OPTIMIZING VIRTUAL MACHINE NETWORK PERFORMANCE

Due to the virtual nature of a VM’s network interface card (NIC), the VM loses a portion of its allocated host network bandwidth, which can reduce the overall workload efficiency of the VM. The following tips can minimize the negative impact of virtualization on the virtual NIC (vNIC) throughput.

**Procedure**

Use any of the following methods and observe if it has a beneficial effect on your VM network performance:
Enable the vhost_net module

On the host, ensure the vhost_net kernel feature is enabled:

```bash
# lsmod | grep vhost
vhost_net            32768  1
vhost               53248  1 vhost_net
tap                24576  1 vhost_net
tun                57344  6 vhost_net
```

If the output of this command is blank, enable the vhost_net kernel module:

```bash
# modprobe vhost_net
```

Set up multi-queue virtio-net

To set up the multi-queue virtio-net feature for a VM, use the virsh edit command to edit the XML configuration of the VM. In the XML, add the following to the <devices> section, and replace N with the number of vCPUs in the VM, up to 16:

```
<interface type='network'>
    <source network='default'/>
    <model type='virtio'/>
    <driver name='vhost' queues='N'/>
</interface>
```

If the VM is running, restart it for the changes to take effect.

Batching network packets

In Linux VM configurations with a long transmission path, batching packets before submitting them to the kernel may improve cache utilization. To set up packet batching, use the following command on the host, and replace tap0 with the name of the network interface that the VMs use:

```bash
# ethtool -C tap0 rx-frames 64
```

SR-IOV

If your host NIC supports SR-IOV, use SR-IOV device assignment for your vNICs. For more information, see Managing SR-IOV devices.

Additional resources

- Understanding virtual networking

17.8. VIRTUAL MACHINE PERFORMANCE MONITORING TOOLS

To identify what consumes the most VM resources and which aspect of VM performance needs optimization, performance diagnostic tools, both general and VM-specific, can be used.

Default OS performance monitoring tools

For standard performance evaluation, you can use the utilities provided by default by your host and guest operating systems:
On your RHEL 9 host, as root, use the **top** utility or the **system monitor** application, and look for **qemu** and **virt** in the output. This shows how much host system resources your VMs are consuming.

- If the monitoring tool displays that any of the **qemu** or **virt** processes consume a large portion of the host CPU or memory capacity, use the **perf** utility to investigate. For details, see below.

- In addition, if a **vhost_net** thread process, named for example **vhost_net-1234**, is displayed as consuming an excessive amount of host CPU capacity, consider using virtual network optimization features, such as **multi-queue virtio-net**.

On the guest operating system, use performance utilities and applications available on the system to evaluate which processes consume the most system resources.

- On Linux systems, you can use the **top** utility.

- On Windows systems, you can use the **Task Manager** application.

**perf kvm**

You can use the **perf** utility to collect and analyze virtualization-specific statistics about the performance of your RHEL 9 host. To do so:

1. On the host, install the **perf** package:

   ```bash
   # dnf install perf
   ```

2. Use one of the **perf kvm stat** commands to display perf statistics for your virtualization host:

   - For real-time monitoring of your hypervisor, use the **perf kvm stat live** command.

   - To log the perf data of your hypervisor over a period of time, activate the logging using the **perf kvm stat record** command. After the command is canceled or interrupted, the data is saved in the **perf.data.guest** file, which can be analyzed using the **perf kvm stat report** command.

3. Analyze the **perf** output for types of **VM-EXIT** events and their distribution. For example, the **PAUSE_INSTRUCTION** events should be infrequent, but in the following output, the high occurrence of this event suggests that the host CPUs are not handling the running vCPUs well. In such a scenario, consider shutting down some of your active VMs, removing vCPUs from these VMs, or tuning the performance of the vCPUs.

```
# perf kvm stat report

Analyze events for all VMs, all VCPUs:

<table>
<thead>
<tr>
<th>VM-EXIT</th>
<th>Samples</th>
<th>Samples%</th>
<th>Time%</th>
<th>Min Time</th>
<th>Max Time</th>
<th>Avg time</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTERNAL_INTERRUPT</td>
<td>365634</td>
<td>31.59%</td>
<td>18.04%</td>
<td>0.42us</td>
<td>58780.59us</td>
<td>204.08us (+- 0.99% )</td>
</tr>
<tr>
<td>MSR_WRITE</td>
<td>293428</td>
<td>25.35%</td>
<td>0.13%</td>
<td>0.59us</td>
<td>17873.02us</td>
<td>1.80us (+- 4.63% )</td>
</tr>
<tr>
<td>PREEMPTION_TIMER</td>
<td>276162</td>
<td>23.86%</td>
<td>0.23%</td>
<td>0.51us</td>
<td>21396.03us</td>
<td>3.38us (+- 5.19% )</td>
</tr>
<tr>
<td>PAUSE_INSTRUCTION</td>
<td>189375</td>
<td>16.36%</td>
<td>11.75%</td>
<td>0.72us</td>
<td>29655.25us</td>
<td>256.77us</td>
</tr>
</tbody>
</table>
```
Other event types that can signal problems in the output of `perf kvm stat` include:

- **INSN_EMULATION** - suggests suboptimal VM I/O configuration.

For more information on using `perf` to monitor virtualization performance, see the `perf-kvm` man page.

**numastat**

To see the current NUMA configuration of your system, you can use the `numastat` utility, which is provided by installing the `numactl` package.

The following shows a host with 4 running VMs, each obtaining memory from multiple NUMA nodes. This is not optimal for vCPU performance, and warrants adjusting:

```
# numastat -c qemu-kvm
```

Per-node process memory usage (in MBs)

```
<table>
<thead>
<tr>
<th>PID</th>
<th>Node 0</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Node 4</th>
<th>Node 5</th>
<th>Node 6</th>
<th>Node 7</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>51722 (qemu-kvm)</td>
<td>68</td>
<td>16</td>
<td>357</td>
<td>6936</td>
<td>2</td>
<td>3</td>
<td>147</td>
<td>598</td>
<td>8128</td>
</tr>
<tr>
<td>51747 (qemu-kvm)</td>
<td>245</td>
<td>11</td>
<td>5</td>
<td>18</td>
<td>5172</td>
<td>2532</td>
<td>1</td>
<td>92</td>
<td>8076</td>
</tr>
<tr>
<td>53736 (qemu-kvm)</td>
<td>62</td>
<td>432</td>
<td>1661</td>
<td>506</td>
<td>4851</td>
<td>136</td>
<td>22</td>
<td>445</td>
<td>8116</td>
</tr>
<tr>
<td>53773 (qemu-kvm)</td>
<td>1393</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>6702</td>
<td>8114</td>
</tr>
<tr>
<td>Total</td>
<td>1769</td>
<td>463</td>
<td>2024</td>
<td>7462</td>
<td>10037</td>
<td>2672</td>
<td>169</td>
<td>7837</td>
<td>32434</td>
</tr>
</tbody>
</table>
```

In contrast, the following shows memory being provided to each VM by a single node, which is significantly more efficient.

```
# numastat -c qemu-kvm
```

Per-node process memory usage (in MBs)

```
<table>
<thead>
<tr>
<th>PID</th>
<th>Node 0</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Node 4</th>
<th>Node 5</th>
<th>Node 6</th>
<th>Node 7</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>51747 (qemu-kvm)</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>8072</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>8080</td>
</tr>
<tr>
<td>53736 (qemu-kvm)</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>8113</td>
<td>0</td>
<td>8120</td>
<td>8120</td>
</tr>
<tr>
<td>53773 (qemu-kvm)</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8110</td>
<td>8118</td>
</tr>
<tr>
<td>59065 (qemu-kvm)</td>
<td>0</td>
<td>0</td>
<td>8050</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8051</td>
</tr>
<tr>
<td>Total</td>
<td>0</td>
<td>0</td>
<td>8072</td>
<td>0</td>
<td>8072</td>
<td>0</td>
<td>8114</td>
<td>8110</td>
<td>32368</td>
</tr>
</tbody>
</table>
```

17.9. ADDITIONAL RESOURCES
- Optimizing Windows virtual machines
CHAPTER 18. SECURING VIRTUAL MACHINES

As an administrator of a RHEL 9 system with virtual machines (VMs), ensuring that your VMs are as secure as possible significantly lowers the risk of your guest and host OSs being infected by malicious software.

This document outlines the mechanics of securing VMs on a RHEL 9 host and provides a list of methods to increase the security of your VMs.

18.1. HOW SECURITY WORKS IN VIRTUAL MACHINES

When using virtual machines (VMs), multiple operating systems can be housed within a single host machine. These systems are connected with the host through the hypervisor, and usually also through a virtual network. As a consequence, each VM can be used as a vector for attacking the host with malicious software, and the host can be used as a vector for attacking any of the VMs.

Figure 18.1. A potential malware attack vector on a virtualization host

Because the hypervisor uses the host kernel to manage VMs, services running on the VM’s operating system are frequently used for injecting malicious code into the host system. However, you can protect your system against such security threats by using a number of security features on your host and your guest systems.

These features, such as SELinux or QEMU sandboxing, provide various measures that make it more difficult for malicious code to attack the hypervisor and transfer between your host and your VMs.
Many of the features that RHEL 9 provides for VM security are always active and do not have to be enabled or configured. For details, see Automatic features for virtual machine security.

In addition, you can adhere to a variety of best practices to minimize the vulnerability of your VMs and your hypervisor. For more information, see Best practices for securing virtual machines.

**18.2. BEST PRACTICES FOR SECURING VIRTUAL MACHINES**

Following the instructions below significantly decreases the risk of your virtual machines being infected with malicious code and used as attack vectors to infect your host system.

**On the guest side:**

- Secure the virtual machine as if it was a physical machine. The specific methods available to enhance security depend on the guest OS.
  
  If your VM is running RHEL 9, see Securing Red Hat Enterprise Linux 9 for detailed instructions on improving the security of your guest system.

**On the host side:**

- When managing VMs remotely, use cryptographic utilities such as SSH and network protocols such as SSL for connecting to the VMs.

- Ensure SELinux is in Enforcing mode:

  ```
  # getenforce
  Enforcing
  ```

  If SELinux is disabled or in Permissive mode, see the Using SELinux document for instructions on activating Enforcing mode.
NOTE

SELinux Enforcing mode also enables the sVirt RHEL 9 feature. This is a set of specialized SELinux booleans for virtualization, which can be manually adjusted for fine-grained VM security management.

- Use VMs with SecureBoot:
  SecureBoot is a feature that ensures that your VM is running a cryptographically signed OS. This prevents VMs whose OS has been altered by a malware attack from booting.

  SecureBoot can only be applied when installing a Linux VM that uses OVMF firmware. For instructions, see Creating a SecureBoot virtual machine.

- Do not use qemu-* commands, such as qemu-kvm.
  QEMU is an essential component of the virtualization architecture in RHEL 9, but it is difficult to manage manually, and improper QEMU configurations may cause security vulnerabilities. Therefore, using qemu-* commands is not supported by Red Hat. Instead, use libvirt utilities, such as virsh, virt-install, and virt-xml, as these orchestrate QEMU according to the best practices.

Additional resources

- SELinux booleans for virtualization in RHEL

18.3. CREATING A SECUREBOOT VIRTUAL MACHINE

You can create a Linux virtual machine (VM) that uses the SecureBoot feature, which ensures that your VM is running a cryptographically signed OS. This can be useful if the guest OS of a VM has been altered by malware. In such a scenario, SecureBoot prevents the VM from booting, which stops the potential spread of the malware to your host machine.

Prerequisites

- The VM is using the Q35 machine type.

- The edk2-OVMF packages is installed:
  
  # dnf install edk2-ovmf

- An operating system (OS) installation source is available locally or on a network. This can be one of the following formats:
  
  - An ISO image of an installation medium
  
  - A disk image of an existing VM installation
WARNING
Installing from a host CD-ROM or DVD-ROM device is not possible in RHEL 9. If you select a CD-ROM or DVD-ROM as the installation source when using any VM installation method available in RHEL 9, the installation will fail. For more information, see the Red Hat Knowledgebase.

- Optional: A Kickstart file can be provided for faster and easier configuration of the installation.

Procedure

1. Use the `virt-install` command to create a VM as detailed in Creating virtual machines using the command-line interface. For the `--boot` option, use the `uefi,nvram_template=/usr/share/OVMF/OVMF_VARS.secboot.fd` value. This uses the `OVMF_VARS.secboot.fd` and `OVMF_CODE.secboot.fd` files as templates for the VM’s non-volatile RAM (NVRAM) settings, which enables the SecureBoot feature.

   For example:

   ```
   # virt-install --name rhel8sb --memory 4096 --vcpus 4 --os-variant rhel9.0 --boot uefi,nvram_template=/usr/share/OVMF/OVMF_VARS.secboot.fd --disk boot_order=2,size=10 --disk boot_order=1,device=cdrom,bus=scsi,path=/images/RHEL-9.0-installation.iso
   ```

2. Follow the OS installation procedure according to the instructions on the screen.

Verification

1. After the guest OS is installed, access the VM’s command line by opening the terminal in the graphical guest console or connecting to the guest OS using SSH.

2. To confirm that SecureBoot has been enabled on the VM, use the `mokutil --sb-state` command:

   ```
   # mokutil --sb-state
   SecureBoot enabled
   ```

Additional resources

- Installing RHEL 9 on AMD64, Intel 64, and 64-bit ARM

18.4. LIMITING WHAT ACTIONS ARE AVAILABLE TO VIRTUAL MACHINE USERS

In some cases, actions that users of virtual machines (VMs) hosted on RHEL 9 can perform by default may pose a security risk. If that is the case, you can limit the actions available to VM users by configuring the `libvirt` daemons to use the `polkit` policy toolkit on the host machine.

Procedure
1. **Optional:** Ensure your system’s **polkit** control policies related to **libvirt** are set up according to your preferences.
      
      ```
      # ls /usr/share/polkit-1/actions | grep libvirt
      # ls /usr/share/polkit-1/rules.d | grep libvirt
      ```
   b. Open the files and review the rule settings.
      For information on reading the syntax of **polkit** control policies, use `man polkit`.
   c. Modify the **libvirt** control policies. To do so:
      i. Create a new `.conf` file in the `/etc/polkit-1/rules.d/` directory.
      ii. Add your custom policies to this file, and save it.
      For further information and examples of **libvirt** control policies, see the **libvirt** upstream documentation.

2. Configure your VMs to use access policies determined by **polkit**.
   a. Find all virtualization driver configuration files in the `/etc/libvirt/` directory.
      
      ```
      # ls /etc/libvirt/ | grep virt*d.conf
      ```
   b. In each of the files, uncomment the `access_drivers = [ "polkit" ]` line and save the file.

3. For each file that you modified in the previous step, restart the corresponding service.
   For example, if you have modified `/etc/libvirt/virtqemud.conf`, restart the **virtqemud** service.
      
      ```
      # systemctl try-restart virtqemud
      ```

**Verification**

- As a user whose VM actions you intended to limit, perform one of the restricted actions.
  For example, if unprivileged users are restricted from viewing VMs created in the system session:
      
      ```
      $ virsh -c qemu:///system list --all
      Id   Name           State
      ---------------------
      ```
  If this command does not list any VMs even though one or more VMs exist on your system, **polkit** successfully restricts the action for unprivileged users.

**Troubleshooting**

- Currently, configuring **libvirt** to use **polkit** makes it impossible to connect to VMs using the RHEL 9 web console, due to an incompatibility with the **libvirt-dbus** service.
  If you require fine-grained access control of VMs in the web console, Red Hat recommends creating a custom D-Bus policy. For instructions, see How to configure fine-grained control of Virtual Machines in Cockpit in the Red Hat Knowledgebase.
Additional resources

- The **man polkit** command
- The **libvirt** man pages on polkit access control policies

### 18.5. AUTOMATIC FEATURES FOR VIRTUAL MACHINE SECURITY

In addition to manual means of improving the security of your virtual machines listed in *Best practices for securing virtual machines*, a number of security features are provided by the **libvirt** software suite and are automatically enabled when using virtualization in RHEL 9. These include:

**System and session connections**

To access all the available utilities for virtual machine management in RHEL 9, you need to use the *system connection* of **libvirt** (*qemu:///system*). To do so, you must have root privileges on the system or be a part of the **libvirt** user group.

Non-root users that are not in the **libvirt** group can only access a *session connection* of **libvirt** (*qemu:///session*), which has to respect the access rights of the local user when accessing resources. For example, using the session connection, you cannot detect or access VMs created in the system connection or by other users. Also, available VM networking configuration options are significantly limited.

**NOTE**

The RHEL 9 documentation assumes you have system connection privileges.

**Virtual machine separation**

Individual VMs run as isolated processes on the host, and rely on security enforced by the host kernel. Therefore, a VM cannot read or access the memory or storage of other VMs on the same host.

**QEMU sandboxing**

A feature that prevents QEMU code from executing system calls that can compromise the security of the host.

**Kernel Address Space Randomization (KASLR)**

Enables randomizing the physical and virtual addresses at which the kernel image is decompressed. Thus, KASLR prevents guest security exploits based on the location of kernel objects.

### 18.6. SELINUX BOOLEANS FOR VIRTUALIZATION

For fine-grained configuration of virtual machines security on a RHEL 9 system, you can configure SELinux booleans on the host to ensure the hypervisor acts in a specific way.

To list all virtualization-related booleans and their statuses, use the **getsebool -a | grep virt** command:

```bash
$ getsebool -a | grep virt
[...]
virt_sandbox_use_netlink --> off
virt_sandbox_use_sys_admin --> off
virt_transition_userdomain --> off
virt_use_comm --> off
virt_use_execmem --> off
virt_use_fusefs --> off
[...]
```
To enable a specific boolean, use the `setsebool -P boolean_name on` command as root. To disable a boolean, use `setsebool -P boolean_name off`.

The following table lists virtualization-related booleans available in RHEL 9 and what they do when enabled:

**Table 18.1. SELinux virtualization booleans**

<table>
<thead>
<tr>
<th>SELinux Boolean</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>staff_use_svirt</td>
<td>Enables non-root users to create and transition VMs to sVirt.</td>
</tr>
<tr>
<td>unprivuser_use_svirt</td>
<td>Enables unprivileged users to create and transition VMs to sVirt.</td>
</tr>
<tr>
<td>virt_sandbox_use_audit</td>
<td>Enables sandbox containers to send audit messages.</td>
</tr>
<tr>
<td>virt_sandbox_use_netlink</td>
<td>Enables sandbox containers to use netlink system calls.</td>
</tr>
<tr>
<td>virt_sandbox_use_sys_admin</td>
<td>Enables sandbox containers to use sys_admin system calls, such as mount.</td>
</tr>
<tr>
<td>virt_transition_userdomain</td>
<td>Enables virtual processes to run as user domains.</td>
</tr>
<tr>
<td>virt_use_comm</td>
<td>Enables virt to use serial/parallel communication ports.</td>
</tr>
<tr>
<td>virt_use_execmem</td>
<td>Enables confined virtual guests to use executable memory and executable stack.</td>
</tr>
<tr>
<td>virt_use_fusefs</td>
<td>Enables virt to read FUSE mounted files.</td>
</tr>
<tr>
<td>virt_use_nfs</td>
<td>Enables virt to manage NFS mounted files.</td>
</tr>
<tr>
<td>virt_use_rawip</td>
<td>Enables virt to interact with rawip sockets.</td>
</tr>
<tr>
<td>virt_use_samba</td>
<td>Enables virt to manage CIFS mounted files.</td>
</tr>
<tr>
<td>virt_use_sanlock</td>
<td>Enables confined virtual guests to interact with the sanlock.</td>
</tr>
<tr>
<td>virt_use_usb</td>
<td>Enables virt to use USB devices.</td>
</tr>
<tr>
<td>virt_use_xserver</td>
<td>Enables virtual machine to interact with the X Window System.</td>
</tr>
</tbody>
</table>
18.7. SETTING UP IBM SECURE EXECUTION ON IBM Z

When using IBM Z hardware to run a RHEL 9 host, you can improve the security of your virtual machines (VMs) by configuring IBM Secure Execution for the VMs.

IBM Secure Execution, also known as Protected Virtualization, prevents the host system from accessing a VM’s state and memory contents. As a result, even if the host is compromised, it cannot be used as a vector for attacking the guest operating system. In addition, Secure Execution can be used to prevent untrusted hosts from obtaining sensitive information from the VM.

The following procedure describes how to convert an existing VM on an IBM Z host into a secured VM.

Prerequisites

- The system hardware is one of the following:
  - IBM z15 or later
  - IBM LinuxONE III or later
- The Secure Execution feature is enabled for your system. To verify, use:
  
  # grep facilities /proc/cpuinfo | grep 158

  If this command displays any output, your CPU is compatible with Secure Execution.
- The kernel includes support for Secure Execution. To confirm, use:
  
  # ls /sys/firmware | grep uv

  If the command generates any output, your kernel supports Secure Execution.
- The host CPU model contains the `unpack` facility. To confirm, use:
  
  # virsh domcapabilities | grep unpack
  <feature policy='require' name='unpack'/>

  If the command generates the above output, your CPU host model is compatible with Secure Execution.
- The CPU mode of the VM is set to `host-model`. To confirm this, use the following and replace `vm-name` with the name of your VM.
  
  # virsh dumpxml vm-name | grep "<cpu mode='host-model'/>"

  If the command generates any output, the VM’s CPU mode is set correctly.
- The `genprotimg` package must be installed on the host.
  
  # dnf install genprotimg

- You have obtained and verified the IBM Z host key document. For instructions to do so, see [Verifying the host key document](#) in IBM documentation.

Procedure
Do the following steps on your host:

1. Add the `prot_virt=1` kernel parameter to the boot configuration of the host.

   ```
   # grubby --update-kernel=ALL --args="prot_virt=1"
   ```

2. Enable virtio devices to use shared buffers in the VM you want to secure. To do so, use `virsh edit` to modify the XML configuration of the VM, and add `iommu='on'` to the `<driver>` line of all devices that have one. For example:

   ```
   <interface type='network'>
   <source network='default'/>
   <model type='virtio'/>
   <driver name='vhost' iommu='on'/>
   </interface>
   ```

   If a device configuration does not contain any `<driver>` line, add `<driver iommu='on'>` instead.

3. Disable memory ballooning on the VM, as the feature is not compatible with Secure Execution. To do so, add the following line to the VM’s XML configuration.

   ```
   <memballoon model='none'/>
   ```

Do the following steps in the guest operating system of the VM you want to secure.

1. Create a parameter file. For example:

   ```
   # touch ~/secure-parameters
   ```

2. In the `/boot/loader/entries` directory, identify the boot loader entry with the latest version:

   ```
   # ls /boot/loader/entries -l
   [...] -rw-r--r--. 1 root root 281 Oct 9 15:51 3ab27a195c2849429927b00679db15c1-4.18.0-240.el8.s390x.conf
   ```

3. Retrieve the kernel options line of the boot loader entry:

   ```
   # cat /boot/loader/entries/3ab27a195c2849429927b00679db15c1-4.18.0-240.el8.s390x.conf | grep options
   options root=/dev/mapper/rhel-root rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
   ```

4. Add the content of the options line and `swiotlb=262144` to the created parameters file.

   ```
   # echo "root=/dev/mapper/rhel-root rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap swiotlb=262144" > ~/secure-parameters
   ```

5. Generate an IBM Secure Execution image. For example, the following creates a `/boot/secure-image` secured image based on the `/boot/vmlinuz-4.18.0-240.el8.s390x` image, using the `secure-parameters` file, the `/boot/initramfs-4.18.0-240.el8.s390x.img` initial RAM disk file, and the `HKD-8651-000201C048.crt` host key document.
Using the `genprotimg` utility creates the secure image, which contains the kernel parameters, initial RAM disk, and boot image.

6. Update the VM’s boot menu to boot from the secure image. In addition, remove the lines starting with `initrd` and `options`, as they are not needed.
   For example, in a RHEL 8.3 VM, the boot menu can be edited in the `/boot/loader/entries/` directory:
   ```
   # cat /boot/loader/entries/3ab27a195c2849429927b00679db15c1-4.18.0-240.el8.s390x.conf
   title Red Hat Enterprise Linux 8.3
   version 4.18.0-240.el8.s390x
   linux /boot/secure-image
   [...]
   
   # shred /boot/vmlinuz-4.18.0-240.el8.s390x
   # shred /boot/initramfs-4.18.0-240.el8.s390x.img
   # shred secure-parameters
   ```

   The original boot image, the initial RAM image, and the kernel parameter file are unprotected, and if they are not removed, VMs with Secure Execution enabled can still be vulnerable to hacking attempts or sensitive data mining.

Verification

- On the host, use the `virsh dumpxml` utility to confirm the XML configuration of the secured VM. The configuration must include the `<driver iommu='on'/>` and `<memballoon model='none'/>` elements.

```
18.8. ATTACHING CRYPTOGRAPHIC COPROCESSORS TO VIRTUAL MACHINES ON IBM Z

To use hardware encryption in your virtual machine (VM) on an IBM Z host, create mediated devices from a cryptographic coprocessor device and assign them to the intended VMs. For detailed instructions, see below.

**Prerequisites**

- Your host is running on IBM Z hardware.
- The cryptographic coprocessor is compatible with device assignment. To confirm this, ensure that the type of your coprocessor is listed as CEX4 or later.

```
# lszcrypt -V
CARD.DOMAIN TYPE MODE STATUS REQUESTS PENDING HWTYPE QDEPTH
FUNCTIONS DRIVER
--------------------------------------------------------------------------------------------
 05 CEX5C CCA-Coproc online 1 0 11 08 S--D--N-- cex4card
 05.0004 CEX5C CCA-Coproc online 1 0 11 08 S--D--N-- cex4queue
 05.00ab CEX5C CCA-Coproc online 1 0 11 08 S--D--N-- cex4queue
```

- The mdevctl package is installed.
- The vfio_ap kernel module is loaded. To verify, use:

```
# lsmod | grep vfio_ap
vfio_ap 24576 0
[..]
```

To load the module, use:

```
# modprobe vfio_ap
```

**Procedure**

1. On the host, reassign your crypto device to the vfio-ap drivers. The following example assigns two crypto devices with bitmask IDs (0x05, 0x0004) and (0x05, 0x00ab) to vfio-ap.

```
# echo -0x05 > /sys/bus/ap/apmask
# echo -0x0004, -0x00ab > /sys/bus/ap/aqmask
```
2. Verify that the crypto devices have been reassigned correctly.

```bash
# lszcrypt -V
CARD.DOMAIN TYPE MODE STATUS REQUESTS PENDING HWTYPE QDEPTH FUNCTIONS DRIVER
--------------------------------------------------------------------------------------------
05 CEX5C CCA-Coproc - 1 0 11 08 S--D--N-- cex4card
05.0004 CEX5C CCA-Coproc - 1 0 11 08 S--D--N-- vfio_ap
05.00ab CEX5C CCA-Coproc - 1 0 11 08 S--D--N-- vfio_ap
```

If the DRIVER values of the domain queues changed to `vfio_ap`, the reassignment succeeded.

3. Generate a device UUID.

```bash
# uuidgen
669d9b23-fe1b-4ecb-be08-a2fabca99b71
```

In the following steps of this procedure, replace `669d9b23-fe1b-4ecb-be08-a2fabca99b71` with your generated UUID.

4. Using the UUID, create a new `vfio_ap` device.

The following example shows creating a persistent mediated device and assigning queues to it. For example, the following commands assign domain adapter `0x05` and domain queues `0x0004` and `0x00ab` to device `669d9b23-fe1b-4ecb-be08-a2fabca99b71`.

```bash
# mdevctl define --uuid 669d9b23-fe1b-4ecb-be08-a2fabca99b71 --parent matrix --type vfio_ap-passthrough
# mdevctl modify --uuid 669d9b23-fe1b-4ecb-be08-a2fabca99b71 --addattr=assign_adapter --value=0x05
# mdevctl modify --uuid 669d9b23-fe1b-4ecb-be08-a2fabca99b71 --addattr=assign_domain --value=0x0004
# mdevctl modify --uuid 669d9b23-fe1b-4ecb-be08-a2fabca99b71 --addattr=assign_domain --value=0x00ab
```

5. Start the mediated device.

```bash
# mdevctl start --uuid 669d9b23-fe1b-4ecb-be08-a2fabca99b71
```

6. Check that the configuration has been applied correctly

```bash
# cat /sys/devices/vfio_ap/matrix/mdev_supported_types/vfio_ap-passthrough/devices/669d9b23-fe1b-4ecb-be08-a2fabca99b71/matrix
05.0004
05.00ab
```

If the output contains the numerical values of queues that you have previously assigned to `vfio_ap`, the process was successful.

7. Use the `virsh edit` command to open the XML configuration of the VM where you want to use the crypto devices.
# virsh edit vm-name

8. Add the following lines to the `<devices>` section in the XML configuration, and save it.

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-ap'>
  <source>
    <address uuid='669d9b23-fe1b-4ecb-be08-a2faba99b71'/>
  </source>
</hostdev>
```

Note that each UUID can only be assigned to one VM at a time.

**Verification**

1. Start the VM to which you assigned the mediated device.

2. After the guest operating system (OS) boots, ensure that it detects the assigned crypto devices.

```
# lszcrypt -V

CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS  PENDING HWTYPE QDEPTH
FUNCTIONS  DRIVER
--------------------------------------------------------------------------------------------
05          CEX5C CCA-Coproc  online         1        0     11     08 S--D--N--  cex4card
05.0004     CEX5C CCA-Coproc  online         1        0     11     08 S--D--N--  cex4queue
05.00ab     CEX5C CCA-Coproc  online         1        0     11     08 S--D--N--  cex4queue
```

The output of this command in the guest OS will be identical to that on a host logical partition with the same cryptographic coprocessor devices available.

**18.9. ENABLING STANDARD HARDWARE SECURITY ON WINDOWS VIRTUAL MACHINES**

To secure Windows virtual machines (VMs), you can enable basic level security using the standard hardware capabilities of the Windows device.

**Prerequisites**

- Make sure you have installed the latest WHQL certified VirtIO drivers.
- Make sure the VM’s firmware supports UEFI boot.
- Install the `edk2-OVMF` package on your host machine.

```
# dnf install edk2-ovmf
```

- Install the `vTPM` packages on your host machine.

```
# dnf install swtpm libtpms
```

- Make sure the VM is using the Q35 machine architecture.
• Make sure you have the Windows installation media.

**Procedure**

1. Enable TPM 2.0 by adding the following parameters to the `<devices>` section in the VM’s XML configuration.

```xml
<devices>
  [...]  
  <tpm model='tpm-crb'>
    <backend type='emulator' version='2.0'/>
  </tpm>
  [...]  
</devices>
```

2. Install Windows in UEFI mode. For more information on how to do so, see Creating a SecureBoot virtual machine.

3. Install the VirtIO drivers on the Windows VM. For more information on how to do so, see Installing virtio drivers on a Windows guest.

4. In UEFI, enable Secure Boot. For more information on how to do so, see Secure Boot.

**Verification**

• Ensure that the **Device Security** page on your Windows machine displays the following message:

  **Settings > Update & Security > Windows Security > Device Security**

  Your device meets the requirements for standard hardware security.

**CHAPTER 18. SECURING VIRTUAL MACHINES**

18.10. ENABLING ENHANCED HARDWARE SECURITY ON WINDOWS VIRTUAL MACHINES

To further secure Windows virtual machines (VMs), you can enable virtualization-based protection of code integrity, also known as Hypervisor-Protected Code Integrity (HVIC).

**Prerequisites**

• Ensure that standard hardware security is enabled. For more information, see Enabling standard hardware security on Windows virtual machines.

• On the KVM command line,
  
  ▪ Specify the CPU model.
  
  ▪ Enable the Virtual Machine Extensions (VMX) feature.
  
  ▪ Enable Hyper-V enlightenments.

  ```
  # -cpu Skylake-Client-
  v3,hv_stimer,hv_sync,hv_relaxed,hv_reenlightenment,hv_spinlocks=0xffff,hv_vpin
dex,hv_vapic,hv_time,hv_frequencies,hv_runtime,+kvm_pv_unhalt,+vmx
  ```
Procedure

1. On your Windows VM, navigate to the Core isolation details page:
   Settings > Update & Security > Windows Security > Device Security > Core isolation details

2. Toggle the switch to enable Memory Integrity.

3. Reboot the VM.

**NOTE**

For other methods of enabling HVCI, see the relevant Microsoft documentation.

Verification

- Ensure that the Device Security page on your Windows VM displays the following message:
  Settings > Update & Security > Windows Security > Device Security
  
  Your device meets the requirements for enhanced hardware security.

- Alternatively, check System Information on the Windows VM:
  a. Run `msinfo32.exe` in a command prompt.
  b. Check if Credential Guard, Hypervisor enforced Code Integrity is listed under Virtualization-based security Services Running
CHAPTER 19. SHARING FILES BETWEEN THE HOST AND ITS VIRTUAL MACHINES

You may frequently require to share data between your host system and the virtual machines (VMs) it runs. To do so quickly and efficiently, you can set up NFS or Samba file shares on your system. As a newly supported feature in RHEL 9, you can also use the virtiofs file system to share data with your Linux VMs.

19.1. SHARING FILES BETWEEN THE HOST AND ITS VIRTUAL MACHINES USING VIRTIOFS

When using RHEL 9 as your hypervisor, you can efficiently share files between your host system and its virtual machines (VM) using the virtiofs feature.

Prerequisites

- Virtualization is installed and enabled on your RHEL 9 host.
- A directory that you want to share with your VMs. If you do not want to share any of your existing directories, create a new one, for example named shared-files.

```
# mkdir /root/shared-files
```

- The VM you want to share data with is using a Linux distribution as its guest OS.

Procedure

1. For each directory on the host that you want to share with your VM, set it as a virtiofs file system in the VM’s XML configuration.

   a. Open the XML configuration of the intended VM.

      ```
      # virsh edit vm-name
      ```

   b. Add an entry similar to the following to the `<devices>` section of the VM’s XML configuration.

      ```
      <filesystem type='mount' accessmode='passthrough'>
      <driver type='virtiofs'/>
      <binary path='/usr/libexec/virtiofsd' xattr='on'/>
      <source dir='/root/shared-files'/>
      <target dir='host-file-share'/>
      </filesystem>
      ```

      This example sets the `/root/shared-files` directory on the host to be visible as `host-file-share` to the VM.

2. Add a NUMA topology for shared memory to the XML configuration. The following example adds a basic topology for all CPUs and all RAM.

   ```
   <cpu mode='host-passthrough' check='none'>
   <numa>
   <cell id='0' cpus='0-{number-vcpus - 1}' memory='{ram-amount-KiB}' unit='KiB'
   ```
3. Add shared memory backing to the `<domain>` section of the XML configuration:

```xml
<domain>
  ...
  <memoryBacking>
    <access mode='shared'/>
  </memoryBacking>
  ...
</domain>
```

4. Boot up the VM.

```bash
# virsh start vm-name
```

5. Mount the file system in the guest operating system (OS). The following example mounts the previously configured `host-file-share` directory with a Linux guest OS.

```bash
# mount -t virtiofs host-file-share /mnt
```

**Verification**

- Ensure that the shared directory became accessible on the VM and that you can now open files stored in the directory.

**Known issues and limitations**

- File-system mount options related to access time, such as `noatime` and `strictatime`, are not likely to work with virtiofs, and Red Hat discourages their use.

**Troubleshooting**

- If `virtiofs` is not optimal for your usecase or supported for your system, you can use `NFS` or `Samba` instead.

### 19.2. USING THE WEB CONSOLE TO SHARE FILES BETWEEN THE HOST AND ITS VIRTUAL MACHINES USING VIRTIOFS

You can use the RHEL web console to efficiently share files between your host system and its virtual machines (VM) using the `virtiofs` feature.

**Prerequisites**

- The web console VM plug-in is installed on your system.
- A directory that you want to share with your VMs. If you do not want to share any of your existing directories, create a new one, for example named `centurion`.

```bash
# mkdir /home/centurion
```
The VM you want to share data with is using a Linux distribution as its guest OS.

**Procedure**

1. In the **Virtual Machines** interface, click the VM with which you want to share files. A new page opens with an **Overview** section with basic information about the selected VM and a **Console** section.

2. Scroll to **Shared directories**. The **Shared directories** section displays information about the host files and directories shared with that VM and options to **Add** or **Remove** a shared directory.

3. Click **Add shared directory**. The **Share a host directory with the guest** dialog appears.

4. Enter the following information:
   - **Source path** - The path to the host directory that you want to share.
   - **Mount tag** - The tag that the VM uses to mount the directory.

5. Set additional options:
   - **Extended attributes** - Set whether to enable extended attributes, `xattr`, on the shared files and directories.

6. Click **Share**. The selected directory is shared with the VM.

**Verification**

- Ensure that the shared directory is accessible on the VM and you can now open files stored in that directory.
19.3. USING THE WEB CONSOLE TO REMOVE SHARED FILES BETWEEN THE HOST AND ITS VIRTUAL MACHINES USING VIRTIOFS

You can use the RHEL web console to remove files shared between your host system and its virtual machines (VM) using the `virtiofs` feature.

Prerequisites

- The web console VM plug-in is installed on your system.
- The directory is no longer being used by the VM.

Procedure

1. In the **Virtual Machines** interface, click on the VM from which you want to remove the shared files.
   A new page opens with an **Overview** section with basic information about the selected VM and a **Console** section.

2. Scroll to **Shared directories**.
   The **Shared directories** section displays information about the host files and directories shared with that VM and options to **Add** or **Remove** a shared directory.

   ![Shared directories](image)

3. Click **Remove** next to the directory you wish to unshare with the VM.
   The **Remove filesystem** dialog appears.

   ![Remove filesystem](image)

4. Click **Remove**.
   The selected directory is unshared with the VM.

Verification

- The shared directory is no longer available and accessible on the VM.

19.4. SHARING FILES BETWEEN THE HOST AND LINUX VIRTUAL MACHINES USING NFS

For efficient file sharing between your RHEL 9 host system and the Linux VMs it is connected to, use the `virtiofs` feature. However, if `virtiofs` is not working for you or does not suit your use case, you can instead export an NFS share that your VMs can mount and access.
Prerequisites

- The **nfs-utils** package is installed on the host.
- A directory that you want to share with your VMs. If you do not want to share any of your existing directories, create a new one, for example named `shared-files`.

```bash
# mkdir shared-files
```

- The host is visible and reachable over a network for the VM. This is generally the case if the VM is connected using the `NAT` and `bridge` type of virtual networks. However, for the `macvtap` connection, you must first set up the `macvlan` feature on the host. To do so:

  1. Create a network device file in the host’s `/etc/systemd/network/` directory, for example called `vm-macvlan.netdev`.

```bash
# vim /etc/systemd/network/vm-macvlan.netdev
```

  2. Edit the network device file to have the following content. You can replace `vm-macvlan` with the name you chose for your network device.

```ini
[NetDev]
Name=vm-macvlan
Kind=macvlan

[MACVLAN]
Mode=bridge
```

  3. Create a network configuration file for your macvlan network device, for example `vm-macvlan.network`.

```bash
# vim /etc/systemd/network/vm-macvlan.network
```

  4. Edit the network configuration file to have the following content. You can replace `vm-macvlan` with the name you chose for your network device.

```ini
[Match]
Name=_vm-macvlan_

[Network]
IPForward=yes
Address=192.168.250.33/24
Gateway=192.168.250.1
DNS=192.168.250.1
```

  5. Create a network configuration file for your physical network interface. For example, if your interface is `enp4s0`:

```bash
# vim /etc/systemd/network/enp4s0.network
```

If you are unsure what interface name to use, you can use the `ifconfig` command on your host to obtain the list of active network interfaces.
6. Edit the physical network configuration file to make the physical network a part of the macvlan interface, in this case `vm-macvlan`:

```yaml
[Match]
Name=enp4s0

[Network]
MACVLAN=vm-macvlan
```

7. Reboot your host.

- **Optional:** For improved security, ensure your VMs are compatible with NFS version 4 or later.

**Procedure**

1. On the host, export a directory with the files you want to share as a network file system (NFS).
   
   a. Obtain the IP address of each virtual machine you want to share files with. The following example obtains the IPs of `testguest1` and `testguest2`.

   ```bash
   # virsh domifaddr testguest1
   Name       MAC address          Protocol     Address
   vnet0      52:53:00:84:57:90    ipv4         192.168.124.220/24
   
   # virsh domifaddr testguest2
   Name       MAC address          Protocol     Address
   vnet1      52:53:00:65:29:21    ipv4         192.168.124.17/24
   
   b. Edit the `/etc/exports` file on the host and add a line that includes the directory you want to share, IPs of VMs you want to share with, and sharing options.

   ```
   Shared directory  VM1-IP(options)  VM2-IP(options)  [...]  
   ```

   For example, the following shares the `/usr/local/shared-files` directory on the host with `testguest1` and `testguest2`, and enables the VMs to edit the content of the directory:

   ```
   ```

   c. Export the updated file system.

   ```bash
   # exportfs -a
   ```

   d. Ensure the NFS process is started:

   ```bash
   # systemctl start nfs-server
   ```

   e. Obtain the IP address of the host system. This will be used for mounting the shared directory on the VMs later.

   ```bash
   # ip addr
   [...]  
   5: virbr0: [BROADCAST,MULTICAST,UP,LOWER_UP] mtu 1500 qdisc noqueue state
UP group default qlen 100
link/ether 52:54:00:32:ff:a5 brd ff:ff:ff:ff:ff:ff
inet 192.168.124.1/24 brd 192.168.124.255 scope global virbr0
valid_lft forever preferred_lft forever

Note that the relevant network is the one being used for connection to the host by the VMs you want to share files with. Usually, this is virbr0.

2. On the guest OS of a VM specified in the /etc/exports file, mount the exported file system.
   a. Create a directory you want to use as a mount point for the shared file system, for example /mnt/host-share:

   ```
   # mkdir /mnt/host-share
   ```
   b. Mount the directory exported by the host on the mount point. This example mounts the /usr/local/shared-files directory exported by the 192.168.124.1 host on /mnt/host-share in the guest:

   ```
   # mount 192.168.124.1:/usr/local/shared-files /mnt/host-share
   ```

Verification

- To verify the mount has succeeded, access and explore the shared directory on the mount point:

  ```
  # cd /mnt/host-share
  # ls
  shared-file1  shared-file2  shared-file3
  ```

19.5. SHARING FILES BETWEEN THE HOST AND WINDOWS VIRTUAL MACHINES USING SAMBA

For efficient file sharing between your RHEL 9 host system and the Windows VMs it is connected to, use the virtiofs feature. However, if virtiofs is not working for you or does not suit your use case, you can instead prepare a Samba server that your VMs can access.

Prerequisites

- The samba packages are installed on your host. If they are not:

  ```
  # dnf install samba
  ```

- The host is visible and reachable over a network for the VM. This is generally the case if the VM is connected using the NAT and bridge type of virtual networks. However, for the macvtap connection, you must first set up the macvlan feature on the host. To do so:

  1. Create a network device file, for example called vm-macvlan.netdev in the host’s /etc/systemd/network/ directory.

     ```
     # vim /etc/systemd/network/vm-macvlan.netdev
     ```
2. Edit the network device file to have the following content. You can replace \texttt{vm-macvlan}
   with the name you chose for your network device.

   \begin{verbatim}[NetDev]
   Name=vm-macvlan
   Kind=macvlan
   
   [MACVLAN]
   Mode=bridge
   \end{verbatim}

3. Create a network configuration file for your macvlan network device, for example \texttt{vm-macvlan.network}.

   \begin{verbatim}
   # vim /etc/systemd/network/vm-macvlan.network
   \end{verbatim}

4. Edit the network configuration file to have the following content. You can replace \texttt{vm-macvlan}
   with the name you chose for your network device.

   \begin{verbatim}[Match]
   Name=_vm-macvlan_
   
   [Network]
   IPForward=yes
   Address=192.168.250.33/24
   Gateway=192.168.250.1
   DNS=192.168.250.1
   \end{verbatim}

5. Create a network configuration file for your physical network interface. For example, if your
   interface is \texttt{enp4s0}:

   \begin{verbatim}
   # vim /etc/systemd/network/enp4s0.network
   \end{verbatim}

   If you are unsure what interface to use, you can use the \texttt{ifconfig} command on your host to
   obtain the list of active network interfaces.

6. Edit the physical network configuration file to make the physical network a part of the
   macvlan interface, in this case \texttt{vm-macvlan}:

   \begin{verbatim}[Match]
   Name=enp4s0
   
   [Network]
   MACVLAN=vm-macvlan
   \end{verbatim}

7. Reboot your host.

Procedure

1. On the host, create a Samba share and make it accessible for external systems.
   a. Add firewall permissions for Samba.

   \begin{verbatim}
   # firewall-cmd --permanent --zone=public --add-service=samba
   success
   \end{verbatim}
b. Edit the `/etc/samba/smb.conf` file:
   
   i. Add the following to the `[global]` section:

   ```
   map to guest = Bad User
   ```

   ii. Add the following at the end of the file:

   ```
   #=== Share Definitions ===
   [VM-share]
   path = /samba/VM-share
   browsable = yes
   guest ok = yes
   read only = no
   hosts allow = 192.168.122.0/24
   ```

   Note that the `hosts allow` line restricts the accessibility of the share only to hosts on the VM network. If you want the share to be accessible by anyone, remove the line.

c. Create the `/samba/VM-share` directory.

   ```
   # mkdir -p /samba/VM-share
   ```

d. Enable the Samba service.

   ```
   # systemctl enable smb.service
   Created symlink /etc/systemd/system/multi-user.target.wants/smb.service → /usr/lib/systemd/system/smb.service.
   ```

e. Restart the Samba service.

   ```
   # systemctl restart smb.service
   ```

f. Allow the `VM-share` directory to be accessible and modifiable for the VMs.

   ```
   # chmod -R 0755 /samba/VM-share/
   # chown -R nobody:nobody /samba/VM-share/
   ```

g. Add the SELinux Samba sharing label to `/etc/samba/VM-share/`

   ```
   # chcon -t samba_share_t /samba/VM-share/
   ```

2. On the Windows guest operating system, attach the Samba share as a network location.
   
a. Open the File Explorer and right-click "This PC".

   b. In the context menu, click **Add a network location**.
c. In the *Add Network Location* wizard that opens, select "Choose a custom network location" and click **Next**.

d. In the "Internet or network address" field, type host-IP/VM-share, where *host-IP* is the IP address of the host. Usually, the host IP is the default gateway of the VM. Afterwards, click **Next**.

e. When the wizard asks if you want to rename the shared directory, keep the default name. This ensures the consistency of file sharing configuration across the VM and the guest. Click **Next**.

f. If accessing the network location was successful, you can now click **Finish** and open the shared directory.
CHAPTER 20. INSTALLING AND MANAGING WINDOWS VIRTUAL MACHINES

To use Microsoft Windows as the guest operating system in your virtual machines (VMs) on a RHEL 9 host, Red Hat recommends taking extra steps to ensure these VMs run correctly.

For this purpose, the following sections provide information on installing and optimizing Windows VMs on the host, as well as installing and configuring drivers in these VMs.

20.1. INSTALLING WINDOWS VIRTUAL MACHINES

You can create a fully-virtualized Windows machine on a RHEL 9 host, launch the graphical Windows installer inside the virtual machine (VM), and optimize the installed Windows guest operating system (OS).

To create the VM and to install the Windows guest OS, use the `virt-install` command or the RHEL 9 web console.

Prerequisites

- A Windows OS installation source, which can be one of the following, and be available locally or on a network:
  - An ISO image of an installation medium
  - A disk image of an existing VM installation

- A storage medium with the KVM `virtio` drivers. To create this medium, see Preparing virtio driver installation media on a host machine.

- Optional: If you are installing Windows 11, you must install the vTPM packages on your host machine.

```
# yum install swtpm libtpms
```

Procedure

1. Create the VM. For instructions, see Creating virtual machines.

   - If using the `virt-install` utility to create the VM, add the following options to the command:
     - The storage medium with the KVM `virtio` drivers. For example:

       ```
       --disk path=/usr/share/virtio-win/virtio-win.iso,device=cdrom
       ```

     - The Windows version you will install. For example, for Windows 11:

       ```
       --os-variant win11
       ```

     For a list of available Windows versions and the appropriate option, use the following command:

     ```
     # osinfo-query os
     ```
If using the web console to create the VM, specify your version of Windows in the Operating System field of the Create New Virtual Machine window. After the VM is created and the guest OS is installed, attach the storage medium with virtio drivers to the VM using the Disks interface. For instructions, see Attaching existing disks to virtual machines using the web console.

2. Install the Windows OS in the VM. For information on how to install a Windows operating system, refer to the relevant Microsoft installation documentation.

3. Configure KVM virtio drivers in the Windows guest OS. For details, see Installing KVM paravirtualized drivers for Windows virtual machines.

Additional resources

- Optimizing Windows virtual machines
- Enabling standard hardware security on Windows virtual machines
- Enabling enhanced hardware security on Windows virtual machines

20.2. OPTIMIZING WINDOWS VIRTUAL MACHINES

When using Microsoft Windows as a guest operating system in a virtual machine (VM) hosted in RHEL 9, the performance of the guest may be negatively impacted.

Therefore, Red Hat recommends optimizing your Windows VMs by doing any combination of the following:

- Using paravirtualized drivers. For more information, see Installing KVM paravirtualized drivers for Windows virtual machines.
- Enabling Hyper-V enlightenments. For more information, see Enabling Hyper-V enlightenments.
- Configuring NetKVM driver parameters. For more information, see Configuring NetKVM driver parameters.
- Optimizing or disabling Windows background processes. For more information, see Optimizing background processes on Windows virtual machines.

20.2.1. Installing KVM paravirtualized drivers for Windows virtual machines

The primary method of improving the performance of your Windows virtual machines (VMs) is to install KVM paravirtualized (virtio) drivers for Windows on the guest operating system (OS).

To do so:

1. Prepare the install media on the host machine. For more information, see Preparing virtio driver installation media on a host machine.

2. Attach the install media to an existing Windows VM, or attach it when creating a new Windows VM.

3. Install the virtio drivers on the Windows guest OS. For more information, see Installing virtio drivers on a Windows guest.
20.2.1. How Windows virtio drivers work

Paravirtualized drivers enhance the performance of virtual machines (VMs) by decreasing I/O latency and increasing throughput to almost bare-metal levels. Red Hat recommends that you use paravirtualized drivers for VMs that run I/O-heavy tasks and applications.

**virtio** drivers are KVM's paravirtualized device drivers, available for Windows VMs running on KVM hosts. These drivers are provided by the **virtio-win** package, which includes drivers for:

- Block (storage) devices
- Network interface controllers
- Video controllers
- Memory ballooning device
- Paravirtual serial port device
- Entropy source device
- Paravirtual panic device
- Input devices, such as mice, keyboards, or tablets
- A small set of emulated devices

**NOTE**

For additional information about emulated, **virtio**, and assigned devices, refer to Managing virtual devices.

Using KVM virtio drivers, the following Microsoft Windows versions are expected to run similarly to physical systems:


- Windows Desktop (non-server) versions:
  - Windows 7 (32-bit and 64-bit versions)
  - Windows 8 (32-bit and 64-bit versions)
  - Windows 8.1 (32-bit and 64-bit versions)
  - Windows 10 (32-bit and 64-bit versions)
  - Windows 11 (64-bit)

20.2.1.2. Preparing virtio driver installation media on a host machine

To install KVM virtio drivers on a Windows virtual machine (VM), you must first prepare the installation media for the virtio driver on the host machine. To do so, install the **virtio-win** package on the host machine and use the .iso file it provides as storage for the VM.
Prerequisites

- Ensure that virtualization is enabled in your RHEL 9 host system. For more information, see Enabling virtualization.

Procedure

1. Download the drivers
   a. Browse to Download Red Hat Enterprise Linux.
   b. Select the Product Variant relevant for your system architecture. For example, for Intel 64 and AMD64, select Red Hat Enterprise Linux for x86_64.
   c. Select the Version of your host system.
   d. In the Packages tab, search for virtio-win.
   e. Click Download Latest next to the virtio-win AppStream package. The RPM file downloads.

2. Install the virtio-win package from the download directory. For example:

   ```bash
   # dnf install ~/Downloads/virtio-win-1.9.9-3.el8.noarch.rpm
   [...]
   Installed:
   virtio-win-1.9.9-3.el8.noarch
   ```

   If the installation succeeds, the virtio-win driver files are prepared in the /usr/share/virtio-win/ directory. These include ISO files and a drivers directory with the driver files in directories, one for each architecture and supported Windows version.

   ```bash
   # ls /usr/share/virtio-win/
   drivers/ guest-agent/ virtio-win-1.9.9.iso virtio-win.iso
   ```

3. Attach the virtio-win.iso file to the Windows VM. To do so, do one of the following:
   - Use the file as a disk when creating a new Windows VM.
   - Add the file as a CD-ROM to an existing Windows VM. For example:

     ```bash
     # virt-xml WindowsVM --add-device --disk virtio-win.iso,device=cdrom
     Domain 'WindowsVM' defined successfully.
     ```

Next steps

- When virtio-win.iso is attached to the Windows VM, you can proceed to installing the virtio driver on the Windows guest operating system.

20.2.1.3. Installing virtio drivers on a Windows guest

To install KVM virtio drivers on a Windows guest operating system (OS), you must add a storage device that contains the drivers - either when creating the virtual machine (VM) or afterwards - and install the drivers in the Windows guest OS.
This example shows how to install the drivers using the graphical interface. You can also use the Microsoft Windows Installer (MSI) command line interface.

Prerequisites

- An installation medium with the KVM virtio drivers must be attached to the VM. For instructions on preparing the medium, see Preparing virtio driver installation media on a host machine.

Procedure

1. In the Windows guest OS, open the File Explorer application.
2. Click This PC.
3. In the Devices and drives pane, open the virtio-win medium.
4. Based on the architecture of the VM’s vCPU, run one of the installers on the medium.
   - If using a 32-bit vCPU, run the virtio-win-gt-x86 installer.
   - If using a 64-bit vCPU, run the virtio-win-gt-x64 installer.
5. In the Virtio-win-guest-tools setup wizard that opens, follow the displayed instructions until you reach the Custom Setup step.
6. In the Custom Setup window, select the device drivers you want to install. The recommended driver set is selected automatically, and the descriptions of the drivers are displayed on the right of the list.

7. Click **next**, then click **Install**.

8. After the installation completes, click **Finish**.

9. Reboot the VM to complete the driver installation.

**Verification**

1. In **This PC**, open the system disk. This is typically **(C:)**.

2. In the **Program Files** directory, open the **Virtio-Win** directory.
   If the **Virtio-Win** directory is present and contains a sub-directory for each of the selected drivers, the installation was successful.
Next steps

- If you install the NetKVM driver, you may also need to configure the Windows guest’s networking parameters.

20.2.2. Enabling Hyper-V enlightenments

Hyper-V enlightenments provide a method for KVM to emulate the Microsoft Hyper-V hypervisor. This improves the performance of Windows virtual machines.

The following sections provide information about the supported Hyper-V enlightenments and how to enable them.

20.2.2.1. Enabling Hyper-V enlightenments on a Windows virtual machine

Hyper-V enlightenments provide better performance in a Windows virtual machine (VM) running in a RHEL 9 host. For instructions on how to enable them, see the following.

Procedure

1. Use the `virsh edit` command to open the XML configuration of the VM. For example:

   ```bash
   # virsh edit windows-vm
   ```

2. Add the following `<hyperv>` sub-section to the `<features>` section of the XML:

   ```xml
   <features>
   [...]
   <hyperv>
     <relaxed state="on"/>
     <vapic state="on"/>
     <spinlocks state="on" retries="8191"/>
   </hyperv>
   ```
If the XML already contains a `<hyperv>` sub-section, modify it as shown above.

3. Change the `clock` section of the configuration as follows:

```xml
<clock offset='localtime'>
  ...
  <timer name='hypervclock' present='yes'/>
</clock>
```

4. Save and exit the XML configuration.

5. If the VM is running, restart it.

**Verification**

- Use the `virsh dumpxml` command to display the XML configuration of the running VM. If it includes the following segments, the Hyper-V enlightenments are enabled on the VM.

```xml
<hyperv>
  <relaxed state='on'/>
  <vapid state='on'/>
  <spinlocks state='on' retries='8191'/>
  <vpindex state='on'/>
  <runtime state='on'/>
  <synic state='on'/>
  <frequencies state='on'/>
  <stimer state='on'/>
  <direct state='on'/>
  </stimer>
  </hyperv>
```
You can configure certain Hyper-V features to optimize Windows VMs. The following table provides information about these configurable Hyper-V features and their values.

<table>
<thead>
<tr>
<th>Enlightenment</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>crash</td>
<td>Provides MSRs to the VMs that can be used to store information and logs if a VM crashes. The information is available in the QEMU log.</td>
<td>on, off</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>If hv_crash is enabled, Windows crash dumps are not created.</td>
<td></td>
</tr>
<tr>
<td>evmcs</td>
<td>Implements paravirtualized protocol between L0 (KVM) and L1 (Hyper-V) hypervisors, which enables faster L2 exits to the hypervisor.</td>
<td>on, off</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>This feature is exclusive to Intel processors.</td>
<td></td>
</tr>
<tr>
<td>frequencies</td>
<td>Enables Hyper-V frequency Machine Specific Registers (MSRs).</td>
<td>on, off</td>
</tr>
<tr>
<td>ippi</td>
<td>Enables paravirtualized inter processor interrupts (IPI) support.</td>
<td>on, off</td>
</tr>
<tr>
<td>Enlightenment</td>
<td>Description</td>
<td>Values</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>no-nonarch-coresharing</td>
<td>Notifies the guest OS that virtual processors will never share a physical core unless they are reported as sibling SMT threads. This information is required by Windows and Hyper-V guests to properly mitigate simultaneous multithreading (SMT) related CPU vulnerabilities.</td>
<td>on, off, auto</td>
</tr>
<tr>
<td>reenlightenment</td>
<td>Notifies when there is a time stamp counter (TSC) frequency change which only occurs during migration. It also allows the guest to keep using the old frequency until it is ready to switch to the new one.</td>
<td>on, off</td>
</tr>
<tr>
<td>relaxed</td>
<td>Disables a Windows sanity check that commonly results in a BSOD when the VM is running on a heavily loaded host. This is similar to the Linux kernel option no_timer_check, which is automatically enabled when Linux is running on KVM.</td>
<td>on, off</td>
</tr>
<tr>
<td>runtime</td>
<td>Sets processor time spent on running the guest code, and on behalf of the guest code.</td>
<td>on, off</td>
</tr>
<tr>
<td>Enlightenment</td>
<td>Description</td>
<td>Values</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>spinlocks</td>
<td>Used by a VM’s operating system to notify Hyper-V that the calling virtual processor is attempting to acquire a resource that is potentially held by another virtual processor within the same partition.</td>
<td>on, off</td>
</tr>
<tr>
<td></td>
<td>Used by Hyper-V to indicate to the virtual machine’s operating system the number of times a spinlock acquisition should be attempted before indicating an excessive spin situation to Hyper-V.</td>
<td></td>
</tr>
<tr>
<td>stimer</td>
<td>Enables synthetic timers for virtual processors. Note that certain Windows versions revert to using HPET (or even RTC when HPET is unavailable) when this enlightenment is not provided, which can lead to significant CPU consumption, even when the virtual CPU is idle.</td>
<td>on, off</td>
</tr>
<tr>
<td>stimer-direct</td>
<td>Enables synthetic timers when an expiration event is delivered via a normal interrupt.</td>
<td>on, off</td>
</tr>
<tr>
<td>synic</td>
<td>Together with stimer, activates the synthetic timer. Windows 8 uses this feature in periodic mode.</td>
<td>on, off</td>
</tr>
<tr>
<td>time</td>
<td>Enables the following Hyper-V-specific clock sources available to the VM,</td>
<td>on, off</td>
</tr>
<tr>
<td></td>
<td>- MSR-based 82 Hyper-V clock source (HV_X64_MSR_TIME_REF_COUNT, 0x40000020)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Reference TSC 83 page which is enabled via MSR (HV_X64_MSR_REFERENCE_TSC, 0x40000021)</td>
<td></td>
</tr>
</tbody>
</table>
### 20.2.3. Configuring NetKVM driver parameters

After the NetKVM driver is installed, you can configure it to better suit your environment. The parameters listed in this section can be configured using the Windows Device Manager (devmgmt.msc).

**IMPORTANT**

Modifying the driver’s parameters causes Windows to reload that driver. This interrupts existing network activity.

**Prerequisites**

- The NetKVM driver is installed on the virtual machine.
  
  For more information, see [Installing KVM paravirtualized drivers for Windows virtual machines](#).

**Procedure**

1. Open Windows Device Manager.
   
   For information on opening Device Manager, refer to the Windows documentation.

2. Locate the **Red Hat VirtIO Ethernet Adapter**.
   
   a. In the Device Manager window, click + next to Network adapters.
   
   b. Under the list of network adapters, double-click **Red Hat VirtIO Ethernet Adapter**. The Properties window for the device opens.

3. View the device parameters.
   
   In the Properties window, click the **Advanced** tab.

4. Modify the device parameters.
   
   a. Click the parameter you want to modify.
Options for that parameter are displayed.

b. Modify the options as needed.
   For information on the NetKVM parameter options, refer to NetKVM driver parameters.

c. Click OK to save the changes.

20.2.4. NetKVM driver parameters

The following table provides information on the configurable NetKVM driver logging parameters.

Table 20.2. Logging parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logging.Enable</td>
<td>A Boolean value that determines whether logging is enabled. The default value is Enabled.</td>
</tr>
<tr>
<td>Logging.Level</td>
<td>An integer that defines the logging level. As the integer increases, so does the verbosity of the log.</td>
</tr>
<tr>
<td></td>
<td>- The default value is 0 (errors only).</td>
</tr>
<tr>
<td></td>
<td>- 1-2 adds configuration messages.</td>
</tr>
<tr>
<td></td>
<td>- 3-4 adds packet flow information.</td>
</tr>
<tr>
<td></td>
<td>- 5-6 adds interrupt and DPC level trace information.</td>
</tr>
</tbody>
</table>

NOTE

High logging levels will slow down your virtual machine.

The following table provides information on the configurable NetKVM driver initial parameters.

Table 20.3. Initial parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign MAC</td>
<td>A string that defines the locally-administered MAC address for the paravirtualized NIC. This is not set by default.</td>
</tr>
<tr>
<td>Init.ConnectionRate(Mb)</td>
<td>An integer that represents the connection rate in megabits per second. The default value for Windows 2008 and later is 10G (10,000 megabits per second).</td>
</tr>
<tr>
<td>Init.Do802.1PQ</td>
<td>A Boolean value that enables Priority/VLAN tag population and removal support. The default value is Enabled.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Init.MTUSize</td>
<td>An integer that defines the maximum transmission unit (MTU). The default value is 1500. Any value from 500 to 65500 is acceptable.</td>
</tr>
<tr>
<td>Init.MaxTxBuffers</td>
<td>An integer that represents the number of TX ring descriptors that will be allocated. The default value is 1024.</td>
</tr>
<tr>
<td></td>
<td>Valid values are: 16, 32, 64, 128, 256, 512, and 1024.</td>
</tr>
<tr>
<td>Init.MaxRxBuffers</td>
<td>An integer that represents the number of RX ring descriptors that will be allocated. The default value is 256.</td>
</tr>
<tr>
<td></td>
<td>Valid values are: 16, 32, 64, 128, 256, 512, and 1024.</td>
</tr>
<tr>
<td>Offload.Tx.Checksum</td>
<td>Specifies the TX checksum offloading mode.</td>
</tr>
<tr>
<td></td>
<td>In Red Hat Enterprise Linux 9, the valid values for this parameter are:</td>
</tr>
<tr>
<td></td>
<td>* All (the default) which enables IP, TCP, and UDP checksum offloading for both IPv4 and IPv6</td>
</tr>
<tr>
<td></td>
<td>* TCP/UDP(v4,v6) which enables TCP and UDP checksum offloading for both IPv4 and IPv6</td>
</tr>
<tr>
<td></td>
<td>* TCP/UDP(v4) which enables TCP and UDP checksum offloading for IPv4 only</td>
</tr>
<tr>
<td></td>
<td>* TCP(v4) which enables only TCP checksum offloading for IPv4 only</td>
</tr>
</tbody>
</table>

### 20.2.5. Optimizing background processes on Windows virtual machines

To optimize the performance of a virtual machine (VM) running a Windows OS, you can configure or disable a variety of Windows processes.

**WARNING**

Certain processes might not work as expected if you change their configuration.

**Procedure**
You can optimize your Windows VMs by performing any combination of the following:

- Remove unused devices, such as USBs or CD-ROMs, and disable the ports.
- Disable background services, such as SuperFetch and Windows Search. For more information about stopping services, see Disabling system services or Stop-Service.
- Disable `useplatformclock`. To do so, run the following command,

  ```
  # bcdedit /set useplatformclock No
  ```

- Review and disable unnecessary scheduled tasks, such as scheduled disk defragmentation. For more information on how to do so, see Disable Scheduled Tasks.
- Make sure the disks are not encrypted.
- Reduce periodic activity of server applications. You can do so by editing the respective timers. For more information, see Multimedia Timers.
- Close the Server Manager application on the VM.
- Disable the antivirus software. Note that disabling the antivirus might compromise the security of the VM.
- Disable the screen saver.
- Keep the Windows OS on the sign-in screen when not in use.

### 20.3. ENABLING STANDARD HARDWARE SECURITY ON WINDOWS VIRTUAL MACHINES

To secure Windows virtual machines (VMs), you can enable basic level security using the standard hardware capabilities of the Windows device.

#### Prerequisites

- Make sure you have installed the latest WHQL certified VirtIO drivers.
- Make sure the VM’s firmware supports UEFI boot.
- Install the `edk2-ovmf` package on your host machine.

  ```
  # dnf install edk2-ovmf
  ```

- Install the `vTPM` packages on your host machine.

  ```
  # dnf install swtpm libtpms
  ```

- Make sure the VM is using the Q35 machine architecture.
- Make sure you have the Windows installation media.

#### Procedure
1. Enable TPM 2.0 by adding the following parameters to the `<devices>` section in the VM’s XML configuration.

```
<devices>
  [...]  
  <tpm model='tpm-crb'>
    <backend type='emulator' version='2.0'/>
  </tpm>
  [...]  
</devices>
```

2. Install Windows in UEFI mode. For more information on how to do so, see Creating a SecureBoot virtual machine.

3. Install the VirtIO drivers on the Windows VM. For more information on how to do so, see Installing virtio drivers on a Windows guest.

4. In UEFI, enable Secure Boot. For more information on how to do so, see Secure Boot.

**Verification**

- Ensure that the Device Security page on your Windows machine displays the following message:
  
  Settings > Update & Security > Windows Security > Device Security

  Your device meets the requirements for standard hardware security.

### 20.4. enabling enhanced hardware security on windows virtual machines

To further secure Windows virtual machines (VMs), you can enable virtualization-based protection of code integrity, also known as Hypervisor-Protected Code Integrity (HVIC).  

**Prerequisites**

- Ensure that standard hardware security is enabled. For more information, see Enabling standard hardware security on Windows virtual machines.

- On the KVM command line,
  - Specify the CPU model.
  - Enable the Virtual Machine Extensions (VMX) feature.
  - Enable Hyper-V enlightenments.

  # -cpu Skylake-Client-v3,hv_stimer,hv_synic,hv_relaxed,hv_reenlightenment,hv_spinlocks=0xfff,hv_vpin dex,hv_vapic,hv_time,hv_frequencies,hv_runtime,+kvm_pv_unhalt,+vmx

**Procedure**

1. On your Windows VM, navigate to the Core isolation details page:
   
   Settings > Update & Security > Windows Security > Device Security > Core isolation details
2. Toggle the switch to enable Memory Integrity.

3. Reboot the VM.

**NOTE**

For other methods of enabling HVCI, see the relevant Microsoft documentation.

**Verification**

- Ensure that the Device Security page on your Windows VM displays the following message:
  
  **Settings > Update & Security > Windows Security > Device Security**

  Your device meets the requirements for enhanced hardware security.

- Alternatively, check System Information on the Windows VM:
  
  a. Run `msinfo32.exe` in a command prompt.
  
  b. Check if Credential Guard, Hypervisor enforced Code Integrity is listed under Virtualization-based security Services Running

**20.5. NEXT STEPS**

- To use utilities for accessing, editing, and creating virtual machine disks or other disk images for a Windows VM, install the `libguestfs-tools` and `libguestfs-winsupport` packages on the host machine:

  
  $ sudo dnf install libguestfs-tools libguestfs-winsupport

- To use utilities for accessing, editing, and creating virtual machine disks or other disk images for a Windows VM, install the `guestfs-tools` and `guestfs-winsupport` packages on the host machine:

  
  $ sudo dnf install guestfs-tools guestfs-winsupport

- To share files between your RHEL 9 host and its Windows VMs, you can use virtiofs or Samba.
CHAPTER 21. DIAGNOSING VIRTUAL MACHINE PROBLEMS

When working with virtual machines (VMs), you may encounter problems with varying levels of severity. Some problems may have a quick and easy fix, while for others, you may have to capture VM-related data and logs to report or diagnose the problems.

The following sections provide detailed information about generating logs and diagnosing some common VM problems, as well as about reporting these problems.

21.1. GENERATING LIBVIRT DEBUG LOGS

To diagnose virtual machine (VM) problems, it is helpful to generate and review libvirt debug logs. Attaching debug logs is also useful when asking for support to resolve VM-related problems.

The following sections explain what debug logs are, how you can set them to be persistent, enable them during runtime, and attach them when reporting problems.

21.1.1. Understanding libvirt debug logs

Debug logs are text files that contain data about events that occur during virtual machine (VM) runtime. The logs provide information about fundamental server-side functionalities, such as host libraries and the libvirt daemon. The log files also contain the standard error output (stderr) of all running VMs.

Debug logging is not enabled by default and has to be enabled when libvirt starts. You can enable logging for a single session or persistently. You can also enable logging when a libvirt daemon session is already running by modifying the daemon run-time settings.

Attaching the libvirt debug logs is also useful when requesting support with a VM problem.

21.1.2. Enabling persistent settings for libvirt debug logs

You can configure libvirt debug logging to be automatically enabled whenever libvirt starts. By default, virtqemud is the main libvirt daemon in RHEL 9. To make persistent changes in the libvirt configuration, you must edit the virtqemud.conf file, located in the /etc/libvirt directory.

NOTE

In some cases, for example when you upgrade from RHEL 8, libvirtd might still be the enabled libvirt daemon. In that case, you must edit the libvirtd.conf file instead.

Procedure

1. Open the virtqemud.conf file in an editor.

2. Replace or set the filters according to your requirements.

<table>
<thead>
<tr>
<th>Table 21.1. Debugging filter values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 logs all messages generated by libvirt.</td>
</tr>
<tr>
<td>2 logs all non-debugging information.</td>
</tr>
<tr>
<td>3 logs all warning and error messages. This is the default value.</td>
</tr>
</tbody>
</table>
Example 21.1. Sample daemon settings for logging filters

The following settings:

- Log all error and warning messages from the `remote`, `util.json`, and `rpc` layers
- Log only error messages from the `event` layer.
- Save the filtered logs to `/var/log/libvirt/libvirt.log`

```
log_filters="3:remote 4:event 3:util.json 3:rpc"
log_outputs="1:file:/var/log/libvirt/libvirt.log"
```

3. Save and exit.

4. Restart the libvirt daemon.

   ```
   $ systemctl restart virtqemud.service
   ```

21.1.3. Enabling libvirt debug logs during runtime

You can modify the libvirt daemon’s runtime settings to enable debug logs and save them to an output file.

This is useful when restarting the libvirt daemon is not possible because restarting fixes the problem, or because there is another process, such as migration or backup, running at the same time. Modifying runtime settings is also useful if you want to try a command without editing the configuration files or restarting the daemon.

**Prerequisites**

- Make sure the `libvirt-admin` package is installed.

**Procedure**

1. Optional: Back up the active set of log filters.

   ```
   # virt-admin -c virtqemud://system daemon-log-filters >> virt-filters-backup
   ```

   **NOTE**

   It is recommended that you back up the active set of filters so that you can restore them after generating the logs. If you do not restore the filters, the messages will continue to be logged which may affect system performance.

2. Use the `virt-admin` utility to enable debugging and set the filters according to your requirements.
Table 21.2. Debugging filter values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>logs all messages generated by libvirt.</td>
</tr>
<tr>
<td>2</td>
<td>logs all non-debugging information.</td>
</tr>
<tr>
<td>3</td>
<td>logs all warning and error messages. This is the default value.</td>
</tr>
<tr>
<td>4</td>
<td>logs only error messages.</td>
</tr>
</tbody>
</table>

Example 21.2. Sample virt-admin setting for logging filters

The following command:

- Logs all error and warning messages from the `remote`, `util.json`, and `rpc` layers
- Logs only error messages from the `event` layer.

```
# virt-admin -c virtqemud:///system daemon-log-filters "3:remote 4:event 3:util.json 3:rpc"
```

3. Use the `virt-admin` utility to save the logs to a specific file or directory.
   For example, the following command saves the log output to the `libvirt.log` file in the `/var/log/libvirt/` directory.

```
# virt-admin -c virtqemud:///system daemon-log-outputs "1:file:/var/log/libvirt/libvirt.log"
```

4. **Optional**: You can also remove the filters to generate a log file that contains all VM-related information. However, it is not recommended since this file may contain a large amount of redundant information produced by libvirt’s modules.

   - Use the `virt-admin` utility to specify an empty set of filters.

```
# virt-admin -c virtqemud:///system daemon-log-filters
```

5. **Optional**: Restore the filters to their original state using the backup file. Perform the second step with the saved values to restore the filters.

21.1.4. Attaching libvirt debug logs to support requests

You may have to request additional support to diagnose and resolve virtual machine (VM) problems. Attaching the debug logs to the support request is highly recommended to ensure that the support team has access to all the information they need to provide a quick resolution of the VM-related problem.

**Procedure**

- To report a problem and request support, open a support case.
Based on the encountered problems, attach the following logs along with your report:

- For problems with the libvirt service, attach the `/var/log/libvirt/libvirt.log` file from the host.
- For problems with a specific VM, attach its respective log file. For example, for the `testguest1` VM, attach the `testguest1.log` file, which can be found at `/var/log/libvirt/qemu/testguest1.log`.

Additional resources

- How to provide log files to Red Hat Support?

### 21.2. DUMPING A VIRTUAL MACHINE CORE

To analyze why a virtual machine (VM) crashed or malfunctioned, you can dump the VM core to a file on disk for later analysis and diagnostics.

This section provides a brief introduction to core dumping and explains how you can dump a VM core to a specific file.

#### 21.2.1. How virtual machine core dumping works

A virtual machine (VM) requires numerous running processes to work accurately and efficiently. In some cases, a running VM may terminate unexpectedly or malfunction while you are using it. Restarting the VM may cause the data to be reset or lost, which makes it difficult to diagnose the exact problem that caused the VM to crash.

In such cases, you can use the `virsh dump` utility to save (or `dump`) the core of a VM to a file before you reboot the VM. The core dump file contains a raw physical memory image of the VM which contains detailed information about the VM. This information can be used to diagnose VM problems, either manually, or by using a tool such as the `crash` utility.

Additional resources

- The `crash` man page
- The `crash` Github repository

#### 21.2.2. Creating a virtual machine core dump file

A virtual machine (VM) core dump contains detailed information about the state of a VM at any given time. This information, which is similar to a snapshot of the VM, can help you detect problems if a VM malfunctions or shuts down suddenly.

**Prerequisites**

- Make sure you have sufficient disk space to save the file. Note that the space occupied by the VM depends on the amount of RAM allocated to the VM.

**Procedure**

- Use the `virsh dump` utility.
  For example, the following command dumps the `lander1` VM’s cores, its memory and the CPU common register file to `gargantua.file` in the `/core/file` directory.
# virsh dump lander1 /core/file/gargantua.file --memory-only
Domain 'lander1' dumped to /core/file/gargantua.file

**IMPORTANT**

The `crash` utility no longer supports the default file format of the `virsh dump` command. To analyze a core dump file using `crash`, you must create the file using the `--memory-only` option.

Additionally, you must use the `--memory-only` option when creating a core dump file to attach to a Red Hat Support Case.

**Troubleshooting**

If the `virsh dump` command fails with a `System is deadlocked on memory` error, ensure you are assigning sufficient memory for the core dump file. To do so, use the following `crashkernel` option value. Alternatively, do not use `crashkernel` at all, which assigns core dump memory automatically.

```
crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M
```

**Additional resources**

- The `virsh dump --help` command
- The `virsh` man page
- Opening a Support Case

## 21.3. BACKTRACING VIRTUAL MACHINE PROCESSES

When a process related to a virtual machine (VM) malfunctions, you can use the `gstack` command along with the process identifier (PID) to generate an execution stack trace of the malfunctioning process. If the process is a part of a thread group then all the threads are traced as well.

**Prerequisites**

- Ensure that the `GDB` package is installed. For details about installing `GDB` and the available components, see [Installing the GNU Debugger](#).

- Make sure you know the PID of the processes that you want to backtrace. You can find the PID by using the `pgrep` command followed by the name of the process. For example:

  ```
  # pgrep libvirt
  22014
  22025
  ```

**Procedure**

- Use the `gstack` utility followed by the PID of the process you wish to backtrace. For example, the following command backtraces the libvirt process with the PID 22014.
Additional resources

- The `gstack` man page
- GNU Debugger (GDB)

Additional resources for reporting virtual machine problems and providing logs

To request additional help and support, you can:

- Raise a service request using the `redhat-support-tool` command line option, the Red Hat Portal UI, or several different methods using FTP.
  - To report problems and request support, see Open a Support Case.

- Upload the SOS Report and the log files when you submit a service request. This ensures that the Red Hat support engineer has all the necessary diagnostic information for reference.
  - For more information about SOS reports, see What is an SOS Report and how to create one in Red Hat Enterprise Linux?
  - For information about attaching log files, see How to provide files to Red Hat Support?
CHAPTER 22. FEATURE SUPPORT AND LIMITATIONS IN RHEL 9 VIRTUALIZATION

This document provides information on feature support and restrictions in Red Hat Enterprise Linux 9 (RHEL 9) virtualization.

22.1. HOW RHEL VIRTUALIZATION SUPPORT WORKS

A set of support limitations applies to virtualization in Red Hat Enterprise Linux 9 (RHEL 9). This means that when you use certain features or exceed a certain amount of allocated resources when using virtual machines in RHEL 9, Red Hat will not support these guests unless you have a specific subscription plan.

Features listed in Recommended features in RHEL 9 virtualization have been tested and certified by Red Hat to work with the KVM hypervisor on a RHEL 9 system. Therefore, they are fully supported and recommended for use in virtualization in RHEL 9.

Features listed in Unsupported features in RHEL 9 virtualization may work, but are not supported and not intended for use in RHEL 9. Therefore, Red Hat strongly recommends not using these features in RHEL 9 with KVM.

Resource allocation limits in RHEL 9 virtualization lists the maximum amount of specific resources supported on a KVM guest in RHEL 9. Guests that exceed these limits are not supported by Red Hat.

In addition, unless stated otherwise, all features and solutions used by the documentation for RHEL 9 virtualization are supported. However, some of these have not been completely tested and therefore may not be fully optimized.

IMPORTANT

Many of these limitations do not apply to other virtualization solutions provided by Red Hat, such as OpenShift Virtualization or Red Hat OpenStack Platform (RHOSP).

22.2. RECOMMENDED FEATURES IN RHEL 9 VIRTUALIZATION

The following features are recommended for use with the KVM hypervisor included with Red Hat Enterprise Linux 9 (RHEL 9):

Host system architectures

RHEL 9 with KVM is only supported on the following host architectures:

- AMD64 and Intel 64
- IBM Z - IBM z13 systems and later

Any other hardware architectures are not supported for using RHEL 9 as a KVM virtualization host, and Red Hat highly discourages doing so. Notably, this includes the 64-bit ARM architecture (ARM 64), which is only provided as Technology Preview.

Guest operating systems

Red Hat supports KVM virtual machines that use the following operating systems (OSs):

- Red Hat Enterprise Linux 7 and later
Microsoft Windows 10 and later
Microsoft Windows Server 2016 and later

Note, however, that by default, your guest OS does not use the same subscription as your host. Therefore, you must activate a separate licence or subscription for the guest OS to work properly.

Machine types

To ensure that your VM is compatible with your host architecture and that the guest OS runs optimally, the VM must use an appropriate machine type.

When creating a VM using the command line, the `virt-install` utility provides multiple methods of setting the machine type.

- When you use the `--os-variant` option, `virt-install` automatically selects the machine type recommended for your host CPU and supported by the guest OS.
- If you do not use `--os-variant` or require a different machine type, use the `--machine` option to specify the machine type explicitly.
- If you specify a `--machine` value that is unsupported or not compatible with your host, `virt-install` fails and displays an error message.

The recommended machine types for KVM virtual machines on supported architectures, and the corresponding values for the `--machine` option, are as follows. Y stands for the latest minor version of RHEL 9.

- On Intel 64 and AMD64 (x86_64): `pc-q35-rhel9.Y.0 → --machine=q35`
- On IBM Z (s390x): `s390-ccw-virtio-rhel9.Y.0 → --machine=s390-ccw-virtio`

To obtain the machine type of an existing VM:

```
# virsh dumpxml VM-name | grep machine=
```

To view the full list of machine types supported on your host:

```
# /usr/libexec/qemu-kvm -M help
```

Additional resources

- Unsupported features in RHEL 9 virtualization
- Resource allocation limits in RHEL 9 virtualization

### 22.3. UNSUPPORTED FEATURES IN RHEL 9 VIRTUALIZATION

The following features are not supported by the KVM hypervisor included with Red Hat Enterprise Linux 9 (RHEL 9):

IMPORTANT

Many of these limitations may not apply to other virtualization solutions provided by Red Hat, such as OpenShift Virtualization or Red Hat OpenStack Platform (RHOSP).

Features supported by other virtualization solutions are described as such in the following paragraphs.

Host system architectures

RHEL 9 with KVM is not supported on any host architectures that are not listed in Recommended features in RHEL 9 virtualization.

Notably, the 64-bit ARM architecture (ARM 64) is provided only as a Technology Preview for KVM virtualization on RHEL 9, and Red Hat therefore discourages its use in production environments.

Guest operating systems

KVM virtual machines (VMs) using the following guest operating systems (OSs) on a RHEL 9 host are not supported:

- Microsoft Windows 8.1 and earlier
- Microsoft Windows Server 2012 and earlier
- macOS
- Solaris for x86 systems
- Any OS released prior to 2009

For a list of guest OSs supported on RHEL hosts, see Certified guest operating systems for Red Hat Enterprise Linux with KVM.

Other solutions:

- For a list of guest OSs supported by other virtualization solutions provided by Red Hat, see Certified Guest Operating Systems in Red Hat OpenStack Platform, Red Hat Virtualization and OpenShift Virtualization.

Creating VMs in containers

Red Hat does not support creating KVM virtual machines in any type of container that includes the elements of the RHEL 9 hypervisor (such as the QEMU emulator or the libvirt package).

Other solutions:

- To create VMs in containers, Red Hat recommends using the OpenShift Virtualization offering.

Undocumented virsh commands and options

Any virsh commands and options that are not explicitly recommended by Red Hat documentation may not work correctly, and Red Hat recommends not using them in your production environment.

The QEMU command line

QEMU is an essential component of the virtualization architecture in RHEL 9, but it is difficult to manage manually, and improper QEMU configurations may cause security vulnerabilities. Therefore, using qemu-* command-line utilities, such as qemu-kvm is not supported by Red Hat.
Instead, use libvirt utilities, such as `virsh`, `virt-install`, and `virt-xml`, as these orchestrate QEMU according to the best practices.

**vCPU hot unplug**
Removing a virtual CPU (vCPU) from a running VM, also referred to as a vCPU hot unplug, is not supported in RHEL 9.

**Memory hot unplug**
Removing a memory device attached to a running VM, also referred to as a memory hot unplug, is unsupported in RHEL 9.

**QEMU-side I/O throttling**
Using the `virsh blkdeviotune` utility to configure maximum input and output levels for operations on virtual disk, also known as QEMU-side I/O throttling, is not supported in RHEL 9.

To set up I/O throttling in RHEL 9, use `virsh blkiotune`. This is also known as libvirt-side I/O throttling. For instructions, see *Disk I/O throttling in virtual machines*.

Other solutions:
- QEMU-side I/O throttling is also supported in RHOSP. For details, see *Setting Resource Limitation on Disk* and the Use Quality-of-Service Specifications section in the RHOSP Storage Guide.
- In addition, OpenShift Virtualization supports QEMU-side I/O throttling as well.

**Storage live migration**
Migrating a disk image of a running VM between hosts is not supported in RHEL 9.

Other solutions:
- Storage live migration is also supported in RHOSP, but with some limitations. For details, see *Migrate a Volume*.
- It is also possible live-migrate VM storage when using OpenShift Virtualization. For more information, see Virtual machine live migration.

**Live snapshots**
Creating or loading a snapshot of a running VM, also referred to as a live snapshot, is not supported in RHEL 9.

In addition, note that non-live VM snapshots are deprecated in RHEL 9. Therefore, creating or loading a snapshot of a shut-down VM is supported, but Red Hat recommends not using it.

Other solutions:
- RHOSP also supports live snapshots. For details, see *Importing virtual machines into the overcloud*.

**vHost Data Path Acceleration**
On RHEL 9 hosts, it is possible to configure vHost Data Path Acceleration (vDPA) for virtio devices, but Red Hat currently does not support this feature, and strongly discourages its use in production environments.
vhost-user

RHEL 9 does not support the implementation of a user-space vHost interface.

Other solutions:

- **vhost-user** is supported in RHOSP, but only for **virtio-net** interfaces. For details, see [virtio-net implementation](#) and [vhost user ports](#).

- OpenShift Virtualization supports **vhost-user** as well.

S3 and S4 system power states

Suspending a VM to the **Suspend to RAM** (S3) or **Suspend to disk** (S4) system power states is not supported. Note that these features are disabled by default, and enabling them will make your VM not supportable by Red Hat.

Note that the S3 and S4 states are also currently not supported in any other virtualization solution provided by Red Hat.

S3-PR on a multipathed vDisk

SCSI3 persistent reservation (S3-PR) on a multipathed vDisk is not supported in RHEL 9. As a consequence, Windows Cluster is not supported in RHEL 9.

virtio-crypto

The drivers for the **virtio-crypto** device are available in the RHEL 9.0 kernel, and the device can thus be enabled on a KVM hypervisor under certain circumstances. However, using the **virtio-crypto** device in RHEL 9 is not supported and its use is therefore highly discouraged.

Note that **virtio-crypto** devices are also not supported in any other virtualization solution provided by Red Hat.

Incremental live backup

Configuring a VM backup that only saves VM changes since the last backup, also known as incremental live backup, is not supported in RHEL 9, and Red Hat highly discourages its use.

net_failover

Using the **net_failover** driver to set up an automated network device failover mechanism is not supported in RHEL 9.

Note that **net_failover** is also currently not supported in any other virtualization solution provided by Red Hat.

Multi-FD migration

Migrating VMs using multiple file descriptors (FDs), also known as multi-FD migration, is not supported in RHEL 9.

Note that multi-FD migrations are also currently not supported in any other virtualization solution provided by Red Hat.

NVMe devices

Attaching Non-volatile Memory express (NVMe) devices to VMs hosted in RHEL 9 is not supported.
Note that attaching NVMe devices to VMs is also currently not supported in any other virtualization solution provided by Red Hat.

**TCG**

QEMU and libvirt include a dynamic translation mode using the QEMU Tiny Code Generator (TCG). This mode does not require hardware virtualization support. However, TCG is not supported by Red Hat.

TCG-based guests can be recognized by examining its XML configuration, for example using the `virsh dumpxml` command.

- The configuration file of a TCG guest contains the following line:
  ```xml
  <domain type='qemu'>
  ```

- The configuration file of a KVM guest contains the following line:
  ```xml
  <domain type='kvm'>
  ```

**Additional resources**

- Recommended features in RHEL 9 virtualization
- Resource allocation limits in RHEL 9 virtualization

### 22.4. RESOURCE ALLOCATION LIMITS IN RHEL 9 VIRTUALIZATION

The following limits apply to virtualized resources that can be allocated to a single KVM virtual machine (VM) on a Red Hat Enterprise Linux 9 (RHEL 9) host.

#### IMPORTANT

Many of these limitations do not apply to other virtualization solutions provided by Red Hat, such as OpenShift Virtualization or Red Hat OpenStack Platform (RHOSP).

**Maximum vCPUs per VM**

RHEL 9 supports up to 384 vCPUs allocated to a single VM.

**PCI devices per VM**

RHEL 9 supports 32 PCI device slots per VM bus, and 8 PCI functions per device slot. This gives a theoretical maximum of 256 PCI functions per bus when multi-function capabilities are enabled in the VM, and no PCI bridges are used.

Each PCI bridge adds a new bus, potentially enabling another 256 device addresses. However, some buses do not make all 256 device addresses available for the user; for example, the root bus has several built-in devices occupying slots.

**Virtualized IDE devices**

KVM is limited to a maximum of 4 virtualized IDE devices per VM.

### 22.5. HOW VIRTUALIZATION ON IBM Z DIFFERS FROM AMD64 AND INTEL 64
KVM virtualization in RHEL 9 on IBM Z systems differs from KVM on AMD64 and Intel 64 systems in the following:

**PCI and USB devices**

Virtual PCI and USB devices are not supported on IBM Z. This also means that `virtio-*-pci` devices are unsupported, and `virtio-*-ccw` devices should be used instead. For example, use `virtio-net-ccw` instead of `virtio-net-pci`.

Note that direct attachment of PCI devices, also known as PCI passthrough, is supported.

**Supported guest OS**

Red Hat only supports VMs hosted on IBM Z if they use RHEL 7, 8, or 9 as their guest operating system.

**Device boot order**

IBM Z does not support the `<boot dev='device'>` XML configuration element. To define device boot order, use the `<boot order='number'>` element in the `<devices>` section of the XML. For example:

```xml
<disk type='file' device='disk'>
  <driver name='qemu' type='qcow2'/>
  <source file='/path/to/qcow2'/>
  <target dev='vda' bus='virtio'/>
  <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0000'/>  
  <boot order='2'>
</disk>
```

**NOTE**

Using `<boot order='number'>` for boot order management is also preferred on AMD64 and Intel 64 hosts.

**Memory hot plug**

Adding memory to a running VM is not possible on IBM Z. Note that removing memory from a running VM (`memory hot unplug`) is also not possible on IBM Z, as well as on AMD64 and Intel 64.

**NUMA topology**

Non-Uniform Memory Access (NUMA) topology for CPUs is not supported by `libvirt` on IBM Z. Therefore, tuning vCPU performance using NUMA is not possible on these systems.

**vfio-ap**

VMs on an IBM Z host can use the `vfio-ap` cryptographic device passthrough, which is not supported on any other architectures.

**SMBIOS**

SMBIOS configuration is not available on IBM Z.

**Watchdog devices**

If using watchdog devices in your VM on an IBM Z host, use the `diag288` model. For example:

```xml
<devices>
  <watchdog model='diag288' action='poweroff'/>
</devices>
```

**kvm-clock**
The `kvm-clock` service is specific to AMD64 and Intel 64 systems, and does not have to be configured for VM time management on IBM Z.

v2v and p2v

The `virt-v2v` and `virt-p2v` utilities are supported only on the AMD64 and Intel 64 architecture, and are not provided on IBM Z.

Migrations

To successfully migrate to a later host model (for example from IBM z14 to z15), or to update the hypervisor, use the `host-model` CPU mode. The `host-passthrough` and `maximum` CPU modes are not recommended, as they are generally not migration-safe.

If you want to specify an explicit CPU model in the `custom` CPU mode, follow these guidelines:

- Do not use CPU models that end with `-base`.
- Do not use the `qemu`, `max` or `host` CPU model.

To successfully migrate to an older host model (such as from z15 to z14), or to an earlier version of QEMU, KVM, or the RHEL kernel, use the CPU type of the oldest available host model without `-base` at the end.

- If you have both the source host and the destination host running, you can instead use the `virsh cpu-baseline` command on the destination host to obtain a suitable CPU model.

Additional resources

- An overview of virtualization features support across architectures

### 22.6. HOW VIRTUALIZATION ON ARM 64 DIFFERS FROM AMD64 AND INTEL 64

KVM virtualization in RHEL 9 on ARM 64 systems is different from KVM on AMD64 and Intel 64 systems in a number of aspects. These include, but are not limited to, the following:

**Support**

Virtualization on ARM 64 is only provided as a Technology Preview on RHEL 9, and is therefore unsupported.

**Guest operating systems**

The only guest operating system currently working on ARM 64 virtual machines (VMs) is RHEL 9.

**Web console management**

Some features of VM management in the RHEL 9 web console may not work correctly on ARM 64 hardware.

**vCPU hot plug and hot unplug**

Attaching a virtual CPU (vCPU) to a running VM, also referred to as a vCPU hot plug, is not supported on ARM 64 hosts. In addition, like on AMD64 and Intel 64 hosts, removing a vCPU from a running VM (vCPU hot unplug), is not supported on ARM 64.

**SecureBoot**

The SecureBoot feature is not available on ARM 64 systems.

**PXE**

Booting in the Preboot Execution Environment (PXE) is only possible with the `virtio-net-pci` network interface controller (NIC). In addition, the built-in `VirtioNetDxe` driver of the virtual machine UEFI
platform firmware (installed with the **edk2-aarch64** package) needs to be used for PXE booting. Note that iPXE option ROMs are not supported.

**Device memory**

Device memory features, such as the dual in-line memory module (DIMM) and non-volatile DIMM (NVDIMM), do not work on ARM 64.

**pvpanic**

The pvpanic device is currently not functional on ARM 64. Make sure to remove the `<panic>` element from the `<devices>` section of the guest XML configuration on ARM 64, as its presence can lead to the VM failing to boot.

**OVMF**

VMs on an ARM 64 host cannot use the OVMF UEFI firmware used on AMD64 and Intel 64, included in the **edk2-ovmf** package. Instead, these VMs use UEFI firmware included in the **edk2-aarch64** package, which provides a similar interface and implements a similar set of features. Specifically, **edk2-aarch64** provides a built-in UEFI shell, but does not support the following functionality:

- SecureBoot
- Management Mode
- TPM-1.2 support

**kvm-clock**

The **kvm-clock** service does not have to be configured for time management in VMs on ARM 64.

**Peripheral devices**

ARM 64 systems do not support all the peripheral devices that are supported on AMD64 and Intel 64 systems. In some cases, the device functionality is not supported at all, and in other cases, a different device is supported for the same functionality.

**Serial console configuration**

When setting up a serial console on a VM, use the `console=ttyAMA0` parameter instead of `console=ttyS0` in the `/etc/default/grub` file.

**Non-maskable interrupts**

Sending non-maskable interrupts (NMIs) to an ARM 64 VM is currently not possible.

**Nested virtualization**

Creating nested VMs is currently not possible on ARM 64 hosts.

**v2v and p2v**

The **virt-v2v** and **virt-p2v** utilities are only supported on the AMD64 and Intel 64 architecture and are, therefore, not provided on ARM 64.

### 22.7. AN OVERVIEW OF VIRTUALIZATION FEATURES SUPPORT IN RHEL 9

The following tables provide comparative information about the support state of selected virtualization features in RHEL 9 across the available system architectures.

**Table 22.1. General support**

<table>
<thead>
<tr>
<th>System Architecture</th>
<th>SecureBoot Support</th>
<th>Management Mode Support</th>
<th>TPM-1.2 Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD64/Intel 64</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ARM 64</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 22.2. Device hot plug and hot unplug

<table>
<thead>
<tr>
<th></th>
<th>Intel 64 and AMD64</th>
<th>IBM Z</th>
<th>ARM 64</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU hot plug</td>
<td>Supported</td>
<td>Supported</td>
<td>UNAVAILABLE</td>
</tr>
<tr>
<td>CPU hot unplug</td>
<td>UNSUPPORTED</td>
<td>UNSUPPORTED</td>
<td>UNAVAILABLE</td>
</tr>
<tr>
<td>Memory hot plug</td>
<td>Supported</td>
<td>UNSUPPORTED</td>
<td>UNAVAILABLE</td>
</tr>
<tr>
<td>Memory hot unplug</td>
<td>UNSUPPORTED</td>
<td>UNSUPPORTED</td>
<td>UNAVAILABLE</td>
</tr>
<tr>
<td>Peripheral device hot plug</td>
<td>Supported</td>
<td>Supported [a]</td>
<td>Available but UNSUPPORTED</td>
</tr>
<tr>
<td>Peripheral device hot unplug</td>
<td>Supported</td>
<td>Supported [b]</td>
<td>Available but UNSUPPORTED</td>
</tr>
</tbody>
</table>

[a] Requires using virtio-*ccw devices instead of virtio-*pci

[b] Requires using virtio-*ccw devices instead of virtio-*pci

Table 22.3. Other selected features

<table>
<thead>
<tr>
<th></th>
<th>Intel 64 and AMD64</th>
<th>IBM Z</th>
<th>ARM 64</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMA tuning</td>
<td>Supported</td>
<td>UNSUPPORTED</td>
<td>UNAVAILABLE</td>
</tr>
<tr>
<td>SR-IOV devices</td>
<td>Supported</td>
<td>UNSUPPORTED</td>
<td>UNAVAILABLE</td>
</tr>
<tr>
<td>virt-v2v and p2v</td>
<td>Supported</td>
<td>UNSUPPORTED</td>
<td>UNAVAILABLE</td>
</tr>
</tbody>
</table>

Note that some of the unsupported features are supported on other Red Hat products, such as Red Hat Virtualization and Red Hat OpenStack platform. For more information, see Unsupported features in RHEL 9 virtualization.

Additional sources

- Unsupported features in RHEL 9 virtualization