Red Hat Enterprise Linux 8

Configuring and managing virtualization

Setting up your host, creating and administering virtual machines, and understanding virtualization features in Red Hat Enterprise Linux 8
Setting up your host, creating and administering virtual machines, and understanding virtualization features in Red Hat Enterprise Linux 8
Abstract

This document describes how to manage virtualization on Red Hat Enterprise Linux 8. In addition to general information about virtualization, it describes how to manage virtualization with command-line tools, as well as with the RHEL 8 web console.
# Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION .................................................. 11

CHAPTER 1. VIRTUALIZATION IN RHEL 8 - AN OVERVIEW ........................................ 12
  1.1. WHAT IS VIRTUALIZATION IN RHEL 8? ......................................................... 12
  1.2. ADVANTAGES OF VIRTUALIZATION ............................................................. 12
  1.3. RHEL 8 VIRTUAL MACHINE COMPONENTS AND THEIR INTERACTION .......................... 13
      Hypervisor ........................................................................................................ 13
      XML configuration ............................................................................................ 13
      Component interaction ......................................................................................... 13
  1.4. TOOLS AND INTERFACES FOR VIRTUALIZATION MANAGEMENT IN RHEL 8 ................. 14
      Command-line interface ...................................................................................... 14
      Graphical interfaces ............................................................................................ 15
  1.5. RED HAT VIRTUALIZATION SOLUTIONS ...................................................... 15

CHAPTER 2. GETTING STARTED WITH VIRTUALIZATION IN RHEL 8 ............................. 17
  2.1. ENABLING VIRTUALIZATION IN RHEL 8 ....................................................... 17
      Prerequisites ....................................................................................................... 17
      Procedure ............................................................................................................ 17
      Additional information ....................................................................................... 18
  2.2. CREATING VIRTUAL MACHINES .................................................................. 18
      2.2.1. Prerequisites .......................................................................................... 18
      2.2.2. Creating virtual machines using the command-line interface ....................... 19
      Prerequisites ....................................................................................................... 19
      Procedure ............................................................................................................ 19
      2.2.3. Creating virtual machines using the RHEL 8 web console ................................ 20
      Prerequisites ....................................................................................................... 21
      Procedure ............................................................................................................ 21
      Additional resources ......................................................................................... 22
  2.3. STARTING VIRTUAL MACHINES .................................................................. 22
      2.3.1. Prerequisites .......................................................................................... 22
      2.3.2. Starting a virtual machine using the command-line interface ....................... 22
      2.3.3. Powering up virtual machines in the RHEL 8 web console .......................... 22
      Prerequisites ....................................................................................................... 23
      Procedure ............................................................................................................ 23
      Additional resources ......................................................................................... 23
  2.4. CONNECTING TO VIRTUAL MACHINES ...................................................... 23
      2.4.1. Prerequisites .......................................................................................... 23
      2.4.2. Viewing the virtual machine graphical console in the RHEL 8 web console ........ 23
      Prerequisites ....................................................................................................... 23
      Procedure ............................................................................................................ 24
      Additional Resources ........................................................................................... 24
      2.4.3. Opening a virtual machine graphical console using Virt Viewer ...................... 25
      Prerequisites ....................................................................................................... 25
      Procedure ............................................................................................................ 25
      Additional resources ........................................................................................... 26
      2.4.4. Connecting to a virtual machine using SSH ................................................. 26
      Prerequisites ....................................................................................................... 26
      Procedure ............................................................................................................ 27
      2.4.5. Opening a virtual machine serial console ................................................... 27
      Prerequisites ....................................................................................................... 28
      Procedure ............................................................................................................ 28
5.5.4. Sending non-maskable interrupts to VMs using the RHEL 8 web console

Prerequisites
Procedure
Additional resources

5.6. VIEWING VIRTUAL MACHINE INFORMATION USING THE RHEL 8 WEB CONSOLE

5.6.1. Viewing a virtualization overview in the RHEL 8 web console

Prerequisites
Procedure
Additional resources

5.6.2. Viewing storage pool information using the RHEL 8 web console

Prerequisites
Procedure
Additional resources

5.6.3. Viewing basic virtual machine information in the RHEL 8 web console

Prerequisites
Procedure
Additional resources

5.6.4. Viewing virtual machine resource usage in the RHEL 8 web console

Prerequisites
Procedure
Additional resources

5.6.5. Viewing virtual machine disk information in the RHEL 8 web console

Prerequisites
Procedure
Additional resources

5.6.6. Viewing virtual NIC information in the RHEL 8 web console

Prerequisites
Procedure
Additional resources

5.7. MANAGING VIRTUAL CPUS USING THE RHEL 8 WEB CONSOLE

Prerequisites
Procedure

5.8. MANAGING VIRTUAL MACHINE DISKS USING THE RHEL 8 WEB CONSOLE

5.8.1. Viewing virtual machine disk information in the RHEL 8 web console

Prerequisites
Procedure
Additional resources

5.8.2. Adding new disks to virtual machines using the RHEL 8 web console

Prerequisites
Procedure
Additional resources

5.8.3. Attaching existing disks to virtual machines using the RHEL 8 web console

Prerequisites
Procedure
Additional resources

5.8.4. Detaching disks from virtual machines

Prerequisites
Procedure
Additional resources

5.9. USING THE RHEL 8 WEB CONSOLE FOR MANAGING VIRTUAL MACHINE VNICS
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.9.1. Viewing virtual NIC information in the RHEL 8 web console</td>
<td>68</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>68</td>
</tr>
<tr>
<td>Procedure</td>
<td>68</td>
</tr>
<tr>
<td>Additional resources</td>
<td>69</td>
</tr>
<tr>
<td>5.9.2. Connecting virtual NICs in the RHEL 8 web console</td>
<td>69</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>69</td>
</tr>
<tr>
<td>Procedure</td>
<td>70</td>
</tr>
<tr>
<td>5.9.3. Disconnecting virtual NICs in the RHEL 8 web console</td>
<td>70</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>70</td>
</tr>
<tr>
<td>Procedure</td>
<td>70</td>
</tr>
<tr>
<td>5.10. INTERACTING WITH VIRTUAL MACHINES USING THE RHEL 8 WEB CONSOLE</td>
<td>71</td>
</tr>
<tr>
<td>5.10.1. Viewing the virtual machine graphical console in the RHEL 8 web console</td>
<td>71</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>71</td>
</tr>
<tr>
<td>Procedure</td>
<td>71</td>
</tr>
<tr>
<td>Additional Resources</td>
<td>72</td>
</tr>
<tr>
<td>5.10.2. Viewing virtual machine consoles in remote viewers using the RHEL 8 web console</td>
<td>72</td>
</tr>
<tr>
<td>5.10.2.1. Viewing the graphical console in a remote viewer</td>
<td>72</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>73</td>
</tr>
<tr>
<td>Procedure</td>
<td>73</td>
</tr>
<tr>
<td>Additional Resources</td>
<td>75</td>
</tr>
<tr>
<td>5.10.2.2. Viewing the graphical console in a remote viewer connecting manually</td>
<td>75</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>75</td>
</tr>
<tr>
<td>Procedure</td>
<td>76</td>
</tr>
<tr>
<td>Additional Resources</td>
<td>76</td>
</tr>
<tr>
<td>5.10.3. Viewing the virtual machine serial console in the RHEL 8 web console</td>
<td>77</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>77</td>
</tr>
<tr>
<td>Procedure</td>
<td>77</td>
</tr>
<tr>
<td>Additional Resources</td>
<td>77</td>
</tr>
<tr>
<td>5.11. CREATING STORAGE POOLS USING THE RHEL 8 WEB CONSOLE</td>
<td>78</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>78</td>
</tr>
<tr>
<td>Procedure</td>
<td>78</td>
</tr>
<tr>
<td>Related information</td>
<td>79</td>
</tr>
<tr>
<td>5.12. DIFFERENCES BETWEEN VIRTUALIZATION FEATURES IN VIRTUAL MACHINE MANAGER AND THE RHEL 8 WEB CONSOLE</td>
<td>79</td>
</tr>
<tr>
<td>6.1. HOW VIRTUAL DEVICES WORK</td>
<td>81</td>
</tr>
<tr>
<td>The basics</td>
<td>81</td>
</tr>
<tr>
<td>Performance or flexibility</td>
<td>81</td>
</tr>
<tr>
<td>Additional resources</td>
<td>82</td>
</tr>
<tr>
<td>6.2. ATTACHING DEVICES TO VIRTUAL MACHINES</td>
<td>82</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>82</td>
</tr>
<tr>
<td>Procedure</td>
<td>82</td>
</tr>
<tr>
<td>Additional resources</td>
<td>83</td>
</tr>
<tr>
<td>6.3. MODIFYING DEVICES ATTACHED TO VIRTUAL MACHINES</td>
<td>83</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>83</td>
</tr>
<tr>
<td>Procedure</td>
<td>84</td>
</tr>
<tr>
<td>Additional resources</td>
<td>84</td>
</tr>
<tr>
<td>6.4. REMOVING DEVICES FROM VIRTUAL MACHINES</td>
<td>84</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>85</td>
</tr>
<tr>
<td>Procedure</td>
<td>85</td>
</tr>
<tr>
<td>Additional resources</td>
<td>85</td>
</tr>
<tr>
<td>6.5. TYPES OF VIRTUAL DEVICES</td>
<td>85</td>
</tr>
</tbody>
</table>
CHAPTER 7. MANAGING STORAGE FOR VIRTUAL MACHINES

7.1. UNDERSTANDING VIRTUAL MACHINE STORAGE

7.1.1. Virtual machine storage

7.1.2. Storage pools

    Storage pool storage types
    Storage pool actions
    Supported and unsupported storage pool types

7.1.3. Storage volumes

7.2. MANAGING STORAGE FOR VIRTUAL MACHINES USING THE CLI

7.2.1. Viewing virtual machine storage information using the CLI

    7.2.1.1. Viewing storage pool information using the CLI
     
        Procedure
        Additional resources

    7.2.1.2. Viewing storage volume information using the CLI
     
        Procedure

7.2.2. Creating and assigning storage for virtual machines using the CLI

    7.2.2.1. Creating and assigning directory-based storage for virtual machines using the CLI
     
        7.2.2.1.1. Creating directory-based storage pools using the CLI
         
            Procedure

        7.2.2.1.2. Directory-based storage pool parameters
         
            Parameters
            Example

    7.2.2.2. Creating and assigning disk-based storage for virtual machines using the CLI
     
        7.2.2.2.1. Creating disk-based storage pools using the CLI
         
            Recommendations
            Procedure

        7.2.2.2.2. Disk-based storage pool parameters
         
            Parameters
            Example

    7.2.2.3. Creating and assigning filesystem-based storage for virtual machines using the CLI
     
        7.2.2.3.1. Creating filesystem-based storage pools using the CLI
         
            Recommendations
            Procedure

        7.2.2.3.2. Filesystem-based storage pool parameters
         
            Parameters
            Example

    7.2.2.4. Creating and assigning GlusterFS storage for virtual machines using the CLI
     
        7.2.2.4.1. Creating GlusterFS-based storage pools using the CLI
         
            Prerequisites
            Procedure

        7.2.2.4.2. GlusterFS-based storage pool parameters
         
            Parameters
            Example

    7.2.2.5. Creating and assigning iSCSI-based storage for virtual machines using the CLI
     
        Recommendations
        Prerequisites

        7.2.2.5.1. Creating iSCSI-based storage pools using the CLI
         
            Procedure

        7.2.2.5.2. iSCSI-based storage pool parameters
         
            Parameters
            Example

        7.2.2.5.3. Securing iSCSI storage pools with libvirt secrets
         
            Procedure
7.2.2.6. Creating and assigning LVM-based storage for virtual machines using the CLI

7.2.2.6.1. Creating LVM-based storage pools using the CLI
  Recommendations
  Procedure

7.2.2.6.2. LVM-based storage pool parameters
  Parameters
  Example

7.2.2.7. Creating and assigning network-based storage for virtual machines using the CLI

7.2.2.7.1. Prerequisites

7.2.2.7.2. Creating network-based storage pools using the CLI
  Procedure

7.2.2.7.3. NFS-based storage pool parameters
  Parameters
  Example

7.2.2.8. Creating and assigning vHBA-based storage for virtual machines using the CLI

7.2.2.8.1. Recommendations

7.2.2.8.2. Prerequisites

7.2.2.8.3. Creating vHBAs
  Procedure

7.2.2.8.4. Creating vHBA-based storage pools using the CLI
  Prerequisites
  Procedure

7.2.2.8.5. vHBA-based storage pool parameters
  Parameters
  Examples

7.2.2.8.6. Assigning and connecting SCSI LUN-based storage volumes to virtual machines using the CLI

7.2.2.8.6.1. Prerequisites

7.2.2.8.6.2. Creating vHBA-based storage pools using the CLI
  Prerequisites
  Procedure

7.2.2.9. Creating and assigning storage volumes using the CLI

7.2.2.9.1. Prerequisites

7.2.2.9.2. Procedure

7.2.3. Deleting storage for virtual machines using the CLI

7.2.3.1. Deleting storage pools using the CLI
  Prerequisites
  Procedure

7.2.3.2. Deleting storage volumes using the CLI
  Prerequisites
  Procedure
CHAPTER 9. OPTIMIZING VIRTUAL MACHINE PERFORMANCE IN RHEL 8 ........................................ 133
  9.1. WHAT INFLUENCES VIRTUAL MACHINE PERFORMANCE ........................................... 133
  9.2. OPTIMIZING VIRTUAL MACHINE PERFORMANCE USING TUNED .................. 133
    Prerequisites .................................................................................................................. 134
    Procedure ...................................................................................................................... 134
    Additional resources ................................................................................................. 134
  9.3. OPTIMIZING VIRTUAL MACHINE I/O PERFORMANCE ......................................... 134
    9.3.1. Tuning block I/O in virtual machines ............................................................. 135
    Procedure ...................................................................................................................... 135
    9.3.2. Disk I/O throttling in virtual machines ......................................................... 135
    Procedure ...................................................................................................................... 135
    Additional information ............................................................................................. 136
    9.3.3. Enabling multi-queue virtio-scsi ............................................................... 136
    Procedure ...................................................................................................................... 136
  9.4. OPTIMIZING VIRTUAL MACHINE CPU PERFORMANCE .................................. 136
    9.4.1. Configuring NUMA in a virtual machine .................................................. 137
    Prerequisites .................................................................................................................. 137
    Procedure ...................................................................................................................... 137
    Automatic methods .................................................................................................... 137
    Manual methods ......................................................................................................... 137
    Additional resources ................................................................................................. 138
    9.4.2. Sample vCPU performance tuning scenario ............................................. 138
    Starting scenario ........................................................................................................ 138
    Solution ......................................................................................................................... 139
    Additional resources ................................................................................................. 142
  9.5. OPTIMIZING VIRTUAL MACHINE NETWORK PERFORMANCE .................... 139
    Procedure ...................................................................................................................... 143
    Additional resources ................................................................................................. 144
  9.6. VIRTUAL MACHINE PERFORMANCE MONITORING TOOLS ....................... 144
    Default OS performance monitoring tools ......................................................... 145
    perf kvm ..................................................................................................................... 145
    numastat ....................................................................................................................... 146

CHAPTER 10. SHARING FILES BETWEEN THE HOST AND ITS VIRTUAL MACHINES .......... 148
  10.1. SHARING FILES BETWEEN THE HOST AND LINUX VIRTUAL MACHINES ........ 148
    Prerequisites .................................................................................................................. 148
    Procedure ...................................................................................................................... 149
  10.2. SHARING FILES BETWEEN THE HOST AND WINDOWS VIRTUAL MACHINES .... 150
    Prerequisites .................................................................................................................. 150
    Procedure ...................................................................................................................... 152

CHAPTER 11. UNDERSTANDING VIRTUAL NETWORKING ................................................. 155
  11.1. VIRTUAL NETWORKING IN ROUTED MODE .................................................... 156
    Common topologies .................................................................................................... 156
    DMZ ............................................................................................................................... 157
    Virtual server hosting ............................................................................................... 157
  11.2. VIRTUAL NETWORKING IN BRIDGED MODE ..................................................... 157
    Common scenarios .................................................................................................... 158
    Additional resources ................................................................................................. 159
  11.3. VIRTUAL NETWORKING IN ISOLATED MODE .................................................... 159
  11.4. VIRTUAL NETWORKING NETWORK ADDRESS TRANSLATION ................ 159
  11.5. VIRTUAL NETWORKING IN OPEN MODE ....................................................... 160
  11.6. VIRTUAL NETWORKING DNS AND DHCP .................................................... 160
14.1. HOW RHEL 8 VIRTUALIZATION SUPPORT WORKS 187
14.2. RECOMMENDED FEATURES IN RHEL 8 VIRTUALIZATION 187
   Additional resources 188
14.3. UNSUPPORTED FEATURES IN RHEL 8 VIRTUALIZATION 188
   Additional resources 189
14.4. RESOURCE ALLOCATION LIMITS IN RHEL 8 VIRTUALIZATION 189
PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

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

- For simple comments on specific passages, make sure you are viewing the documentation in the Multi-page HTML format. Highlight the part of text that you want to comment on. Then, click the Add Feedback pop-up that appears below the highlighted text, and follow the displayed instructions.

- For submitting more complex feedback, create a Bugzilla 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. VIRTUALIZATION IN RHEL 8 - AN OVERVIEW

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 8: its basics, advantages, components, and other possible virtualization solutions provided by Red Hat.

1.1. WHAT IS VIRTUALIZATION IN RHEL 8?

Red Hat Enterprise Linux 8 (RHEL 8) provides the virtualization functionality. This means that with the help of virtualization software, a machine running RHEL 8 can 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 Section 1.2, “Advantages of virtualization”.

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

To try out virtualization in RHEL 8, see Chapter 2, Getting started with virtualization in RHEL 8.

NOTE

In addition to RHEL 8 virtualization, Red Hat offers a number of specialized virtualization solutions, each with a different user focus and features. For more information, see Section 1.5, “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 in software, 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**
  All of a VM’s configuration is saved as data on the host, and under software control. Therefore, a VM can easily be created, removed, cloned, migrated, operated remotely, or connected to remote storage.

  In addition, the current state of the VM can be backed up as a snapshot at any time. A snapshot can then be loaded to restore the system to the saved state.

- **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 that 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 6 guest OS, you can run applications released for RHEL 6 on a RHEL 8 host system.

**NOTE**
Not all operating systems are supported as a guest OS in a RHEL 8 host. For details, see Section 14.2, “Recommended features in RHEL 8 virtualization”.

## 1.3. RHEL 8 VIRTUAL MACHINE COMPONENTS AND THEIR INTERACTION

Virtualization in RHEL 8 consists of the following principal software components:

### Hypervisor
The basis of creating virtual machines (VMs) in RHEL 8 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) kernel module and virtualization kernel drivers, such as virtio and vfio. 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) describes a specific VM. It 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.

### 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 8 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 in RHEL 8*.

**Figure 1.1. RHEL 8 virtualization architecture**

---

1.4. TOOLS AND INTERFACES FOR VIRTUALIZATION MANAGEMENT IN RHEL 8

You can manage virtualization in RHEL 8 using the command-line interface (CLI) or several graphical user interface (GUIs).

**Command-line interface**

The CLI is the most powerful method of managing virtualization in RHEL 8. Prominent CLI commands for virtualization 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 virtual machine (VM) - `virsh start` and `virsh shutdown`
  - Listing available VMs - `virsh list`
  - Creating a VM from a configuration file - `virsh create`
• Entering a virtualization shell - \texttt{virsh}

For more information, see the \texttt{virsh(1)} man page.

• \texttt{virt-install} - A CLI utility for creating new VMs. For more information, see the \texttt{virt-install(1)} man page.

• \texttt{virt-xml} - A utility for editing the configuration of a VM.

• \texttt{guestfish} - A utility for examining and modifying VM disk images. For more information, see the \texttt{guestfish(1)} man page.

For instructions on basic virtualization management with CLI, see Chapter 2, \textit{Getting started with virtualization in RHEL 8}.

**Graphical interfaces**

You can use the following GUls to manage virtualization in RHEL 8:

• The \texttt{RHEL 8 web console}, also known as \textit{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 Chapter 5, \textit{Using the RHEL 8 web console for managing virtual machines}.

• The Virtual Machine Manager (\texttt{virt-manager}) application provides a specialized GUI for managing VMs and virtualization hosts.

  \textbf{IMPORTANT}

  Although still supported in RHEL 8, \texttt{virt-manager} has been deprecated. The RHEL 8 web console is intended to become its replacement in a subsequent release. It is, therefore, recommended that you get familiar with the web console for managing virtualization in a GUI.

  However, in RHEL 8, some features may only be accessible from either \texttt{virt-manager} or the command line. For details, see Section 5.12, “Differences between virtualization features in virtual machine manager and the RHEL 8 web console”.

• The \texttt{Gnome Boxes} application is a lightweight graphical interface to view and access VMs and remote systems. Gnome Boxes is primarily designed for use on desktop systems.

  \textbf{IMPORTANT}

  Gnome Boxes is provided as a part of the GNOME desktop environment and is supported on RHEL 8, but Red Hat recommends that you use the web console for managing virtualization in a GUI.

\section*{1.5. RED HAT VIRTUALIZATION SOLUTIONS}

The following Red Hat products are built on top of RHEL 8 virtualization features and expand the KVM virtualization capabilities available in RHEL 8. In addition, many limitations of RHEL 8 virtualization do not apply to these products:

Red Hat Virtualization (RHV)
RHV is designed for enterprise-class scalability and performance, and enables management of your entire virtual infrastructure, including hosts, virtual machines, networks, storage, and users from a centralized graphical interface. For information about the differences between virtualization in Red Hat Enterprise Linux and Red Hat Virtualization, see the Red Hat Customer Portal.

Red Hat Virtualization can be used by enterprises running large deployments or mission-critical applications. Examples of large deployments suited to Red Hat Virtualization include databases, trading platforms, and messaging systems that must run continuously without any downtime.

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

To download a fully supported 60-day evaluation version of Red Hat Virtualization, see https://access.redhat.com/products/red-hat-virtualization/evaluation

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.

To download a fully supported 60-day evaluation version of Red Hat OpenStack Platform, see https://access.redhat.com/products/red-hat-openstack-platform/evaluation

In addition, specific Red Hat products provide operating-system-level virtualization, also known as containerization:

- Containers are isolated instances of the host OS and operate on top of an existing OS kernel. For more information on containers, see the Red Hat Customer Portal.

- Containers do not have the versatility of KVM virtualization, but are more lightweight and flexible to handle. For a more detailed comparison, see the Introduction to Linux Containers.
CHAPTER 2. GETTING STARTED WITH VIRTUALIZATION IN RHEL 8

To start using virtualization in RHEL 8, follow the steps below. The default method for this is the command-line interface (CLI), but for user convenience, some of the steps can be completed in the web console GUI.

1. Enable the virtualization module and install the virtualization packages - see Section 2.1, “Enabling virtualization in RHEL 8”.

2. Create a virtual machine (VM):
   - For CLI, see Section 2.2.2, “Creating virtual machines using the command-line interface”.
   - For GUI, see Section 2.2.3, “Creating virtual machines using the RHEL 8 web console”.

3. Start the VM:
   - For CLI, see Section 2.3.2, “Starting a virtual machine using the command-line interface”.
   - For GUI, see Section 2.3.3, “Powering up virtual machines in the RHEL 8 web console”.

4. Connect to the VM:
   - For CLI, see ] or xref:proc_opening-a-virtual-machine-graphical-console-using-virt-viewer_assembly_connecting-to-virtual-machines[].
   - For GUI, see Section 2.4.2, “Viewing the virtual machine graphical console in the RHEL 8 web console”.

**NOTE**

The web console currently provides only a subset of VM management functions, so using the command line is recommended for advanced use of virtualization in RHEL 8.

### 2.1. ENABLING VIRTUALIZATION IN RHEL 8

To use virtualization in RHEL 8, you must enable the virtualization module, install virtualization packages, and ensure your system is configured to host virtual machines.

**Prerequisites**

- Red Hat Enterprise Linux 8 must be installed and registered on your host machine.
- Your system must meet 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 packages in the virtualization module:
1. Install the packages in the virtualization module:

```bash
# yum module install virt
```

2. Install the `virt-install` package:

```bash
# yum install virt-install
```

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

```bash
# 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)
```

4. 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.

**Additional information**

- Note that if virtualization is not supported by your host CPU, `virt-host-validate` generates the following output:

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

  However, attempting to create virtual machines on such a host system will fail, rather than have performance problems.

**2.2. CREATING VIRTUAL MACHINES**

To create a virtual machine (VM) in RHEL 8, use the **command line interface** or the **the RHEL 8 web console**.

**2.2.1. Prerequisites**

- Virtualization must be installed and enabled on your system.

- Prior to creating VMs on your system, consider the amount of system resources you need 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.
2.2.2. Creating virtual machines using the command-line interface

To create a virtual machine (VM) on your RHEL 8 host using the `virt-install` utility, follow the instructions below.

**Prerequisites**

- An operating system (OS) installation source, which can be one of the following, and can be available locally or on a network:
  - An ISO image of an installation medium
  - A disk image of an existing virtual machine installation
- Optionally, a Kickstart file can also 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. In its arguments, specify at minimum:

- The name of the new machine
- The amount of allocated memory
- The number of allocated virtual CPUs (vCPUs)
- The type and size of allocated storage
- The type and location of the OS installation source

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, 2 vCPUs, and a 8 GiB qcow2 virtual disk is automatically configured for the VM.

  ```
  # virt-install --name demo-guest1 --memory 2048 --vcpus 2 --disk size=8 --os-variant win10 --cdrom /home/username/Downloads/Win10install.iso
  ```

- The following creates a VM named `demo-guest2` that uses the `/home/username/Downloads/rhel8.iso` image to run a RHEL 8 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 rhel8.0 --cdrom /home/username/Downloads/rhel8.iso
```

The following creates a RHEL 8 VM named demo-guest3 connects to an existing disk image, /home/username/backup/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 --os-variant rhel8.0 --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 16 GiB qcow2 virtual disk.

```
# virt-install --name demo-guest4 --memory 2048 --vcpus 2 --os-variant rhel8.0 --location http://example.com/OS-install --initrd-inject /home/username/ks.cfg --extra-args="ks=file:/ks.cfg console=tty0 console=ttyS0,115200n8"
```

The following creates a VM named demo-guest5 that installs from a RHEL8.iso 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=16 --os-variant rhel8.0 --location RHEL8.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 rhel8.0 --location RHEL8.iso --graphics none --extra-args='console=ttyS0'
```

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

**NOTE**

A number of other options can be specified for virt-install to further configure the VM and its OS installation. For details, see the virt-install man page.

### 2.2.3. Creating virtual machines using the RHEL 8 web console
To create a virtual machine (VM) on the host machine to which the web console is connected, follow the instructions below.

**Prerequisites**

- To be able to use the RHEL 8 web console to manage VMs, you must **install the web console VM plug-in**.
- Before creating VMs, consider the amount of system resources you need 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.
- A locally available operating system (OS) installation source, which can be one of the following:
  - An ISO image of an installation medium
  - A disk image of an existing VM installation

**Procedure**

1. Click **Create VM** in the Virtual Machines interface of the RHEL 8 web console. The Create New Virtual Machine dialog appears.

2. Enter the basic configuration of the VM you want to create.
   
   - **Connection** - The connection to the host to be used by the VM.
   - **Name** - The name of the VM.
   - **Installation Source Type** - The type of the installation source: Filesystem, URL

<table>
<thead>
<tr>
<th>Create New Virtual Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection</strong></td>
</tr>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td><strong>Installation Source Type</strong></td>
</tr>
<tr>
<td><strong>Installation Source</strong></td>
</tr>
<tr>
<td><strong>OS Vendor</strong></td>
</tr>
<tr>
<td><strong>Operating System</strong></td>
</tr>
<tr>
<td><strong>Memory</strong></td>
</tr>
<tr>
<td><strong>Storage Size</strong></td>
</tr>
<tr>
<td><strong>Immediately Start VM</strong></td>
</tr>
</tbody>
</table>
• **Installation Source** - The path or URL that points to the installation source.

• **OS Vendor** - The vendor of the VM’s operating system.

• **Operating System** - The VM’s operating system.

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

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

• **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.

You must install the operating system the first time the VM is run.

**Additional resources**

- For information on installing an operating system on a VM, see Section 5.3.2, “Installing operating systems using the RHEL 8 web console”.

2.3. STARTING VIRTUAL MACHINES

To start a virtual machine (VM) in RHEL 8, you can use the **command line interface** or the **web console GUI**.

2.3.1. Prerequisites

- Before a VM can be started, it must be created and ideally also installed with an OS. For instructions on how to do so, see Section 2.2, “Creating virtual machines”.

2.3.2. Starting a virtual machine using the command-line interface

- To start a local VM using the command-line interface, use the **virsh start** command:

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

- If the VM is on a remote host, use QEMU+SSH connection to the host. For example, the following 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:
  Last login: Mon Feb 18 07:28:55 2019
  Domain demo-guest1 started
  ```

  Note that managing VMs on remote host can be simplified by modifying your **libvirt and SSH configuration**.

2.3.3. Powering up virtual machines in the RHEL 8 web console
If a virtual machine (VM) is in the **shut off** state, you can start it using the RHEL 8 web console.

**Prerequisites**
To be able to use the RHEL 8 web console to manage VMs, you must **install the web console VM plug-in**.

**Procedure**

1. Click a row with the name of the VM you want to start.
   The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click **Run**.
   The VM starts.

**Additional resources**

- For information on shutting down a VM, see Section 5.5.2, "Powering down virtual machines in the RHEL 8 web console".

- For information on restarting a VM, see Section 5.5.3, "Restarting virtual machines using the RHEL 8 web console".

- For information on sending a non-maskable interrupt to a VM, see Section 5.5.4, "Sending non-maskable interrupts to VMs using the RHEL 8 web console".

### 2.4. CONNECTING TO VIRTUAL MACHINES

To interact with a virtual machine (VM) in RHEL 8, you need to connect to it by doing one of the following:

- When using the RHEL 8 web console interface, use the Virtual Machines pane in the web console interface. For more information, see Section 2.4.2, "Viewing the virtual machine graphical console in the RHEL 8 web console"

- If you need to interact with a VM graphical display without using the RHEL 8 web console, use the Virt Viewer application. For details, see Section 2.4.3, "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 also configure your system for **more convenient access to remote hosts**.

#### 2.4.1. Prerequisites

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

#### 2.4.2. Viewing the virtual machine graphical console in the RHEL 8 web console

You can view the graphical console of a selected virtual machine (VM) in the RHEL 8 web console. The VM console shows the graphical output of the VM.

**Prerequisites**
To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

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

Procedure

1. Click a row with the name of the VM whose graphical console you want to view. The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click Consoles. The graphical console appears in the web interface.

![Virtual Machines](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.

**NOTE**

The server on which the RHEL 8 web console is running can intercept specific key combinations, such as Ctrl+Alt+F1, 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+F1 combination to the VM, click the Send key menu and select the Ctrl+Alt+F1 menu entry.

Additional Resources

- For details on viewing the graphical console in a remote viewer, see Section 5.10.2, “Viewing virtual machine consoles in remote viewers using the RHEL 8 web console”.
For details on viewing the serial console in the RHEL 8 web console, see Section 5.10.3, “Viewing the virtual machine serial console in the RHEL 8 web console”.

2.4.3. 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 virtual machine 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:

  ```bash
  # 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.

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

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

Additional resources

- For more information on using Virt Viewer, see the **virt-viewer** man page.

- Connecting to VMs on a remote host can be simplified by modifying your libvirt and SSH configuration.

- For management of virtual machines in an interactive GUI in RHEL 8, you can use the web console interface. For more information, see Section 5.10, "Interacting with virtual machines using the RHEL 8 web console".

2.4.4. 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**

- Network connection and root access privileges to the target VM.

- The **libvirt-nss** component must be installed and enabled on the VM’s host. If it is not, do the following:
a. Install the `libvirt-nss` package:

```
# yum 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  
[...]
```

- If the target VM is located on a remote host, connection and root access privileges to the host are also needed.

Procedure

1. **Optional**: 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 2018  
root~#
```

2. Use the VM’s name and user access credentials to connect to it. For example, the following connects to 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~]#
```

**NOTE**

If you do not know the virtual machine’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
```

2.4.5. Opening a virtual 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 or SPICE protocols, and thus does not offer video display for GUI tools.
Does not have network connection, and thus cannot be interacted with using SSH.

Prerequisites

- 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` virtual machine, 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

- For more information about the VM serial console, see the virsh man page.
2.4.6. 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:
Last login: Mon Feb 18 07:28:55 2019

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:
Last login: Mon Feb 18 07:28:55 2019

Id   Name              State
---------------------------------
1    remote-guest      running
```

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`:
As a result, you can manage remote VMs by using libvirt-based utilities on the local system and adding `--connect qemu-host-alias` to the commands. This automatically performs the commands over SSH on the remote host.

For example, 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:
Last login: Mon Feb 18 07:28:55 2019

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

4. **[Optional]** 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:
Last login: Mon Feb 18 07:28:55 2019

  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.

**Additional resources**

- When connecting to a remote host, it is also possible to avoid having to provide the root password to the remote system. To do so, do one or more of the following:
  - Set up key-based SSH access to the remote host.
  - Use SSH connection multiplexing to connect to the remote system.
  - Set up a kerberos authentication ticket on the remote system.
Utilities that can use the -c (or --connect) option and the remote host access configuration described above include:

- virt-install
- virt-viewer
- virsh
- virt-manager

2.5. SHUTTING DOWN VIRTUAL MACHINES

To shut down a running virtual machine in Red Hat Enterprise Linux 8, use the command line interface or the web console GUI.

2.5.1. 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:
    ```bash
    # 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:
    ```bash
    # virsh -c qemu+ssh://root@10.0.0.1/system shutdown demo-guest1
    root@10.0.0.1's password:
    Last login: Mon Feb 18 07:28:55 2019
    Domain demo-guest1 is being shutdown
    ```

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

```bash
# 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 destroys the running VM instance. However, in rare cases, this command may cause corruption of the VM’s file system, so using `virsh destroy` is only recommended if all other shutdown methods have failed.

2.5.2. Powering down virtual machines in the RHEL 8 web console

If a virtual machine (VM) is in the running state, you can shut it down using the RHEL 8 web console.
Prerequisites
To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

Procedure

1. Click a row with the name of the VM you want to shut down. The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click **Shut Down**.
   The VM shuts down.

   **NOTE**
   If the VM does not shut down, click the arrow next to the **Shut Down** button and select **Force Shut Down**.

Additional resources

- For information on starting a VM, see Section 5.5.1, "Powering up virtual machines in the RHEL 8 web console".

- For information on restarting a VM, see Section 5.5.3, "Restarting virtual machines using the RHEL 8 web console".

- For information on sending a non-maskable interrupt to a VM, see Section 5.5.4, "Sending non-maskable interrupts to VMs using the RHEL 8 web console".

2.6. RELATED INFORMATION

- The information above apply to the AMD64 and Intel 64 architectures. If you want to use RHEL8 virtualization on other supported architectures, different setup procedures are needed and certain features may be restricted or work differently. For details, see the appropriate section below:
  - Chapter 4, *Getting started with virtualization in RHEL 8 on IBM Z*
  - Chapter 3, *Getting started with virtualization in RHEL 8 on IBM POWER*
CHAPTER 3. GETTING STARTED WITH VIRTUALIZATION IN RHEL 8 ON IBM POWER

When using RHEL 8 on IBM POWER8 or POWER9 hardware, it is possible to use KVM virtualization. However, when enabling the KVM hypervisor on your system, extra steps are needed compared to virtualization on AMD64 and Intel 64 architectures. Certain RHEL 8 virtualization features also have different or restricted functionality on IBM POWER.

Apart from the information in the following sections, using virtualization on IBM POWER works the same as on AMD64 and Intel 64. Therefore, you can see other RHEL 8 virtualization documentation for more information about using virtualization on IBM POWER.

3.1. ENABLING VIRTUALIZATION ON IBM POWER

To set up a KVM hypervisor and be able to create virtual machines (VMs) on an IBM POWER8 or IBM POWER9 system running RHEL 8, follow the instructions below.

Prerequisites

- RHEL 8 is installed and registered on your host machine.
- 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.
- Your CPU machine type must support IBM POWER virtualization.
  To verify this, query the platform information in your `/proc/cpuinfo` file.

```
# grep ^platform /proc/cpuinfo/
platform : PowerNV
```

If the output of this command includes the `PowerNV` entry, you are running a PowerNV machine type and can use virtualization on IBM POWER.

Procedure

1. Load the KVM-HV kernel module

```
# modprobe kvm_hv
```

2. Verify that the KVM kernel module is loaded

```
# Ismod | grep kvm
```

If KVM loaded successfully, the output of this command includes `kvm_hv`.

3. Install the packages in the virtualization module:

```
# yum module install virt
```

4. Install the `virt-install` package:
# yum install virt-install

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

```bash
# 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 : PASS
```

6. 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.

Additional information

- Note that if virtualization is not supported by your host CPU, `virt-host-validate` generates the following output:

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

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

### 3.2. HOW VIRTUALIZATION ON IBM POWER DIFFERS FROM AMD64 AND INTEL 64

KVM virtualization in RHEL 8 on IBM POWER systems is different from KVM on AMD64 and Intel 64 systems in a number of aspects, notably:

**Memory requirements**

VMs on IBM POWER consume more memory. Therefore, the recommended minimum memory allocation for a virtual machine (VM) on an IBM POWER host is 2GB RAM.

**Display protocols**

The SPICE protocol is not supported on IBM POWER systems. To display the graphical output of a VM, use the **VNC** protocol. In addition, only the following virtual graphics card devices are supported:

- **vga** - only supported in `-vga std` mode and not in `-vga cirrus` mode.
- **virtio-vga**
- **virtio-gpu**

**SMBIOS**
SMBIOS configuration is not available

Memory allocation errors

POWER8 VMs, including compatibility mode VMs, may fail with an error similar to:

qemu-kvm: Failed to allocate KVM HPT of order 33 (try smaller maxmem?): Cannot allocate memory

This is significantly more likely to occur on VMs that use RHEL 7.3 and prior as the guest OS.

To fix the problem, increase the CMA memory pool available for the guest’s hashed page table (HPT) by adding `kvm_cma_resv_ratio=memory` to the host’s kernel command line, where `memory` is the percentage of the host memory that should be reserved for the CMA pool (defaults to 5).

Huge pages

Transparent huge pages (THPs) do not provide any notable performance benefits on IBM POWER8 VMs. However, IBM POWER9 VMs can benefit from THPs as expected.

In addition, the size of static huge pages on IBM POWER8 systems are 16 MiB and 16 GiB, as opposed to 2 MiB and 1 GiB on AMD64, Intel 64, and IBM POWER9. As a consequence, to migrate a VM configured with static huge pages from an IBM POWER8 host to an IBM POWER9 host, you must first set up 1GiB huge pages on the VM.

kvm-clock

The `kvm-clock` service does not have to be configured for time management in VMs on IBM POWER9.

pvpanic

IBM POWER9 systems do not support the `pvpanic` device. However, an equivalent functionality is available and activated by default on this architecture. To enable it in a VM, use the `<on_crash>` XML configuration element with the `preserve` value.

In addition, make sure to remove the `<panic>` element from the `<devices>` section, as its presence can lead to the VM failing to boot on IBM POWER systems.

Single-threaded host

On IBM POWER8 systems, the host machine must run in `single-threaded mode` to support VMs. This is automatically configured if the `qemu-kvm` packages are installed. However, VMs running on single-threaded hosts can still use multiple threads.

Peripheral devices

A number of peripheral devices supported on AMD64 and Intel 64 systems are not supported on IBM POWER systems, or a different device is supported as a replacement.

- Devices used for PCI-E hierarchy, including `ioh3420` and `xio3130-downstream`, are not supported. This functionality is replaced by multiple independent PCI root bridges provided by the `spapr-pci-host-bridge` device.

- UHCI and EHCI PCI controllers are not supported. Use OHCI and XHCI controllers instead.

- IDE devices, including the virtual IDE CD-ROM (`ide-cd`) and the virtual IDE disk (`ide-hd`), are not supported. Use the `virtio-scsi` and `virtio-blk` devices instead.

- Emulated PCI NICs (`rtl8139`) are not supported. Use the `virtio-net` device instead.

- Sound devices, including `intel-hda`, `hda-output`, and `AC97`, are not supported.
USB redirection devices, including `usb-redir` and `usb-tablet`, are not supported.

v2v and p2v

The `virt-v2v` and `virt-p2v` utilities are only supported on the AMD64 and Intel 64 architecture. Because of this, they are not provided on IBM POWER.
CHAPTER 4. GETTING STARTED WITH VIRTUALIZATION IN RHEL 8 ON IBM Z

When using RHEL 8 on IBM Z hardware, it is possible to use KVM virtualization. However, when enabling the KVM hypervisor on your system, extra steps are needed compared to virtualization on AMD64 and Intel 64 architectures. Certain RHEL 8 virtualization features also have different or restricted functionality on IBM Z.

Apart from the information in the following sections, using virtualization on IBM Z works the same as on AMD64 and Intel 64. Therefore, you can see other RHEL 8 virtualization documentation for more information about using virtualization on IBM Z.

4.1. ENABLING VIRTUALIZATION ON IBM Z

To set up a KVM hypervisor and be able to create virtual machines (VMs) on an IBM Z system running RHEL 8, follow the instructions below.

Prerequisites

- RHEL 8 is installed and registered on your host machine.
- 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.
- Your IBM Z host system needs to use a z13 CPU or later.
- RHEL 8 has to be installed on a logical partition (LPAR). In addition, the LPAR must support the start-interpretive execution (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. Load the KVM kernel module:

```
# modprobe kvm
```

2. Verify that the KVM kernel module is loaded:

```
# lsmod | grep kvm
```

If KVM loaded successfully, the output of this command includes kvm:

3. Install the packages in the virtualization module:

```
# yum module install virt
```

4. Install the virt-install package:
# yum install virt-install

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

```
# 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
[...]
```

6. 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.

**Additional information**

- Note that if 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, attempting to create VMs on such a host system will fail, rather than have performance problems.

**4.2. HOW VIRTUALIZATION ON IBM Z DIFFERS FROM AMD64 AND INTEL 64**

KVM virtualization in RHEL 8 on IBM Z systems differs from KVM on AMD64 and Intel 64 systems in the following:

**No graphical output**

   Displaying the VM graphical output is not possible when connecting to the VM using the VNC protocol. This is due to the `gnome-desktop` utility not being supported on IBM Z.

**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.

**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:

```
<devices>
  <disk type='file' snapshot='external'>
NOTE

Using `<boot order='number'>` for boot order management is preferred also on AMD64 and Intel 64 hosts.

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 only supported on the AMD64 and Intel 64 architecture. Because of this, they are not provided on IBM Z.

4.3. RELATED INFORMATION

- When setting up a VM on an IBM Z system, it is recommended to protect the guest OS from the "Spectre" vulnerability. To do so, use the virsh edit command to modify the VM's XML configuration and configure its CPU in one of the following ways:
  - Use the host CPU model, for example as follows:
    ```xml
    <cpu mode='host-model' check='partial'>
    <model fallback='allow'/>
    </cpu>
    ```
    This makes the ppa15 and bpb features available to the guest if the host supports them.
• If using a specific host model, add the **ppa15** and **pbp** features. The following example uses the zEC12 CPU model:

```xml
<cpu mode='custom' match='exact' check='partial'>
  <model fallback='allow'>zEC12</model>
  <feature policy='force' name='ppa15'/>
  <feature policy='force' name='pbp'/>
</cpu>
```

Note that when using the **ppa15** feature with the **z114** and **z196** CPU models on a host machine that uses a z12 CPU, you also need to use the latest microcode level (bundle 95 or later).

• Note that running KVM on the z/VM OS is not supported.
CHAPTER 5. USING THE RHEL 8 WEB CONSOLE FOR MANAGING VIRTUAL MACHINES

To manage virtual machines in a graphical interface, you can use the Virtual Machines pane in the RHEL 8 web console.

The following sections describe the web console’s virtualization management capabilities and provide instructions for using them.

5.1. OVERVIEW OF VIRTUAL MACHINE MANAGEMENT USING THE RHEL 8 WEB CONSOLE

The RHEL 8 web console is a web-based interface for system administration. With the installation of a web console plug-in, the web console can be used to manage virtual machines (VMs) on the servers to which the web console can connect. It provides a graphical view of VMs on a host system to which the web console can connect, and allows monitoring system resources and adjusting configuration with ease.

Using the RHEL 8 web console for VM management, you can do the following:

- Create and delete VMs
- Install operating systems on VMs
- Run and shut down VMs
- View information about VMs
- Create and attach disks to VMs
- Configure virtual CPU settings for VMs
• Manage virtual network interfaces
• Interact with VMs using VM consoles

NOTE

The Virtual Machine Manager (virt-manager) application is still supported in RHEL 8 but has been deprecated. The RHEL 8 web console is intended to become its replacement in a subsequent release. It is, therefore, recommended that you get familiar with the web console for managing virtualization in a GUI.

However, in RHEL 8, some features may only be accessible from either virt-manager or the command line. For details, see Section 5.12, “Differences between virtualization features in virtual machine manager and the RHEL 8 web console”.

For more information on the Virtual Machine Manager, see RHEL 7 documentation.

5.2. SETTING UP THE RHEL 8 WEB CONSOLE TO MANAGE VIRTUAL MACHINES

Before using the RHEL 8 web console to manage VMs, you must install the web console virtual machine plug-in.

Prerequisites

• Ensure that the web console is installed on your machine.

```sh
$ yum info cockpit
Installed Packages
Name : cockpit
[...]
```

If the web console is not installed, see the Managing systems using the RHEL 8 web console guide for more information about installing the web console.

Procedure

• Install the cockpit-machines plug-in.

```sh
# yum install cockpit-machines
```

If the installation is successful, Virtual Machines appears in the web console side menu.
5.3. CREATING VIRTUAL MACHINES AND INSTALLING GUEST OPERATING SYSTEMS USING THE RHEL 8 WEB CONSOLE

The following sections provide information on how to use the RHEL 8 web console to create virtual machines (VMs) and install operating systems on VMs.

5.3.1. Creating virtual machines using the RHEL 8 web console

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

Prerequisites

- To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.
- Before creating VMs, consider the amount of system resources you need 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.
- A locally available operating system (OS) installation source, which can be one of the following:
  - An ISO image of an installation medium
  - A disk image of an existing VM installation

Procedure

1. Click Create VM in the Virtual Machines interface of the RHEL 8 web console. The Create New Virtual Machine dialog appears.
2. Enter the basic configuration of the VM you want to create.

- **Connection** - The connection to the host to be used by the VM.
- **Name** - The name of the VM.
- **Installation Source Type** - The type of the installation source: Filesystem, URL
- **Installation Source** - The path or URL that points to the installation source.
- **OS Vendor** - The vendor of the VM's operating system.
- **Operating System** - The VM's operating system.
- **Memory** - The amount of memory with which to configure the VM.
- **Storage Size** - The amount of storage space with which to configure the VM.
- **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.

You must install the operating system the first time the VM is run.

**Additional resources**

- For information on installing an operating system on a VM, see **Section 5.3.2, “Installing operating systems using the RHEL 8 web console”**.
5.3.2. Installing operating systems using the RHEL 8 web console
The first time a virtual machine loads, you must install an operating system on the virtual machine.

**Prerequisites**
- Before using the RHEL 8 web console to manage virtual machines, you must install the web console virtual machine plug-in.
- A VM on which to install an operating system.

**Procedure**
- Click **Install**.
  The installation routine of the operating system runs in the virtual machine console.

**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 virtual machine is created.

**NOTE**
If the installation routine fails, the virtual machine must be deleted and recreated.

5.4. DELETING VIRTUAL MACHINES USING THE RHEL 8 WEB CONSOLE
You can delete a virtual machine and its associated storage files from the host to which the RHEL 8 web console is connected.

**Prerequisites**
To be able to use the RHEL 8 web console to manage virtual machines, you must install the web console virtual machine plug-in.

**Procedure**
1. In the Virtual Machines interface of the RHEL 8 web console, click the name of the VM you want to delete.
   The row expands to reveal the Overview pane with basic information about the selected virtual machine and controls for shutting down and deleting the virtual machine.
2. Click **Delete**. A confirmation dialog appears.

3. [Optional] To delete all or some of the storage files associated with the virtual machine, select the checkboxes next to the storage files you want to delete.

4. Click **Delete**. The virtual machine and any selected associated storage files are deleted.

### 5.5. POWERING UP AND POWERING DOWN VIRTUAL MACHINES USING THE RHEL 8 WEB CONSOLE

Using the RHEL 8 web console, you can run, shut down, and restart virtual machines. You can also send a non-maskable interrupt to a virtual machine that is unresponsive.

#### 5.5.1. Powering up virtual machines in the RHEL 8 web console

If a virtual machine (VM) is in the **shut off** state, you can start it using the RHEL 8 web console.

**Prerequisites**

To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

**Procedure**

1. Click a row with the name of the VM you want to start.
The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click **Run**.
   The VM starts.

**Additional resources**

- For information on shutting down a VM, see Section 5.5.2, “Powering down virtual machines in the RHEL 8 web console”.
- For information on restarting a VM, see Section 5.5.3, “Restarting virtual machines using the RHEL 8 web console”.
- For information on sending a non-maskable interrupt to a VM, see Section 5.5.4, “Sending non-maskable interrupts to VMs using the RHEL 8 web console”.

5.5.2. Powering down virtual machines in the RHEL 8 web console

If a virtual machine (VM) is in the **running** state, you can shut it down using the RHEL 8 web console.

**Prerequisites**
To be able to use the RHEL 8 web console to manage VMs, you must [install the web console VM plug-in](#).

**Procedure**

1. Click a row with the name of the VM you want to shut down.
   The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click **Shut Down**.
   The VM shuts down.

**NOTE**

If the VM does not shut down, click the arrow next to the **Shut Down** button and select **Force Shut Down**.

**Additional resources**

- For information on starting a VM, see Section 5.5.1, "Powering up virtual machines in the RHEL 8 web console".
- For information on restarting a VM, see Section 5.5.3, "Restarting virtual machines using the RHEL 8 web console".
- For information on sending a non-maskable interrupt to a VM, see Section 5.5.4, "Sending non-maskable interrupts to VMs using the RHEL 8 web console".

5.5.3. Restarting virtual machines using the RHEL 8 web console

If a virtual machine (VM) is in the **running** state, you can restart it using the RHEL 8 web console.

**Prerequisites**
To be able to use the RHEL 8 web console to manage VMs, you must [install the web console VM plug-in](#).
Procedure

1. Click a row with the name of the VM you want to restart. The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click Restart. The VM shuts down and restarts.

   **NOTE**
   If the VM does not restart, click the arrow next to the Restart button and select Force Restart.

Additional resources

- For information on starting a VM, see Section 5.5.1, "Powering up virtual machines in the RHEL 8 web console”.
- For information on shutting down a VM, see Section 5.5.2, “Powering down virtual machines in the RHEL 8 web console”.
- For information on sending a non-maskable interrupt to a VM, see Section 5.5.4, "Sending non-maskable interrupts to VMs using the RHEL 8 web console”.

5.5.4. Sending non-maskable interrupts to VMs using the RHEL 8 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 Ctrl+Alt+Del NMI to a VM that is not responsive.

Prerequisites

Before using the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

Procedure

1. Click a row with the name of the VM to which you want to send an NMI. The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click the arrow next to the Shut Down button and select Send Non-Maskable Interrupt. An NMI is sent to the VM.

Additional resources

- For information on starting a VM, see Section 5.5.1, "Powering up virtual machines in the RHEL 8 web console”.
- For information on restarting a VM, see Section 5.5.3, "Restarting virtual machines using the RHEL 8 web console”.
- For information on shutting down a VM, see Section 5.5.2, “Powering down virtual machines in the RHEL 8 web console”.

5.6. VIEWING VIRTUAL MACHINE INFORMATION USING THE RHEL 8 WEB CONSOLE
Using the RHEL 8 web console, you can view information about the virtual storage and VMs to which the web console is connected.

5.6.1. Viewing a virtualization overview in the RHEL 8 web console

The following describes how to view an overview of the available virtual storage and the virtual machines (VMs) to which the web console session is connected.

Prerequisites
To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

Procedure
To view information about the available storage and the VMs to which the web console is attached.

- Click Virtual Machines in the web console’s side menu.

Information about the available storage and the VMs to which the web console session is connected appears.

The information includes the following:

- Storage Pools - The number of storage pools that can be accessed by the web console and their state.
- Networks - The number of networks 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

- For information on viewing detailed information about the storage pools the web console session can access, see [Section 5.6.2, “Viewing storage pool information using the RHEL 8 web console”](#).

- For information on viewing basic information about a selected VM to which the web console session is connected, see [Section 5.6.3, “Viewing basic virtual machine information in the RHEL 8 web console”](#).

- For information on viewing resource usage for a selected VM to which the web console session is connected, see [Section 5.6.4, “Viewing virtual machine resource usage in the RHEL 8 web console”](#).

- For information on viewing disk information about a selected VM to which the web console session is connected, see [Section 5.6.5, “Viewing virtual machine disk information in the RHEL 8 web console”](#).

- For information on viewing virtual network interface card information about a selected VM to which the web console session is connected, see [Section 5.6.6, “Viewing virtual NIC information in the RHEL 8 web console”](#).

### 5.6.2. Viewing storage pool information using the RHEL 8 web console

The following describes how to view detailed storage pool information about the virtual machine (VM) storage pools that the web console session can access.

#### Prerequisites

To be able to use the RHEL 8 web console to manage VMs, you must [install the web console VM plug-in](#).

#### Procedure

To view storage pool information:

1. Click **Storage Pools** at the top of the Virtual Machines tab. 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 size of the storage pool.
- **Connection** - The connection used to access the storage pool.
- **State** - The state of the storage pool.

2. Click a row with the name of the storage whose information you want to see. The row expands to reveal the Overview pane with following information about the selected storage pool:

- **Path** - The path to the storage pool.
- **Persistent** - Whether or not the storage pool is persistent.
- **Autostart** - Whether or not the storage pool starts automatically.
- **Type** - The storage pool type.
To view a list of storage volumes created from the storage pool, click **Storage Volumes**. The Storage Volumes pane appears showing a list of configured storage volumes with their sizes and the amount of space used.

### Additional resources

- For information on viewing information about all of the VMs to which the web console session is connected, see Section 5.6.1, “Viewing a virtualization overview in the RHEL 8 web console”.

- For information on viewing basic information about a selected VM to which the web console session is connected, see Section 5.6.3, “Viewing basic virtual machine information in the RHEL 8 web console”.

- For information on viewing resource usage for a selected VM to which the web console session is connected, see Section 5.6.4, “Viewing virtual machine resource usage in the RHEL 8 web console”.

- For information on viewing disk information about a selected VM to which the web console session is connected, see Section 5.6.5, “Viewing virtual machine disk information in the RHEL 8 web console”.

- For information on viewing virtual network interface card information about a selected VM to which the web console session is connected, see Section 5.6.6, “Viewing virtual NIC information in the RHEL 8 web console”.

### 5.6.3. Viewing basic virtual machine information in the RHEL 8 web console

The following describes how to view basic information about a selected virtual machine (VM) to which the web console session is connected.

#### Prerequisites

To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

#### Procedure
To view basic information about a selected VM.

- Click a row with the name of the VM whose information you want to see. The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

**NOTE**

If another tab is selected, click **Overview**.

The information includes the following:

- **Memory** - The amount of memory assigned to the VM.
- **Emulated Machine** - The machine type emulated by the VM.
- **vCPUs** - The number of virtual CPUs configured for the VM.

**NOTE**

To see more detailed virtual CPU information and configure the virtual CPUs configured for a VM, see Section 5.7, “Managing virtual CPUs using the RHEL 8 web console”.

- **Boot Order** - The boot order configured for the VM.
- **CPU Type** - The architecture of the virtual CPUs configured for the VM.
- **Autostart** - Whether or not autostart is enabled for the VM.

**Additional resources**

- For information on viewing information about all of the VMs to which the web console session is connected, see Section 5.6.1, “Viewing a virtualization overview in the RHEL 8 web console”.
- For information on viewing information about the storage pools to which the web console session is connected, see Section 5.6.2, “Viewing storage pool information using the RHEL 8 web console”.
- For information on viewing resource usage for a selected VM to which the web console session is connected, see Section 5.6.4, “Viewing virtual machine resource usage in the RHEL 8 web console”.
- For information on viewing disk information about a selected VM to which the web console session is connected, see Section 5.6.5, “Viewing virtual machine disk information in the RHEL 8 web console”.
- For information on viewing virtual network interface card information about a selected VM to which the web console session is connected, see Section 5.6.6, “Viewing virtual NIC information in the RHEL 8 web console”.

### 5.6.4. Viewing virtual machine resource usage in the RHEL 8 web console

The following describes how to view resource usage information about a selected virtual machine (VM) to which the web console session is connected.

**Prerequisites**

To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

**Procedure**

To view information about the memory and virtual CPU usage of a selected VM.

1. Click a row with the name of the VM whose information you want to see. The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click **Usage**. The Usage pane appears with information about the memory and virtual CPU usage of the VM.
Additional resources

- For information on viewing information about all of the VMs to which the web console session is connected, see Section 5.6.1, “Viewing a virtualization overview in the RHEL 8 web console”.

- For information on viewing information about the storage pools to which the web console session is connected, see Section 5.6.2, “Viewing storage pool information using the RHEL 8 web console”.

- For information on viewing basic information about a selected VM to which the web console session is connected, see Section 5.6.3, “Viewing basic virtual machine information in the RHEL 8 web console”.

- For information on viewing disk information about a selected VM to which the web console session is connected, see Section 5.6.5, “Viewing virtual machine disk information in the RHEL 8 web console”.

- For information on viewing virtual network interface card information about a selected VM to which the web console session is connected, see Section 5.6.6, “Viewing virtual NIC information in the RHEL 8 web console”.

5.6.5. Viewing virtual machine disk information in the RHEL 8 web console

The following describes how to view disk information about a virtual machine (VM) to which the web console session is connected.

Prerequisites
To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.
Procedure
To view disk information about a selected VM.

1. Click a row with the name of the VM whose information you want to see.
   The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click Disks.
   The Disks pane appears with information about the disks assigned to the VM.

The information includes the following:

- **Device** - The device type of the disk.
- **Target** - The controller type of the disk.
- **Used** - The amount of the disk that is used.
- **Capacity** - The size of the disk.
- **Bus** - The bus type of the disk.
- **Readonly** - Whether or not the disk is read-only.
- **Source** - The disk device or file.

Additional resources
For information on viewing information about all of the VMs to which the web console session is connected, see Section 5.6.1, “Viewing a virtualization overview in the RHEL 8 web console”.

For information on viewing information about the storage pools to which the web console session is connected, see Section 5.6.2, “Viewing storage pool information using the RHEL 8 web console”.

For information on viewing basic information about a selected VM to which the web console session is connected, see Section 5.6.3, “Viewing basic virtual machine information in the RHEL 8 web console”.

For information on viewing resource usage for a selected VM to which the web console session is connected, see Section 5.6.4, “Viewing virtual machine resource usage in the RHEL 8 web console”.

For information on viewing virtual network interface card information about a selected VM to which the web console session is connected, see Section 5.6.6, “Viewing virtual NIC information in the RHEL 8 web console”.

### 5.6.6. Viewing virtual NIC information in the RHEL 8 web console

The following describes how to view information about the virtual network interface cards (vNICs) on a selected virtual machine (VM):

**Prerequisites**

To be able to use the RHEL 8 web console to manage VMs, you must **install the web console VM plug-in**.

**Procedure**

To view information about the virtual network interface cards (NICs) on a selected VM.

1. Click a row with the name of the VM whose information you want to see.
   The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click **Networks**.
   The Networks pane appears with information about the virtual NICs configured for the VM.
The information includes the following:

- **Type** - The type of network interface for the VM. Types include direct, network, bridge, ethernet, hostdev, mcast, user, and server.

- **Model type** - The model of the virtual NIC.

- **MAC Address** - The MAC address of the virtual NIC.

- **Source** - The source of the network interface. This is dependent on the network type.

- **State** - The state of the virtual NIC.

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

4. Change the Network Type and Model.
5. Click **Save**. The network interface is modified.

**NOTE**

When the VM is running, changes to the virtual network interface settings only take effect after the VM is stopped and restarted.

**Additional resources**

- For information on viewing information about all of the VMs to which the web console session is connected, see Section 5.6.1, “Viewing a virtualization overview in the RHEL 8 web console”.
- For information on viewing information about the storage pools to which the web console session is connected, see Section 5.6.2, “Viewing storage pool information using the RHEL 8 web console”.
- For information on viewing basic information about a selected VM to which the web console session is connected, see Section 5.6.3, “Viewing basic virtual machine information in the RHEL 8 web console”.
- For information on viewing resource usage for a selected VM to which the web console session is connected, see Section 5.6.4, “Viewing virtual machine resource usage in the RHEL 8 web console”.
- For information on viewing disk information about a selected VM to which the web console session is connected, see Section 5.6.5, “Viewing virtual machine disk information in the RHEL 8 web console”.

## 5.7. MANAGING VIRTUAL CPUS USING THE RHEL 8 WEB CONSOLE

Using the RHEL 8 web console, you can review and configure virtual CPUs used by virtual machines (VMs) to which the web console is connected.

**Prerequisites**

To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

**Procedure**

1. Click a row with the name of the VMs for which you want to view and configure virtual CPU parameters.
   The row expands to reveal the Overview pane with basic information about the selected VMs, including the number of virtual CPUs, and controls for shutting down and deleting the VM.

2. Click the number of vCPUs in the Overview pane.
   The vCPU Details dialog appears.
3. Configure the virtual CPUs for the selected VM.
   - **vCPU Count** - Enter the number of virtual CPUs for the VM.
   - **vCPU Maximum** - Enter the maximum number of virtual CPUs that can be configured for the VM.
   - **Sockets** - Select the number of sockets to expose to the VM.
   - **Cores per socket** - Select the number of cores for each socket to expose to the VM.
   - **Threads per core** - Select the number of threads for each core to expose to the VM.

4. Click **Apply**.
   The virtual CPUs for the VM are configured.

**NOTE**

When the VM is running, changes to the virtual CPU settings only take effect after the VM is stopped and restarted.

**5.8. MANAGING VIRTUAL MACHINE DISKS USING THE RHEL 8 WEB CONSOLE**

Using the RHEL 8 web console, you can manage the disks configured for the virtual machines to which the web console is connected.

You can:

- View information about disks.
● Create and attach new virtual disks to virtual machines.
● Attach existing virtual disks to virtual machines.
● Detach virtual disks from virtual machines.

5.8.1. Viewing virtual machine disk information in the RHEL 8 web console

The following describes how to view disk information about a virtual machine (VM) to which the web console session is connected.

Prerequisites
To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

Procedure
To view disk information about a selected VM.

1. Click a row with the name of the VM whose information you want to see. The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click Disks. The Disks pane appears with information about the disks assigned to the VM.

The information includes the following:

● Device - The device type of the disk.
- **Target** - The controller type of the disk.
- **Used** - The amount of the disk that is used.
- **Capacity** - The size of the disk.
- **Bus** - The bus type of the disk.
- **Readonly** - Whether or not the disk is read-only.
- **Source** - The disk device or file.

**Additional resources**

- For information on viewing information about all of the VMs to which the web console session is connected, see Section 5.6.1, “Viewing a virtualization overview in the RHEL 8 web console”.
- For information on viewing information about the storage pools to which the web console session is connected, see Section 5.6.2, “Viewing storage pool information using the RHEL 8 web console”.
- For information on viewing basic information about a selected VM to which the web console session is connected, see Section 5.6.3, “Viewing basic virtual machine information in the RHEL 8 web console”.
- For information on viewing resource usage for a selected VM to which the web console session is connected, see Section 5.6.4, “Viewing virtual machine resource usage in the RHEL 8 web console”.
- For information on viewing virtual network interface card information about a selected VM to which the web console session is connected, see Section 5.6.6, “Viewing virtual NIC information in the RHEL 8 web console”.

### 5.8.2. Adding new disks to virtual machines using the RHEL 8 web console

You can add new disks to virtual machines (VMs) by creating a new disk (storage pool) and attaching it to a VM using the RHEL 8 web console.

**NOTE**

You can only use directory-type storage pools when creating new disks for VMs using the RHEL 8 web console.

**Prerequisites**

To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

**Procedure**

1. Click a row with the name of the VM for which you want to create and attach a new disk.
   The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click **Disks**.
   The Disks pane appears with information about the disks configured for the VM.
3. Click **Add Disk**.
   The Add Disk dialog appears.

4. Ensure that the **Create New** option button is selected.

5. Configure the new disk.
   - **Pool** - Select the storage pool from which the virtual disk will be created.
   - **Target** - Select a target for the virtual disk that 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. Supported types: qcow2, raw

• **Persistence** - Whether or not the virtual disk will be persistent. If checked, the virtual disk is persistent. If not checked, the virtual disk is not persistent.

   NOTE

   Transient disks can only be added to VMs that are running.

6. Click **Add**.
   The virtual disk is created and connected to the VM.

**Additional resources**

• For information on viewing disk information about a selected VM to which the web console session is connected, see Section 5.8.1, “Viewing virtual machine disk information in the RHEL 8 web console”.

• For information on attaching existing disks to VMs, see Section 5.8.3, “Attaching existing disks to virtual machines using the RHEL 8 web console”.

• For information on detaching disks from VMs, see Section 5.8.4, “Detaching disks from virtual machines”.

5.8.3. Attaching existing disks to virtual machines using the RHEL 8 web console

The following describes how to attach existing disks to a virtual machine (VM) using the RHEL 8 web console.

   NOTE

   You can only attach directory-type storage pools to VMs using the RHEL 8 web console.

**Prerequisites**

To be able to use the RHEL 8 web console to manage VMs, you must **install the web console VM plug-in**.

**Procedure**

1. Click a row with the name of the VM to which you want to attach an existing disk.
   The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click **Disks**.
   The Disks pane appears with information about the disks configured for the VM.
3. Click **Add Disk**.
   The Add Disk dialog appears.

![Add Disk dialog](image)

4. Click the **Use Existing** option 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.
- **Target** - Select a target for the virtual disk that will be attached.
- **Volume** - Select the storage volume that will be attached.
- **Persistence** - Check to make the virtual disk persistent. Clear to make the virtual disk transient.

6. Click **Add**

   The selected virtual disk is attached to the VM.

**Additional resources**

- For information on viewing disk information about a selected VM to which the web console session is connected, see Section 5.8.1, “Viewing virtual machine disk information in the RHEL 8 web console”.

- For information on creating new disks and attaching them to VMs, see Section 5.8.2, “Adding new disks to virtual machines using the RHEL 8 web console”.

- For information on detaching disks from VMs, see Section 5.8.4, “Detaching disks from virtual machines”.

**5.8.4. Detaching disks from virtual machines**

The following describes how to detach disks from virtual machines (VMs) using the RHEL 8 web console.

**Prerequisites**

To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

**Procedure**

1. Click a row with the name of the VM from which you want to detach an existing disk.
   
   The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.
2. Click **Disks**.
The Disks pane appears with information about the disks configured for the VM.

3. Click **next to the disk you want to detach from the VM.**
The virtual disk is detached from the VM.

**CAUTION**

There is no confirmation before detaching the disk from the VM.

**Additional resources**

- For information on viewing disk information about a selected VM to which the web console session is connected, see Section 5.8.1, “Viewing virtual machine disk information in the RHEL 8 web console”.

- For information on creating new disks and attaching them to VMs, see Section 5.8.2, “Adding new disks to virtual machines using the RHEL 8 web console”.

- For information on attaching existing disks to VMs, see Section 5.8.3, “Attaching existing disks to virtual machines using the RHEL 8 web console”.

**5.9. USING THE RHEL 8 WEB CONSOLE FOR MANAGING VIRTUAL MACHINE VNICS**
Using the RHEL 8 web console, you can manage the virtual network interface cards (vNICs) configured for the virtual machines to which the web console is connected. You can view information about vNICs. You can also connect and disconnect vNICs from virtual machines.

5.9.1. Viewing virtual NIC information in the RHEL 8 web console

The following describes how to view information about the virtual network interface cards (vNICs) on a selected virtual machine (VM):

**Prerequisites**
To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

**Procedure**
To view information about the virtual network interface cards (NICs) on a selected VM.

1. Click a row with the name of the VM whose information you want to see. The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click **Networks**. The Networks pane appears with information about the virtual NICs configured for the VM.

![Networks pane](image)

The information includes the following:

- **Type** - The type of network interface for the VM. Types include direct, network, bridge, ethernet, hostdev, mcast, user, and server.

- **Model type** - The model of the virtual NIC.

- **MAC Address** - The MAC address of the virtual NIC.
• **Source** - The source of the network interface. This is dependent on the network type.

• **State** - The state of the virtual NIC.

3. **To edit the virtual network settings**, click **Edit**. The Virtual Network Interface Settings.

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Userspace SLIRP stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>virtio (Linux, perf)</td>
</tr>
<tr>
<td>Mac Address</td>
<td>52:54:00:5c:76:79</td>
</tr>
</tbody>
</table>

4. **Change the Network Type and Model.**

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

### NOTE

When the VM is running, changes to the virtual network interface settings only take effect after the VM is stopped and restarted.

**Additional resources**

- For information on viewing information about all of the VMs to which the web console session is connected, see Section 5.6.1, “Viewing a virtualization overview in the RHEL 8 web console”.
- For information on viewing information about the storage pools to which the web console session is connected, see Section 5.6.2, “Viewing storage pool information using the RHEL 8 web console”.
- For information on viewing basic information about a selected VM to which the web console session is connected, see Section 5.6.3, “Viewing basic virtual machine information in the RHEL 8 web console”.
- For information on viewing resource usage for a selected VM to which the web console session is connected, see Section 5.6.4, “Viewing virtual machine resource usage in the RHEL 8 web console”.
- For information on viewing disk information about a selected VM to which the web console session is connected, see Section 5.6.5, “Viewing virtual machine disk information in the RHEL 8 web console”.

### 5.9.2. Connecting virtual NICs in the RHEL 8 web console

Using the RHEL 8 web console, you can reconnect disconnected virtual network interface cards (NICs) configured for a selected virtual machine (VM).

**Prerequisites**
To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

Procedure

1. Click a row with the name of the VM whose virtual NIC you want to connect.
   The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click **Networks**.
   The Networks pane appears with information about the virtual NICs configured for the VM.

3. Click **Plug** in the row of the virtual NIC you want to connect.
   The selected virtual NIC connects to the VM.

5.9.3. Disconnecting virtual NICs in the RHEL 8 web console

Using the RHEL 8 web console, you can disconnect the virtual network interface cards (NICs) connected to a selected virtual machine.

Prerequisites
To be able to use the RHEL 8 web console to manage virtual machines, you must install the web console virtual machine plug-in.

Procedure

1. Click a row with the name of the virtual machine whose virtual NIC you want to disconnect.
   The row expands to reveal the Overview pane with basic information about the selected virtual machine and controls for shutting down and deleting the virtual machine.

2. Click **Networks**.
   The Networks pane appears with information about the virtual NICs configured for the virtual machine.
3. Click **Unplug** in the row of the virtual NIC you want to disconnect. The selected virtual NIC disconnects from the virtual machine.

### 5.10. INTERACTING WITH VIRTUAL MACHINES USING THE RHEL 8 WEB CONSOLE

To interact with a VM in the RHEL 8 web console, you need to connect to the VM’s console. Using the RHEL 8 web console, you can view the virtual machine’s consoles. These include both graphical and serial consoles.

- To interact with the VM’s graphical interface in the RHEL 8 web console, use the graphical console in the RHEL 8 web 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 RHEL 8 web console, use the serial console in the RHEL 8 web console.

#### 5.10.1. Viewing the virtual machine graphical console in the RHEL 8 web console

You can view the graphical console of a selected virtual machine (VM) in the RHEL 8 web console. The VM console shows the graphical output of the VM.

**Prerequisites**

- To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.
- Ensure that both the host and the VM support a graphical interface.

**Procedure**

1. Click a row with the name of the VM whose graphical console you want to view. The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click **Consoles**. The graphical console appears in the web interface.
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 RHEL 8 web console is running can intercept specific key combinations, such as `Ctrl+Alt+F1`, 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+F1` combination to the VM, click the **Send key** menu and select the `Ctrl+Alt+F1` menu entry.

Additional Resources

- For details on viewing the graphical console in a remote viewer, see Section 5.10.2, “Viewing virtual machine consoles in remote viewers using the RHEL 8 web console”.

- For details on viewing the serial console in the RHEL 8 web console, see Section 5.10.3, “Viewing the virtual machine serial console in the RHEL 8 web console”.

5.10.2. Viewing virtual machine consoles in remote viewers using the RHEL 8 web console

You can view the virtual machine’s consoles in a remote viewer. The connection can be made by the web console or manually.

5.10.2.1. Viewing the graphical console in a remote viewer
You can view the graphical console of a selected virtual machine (VM) in a remote viewer. The VM console shows the graphical output of the VM.

NOTE
You can launch Virt Viewer from within the RHEL 8 web console. Other VNC and SPICE remote viewers can be launched manually.

Prerequisites
- To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.
- Ensure that both the host and the VM support a graphical interface.
- Before you can view the graphical console in Virt Viewer, Virt Viewer must be installed on the machine to which the web console is connected. To view information on installing Virt Viewer, select the Graphics Console in Desktop Viewer Console Type and click More Information in the Consoles window.

NOTE
Some browser extensions and plug-ins do not allow the web console to open Virt Viewer.

Procedure
1. Click a row with the name of the VM whose graphical console you want to view. The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.
2. Click Consoles. The graphical console appears in the web interface.
3. Select the Graphics Console in Desktop Viewer Console Type.
4. Click **Launch Remote Viewer**. The graphical console appears 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 RHEL 8 web console is running can intercept specific key combinations, such as Ctrl+Alt+F1, 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+F1 combination to the VM, click the **Send key** menu and select the **Ctrl+Alt+F1** menu entry.

**Additional Resources**

- For details on viewing the graphical console in a remote viewer using a manual connection, see Section 5.10.2.2, “Viewing the graphical console in a remote viewer connecting manually”.

- For details on viewing the graphical console in the RHEL 8 web console, see Section 5.10.1, “Viewing the virtual machine graphical console in the RHEL 8 web console”.

- For details on viewing the serial console in the RHEL 8 web console, see Section 5.10.3, “Viewing the virtual machine serial console in the RHEL 8 web console”.

**5.10.2.2. Viewing the graphical console in a remote viewer connecting manually**

You can view the graphical console of a selected virtual machine (VM) in a remote viewer. The VM console shows the graphical output of the VM.

The web interface provides the information necessary to launch any SPICE or VNC viewer to view the VM console.

**Prerequisites**

- To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

- Before you can view the graphical console in a remote viewer, ensure that a SPICE or VNC viewer application is installed on the machine to which the web console is connected. To view information on installing Virt Viewer, select the **Graphics Console in Desktop Viewer** Console Type and click **More Information** in the Consoles window.
Procedure
You can view the VM graphics console in any SPICE or VNC viewer application.

1. Click a row with the name of the VM whose graphical console you want to view.
   The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click Consoles.
   The graphical console appears in the web interface.

3. Select the Graphics Console in Desktop Viewer Console Type.
   The following Manual Connection information appears on the right side of the pane.

   Manual Connection
   Connect with any SPICE or VNC viewer application.

   Address: 127.0.0.1
   SPICE Port: 5900
   VNC Port: 5901

4. Enter the information in the SPICE or VNC viewer.

For more information, see the documentation for the SPICE or VNC viewer.

Additional Resources

- For details on viewing the graphical console in a remote viewer using the RHEL 8 web console to make the connection, see Section 5.10.2.1, “Viewing the graphical console in a remote viewer”.

- For details on viewing the graphical console in the RHEL 8 web console, see Section 5.10.1, “Viewing the virtual machine graphical console in the RHEL 8 web console”.

More Information

Clicking "Launch Remote Viewer" will download a .vv file and launch Remote Viewer.

Remote Viewer is available for most operating systems. To install it, search for it in GNOME Software or run the following:

- RHEL, CentOS: sudo yum install virt-viewer
- Fedora: sudo dnf install virt-viewer
- Ubuntu, Debian: sudo apt-get install virt-viewer
- Windows: Download the MSI from virt-manager.org
For details on viewing the serial console in the RHEL 8 web console, see Section 5.10.3, “Viewing the virtual machine serial console in the RHEL 8 web console”.

5.10.3. Viewing the virtual machine serial console in the RHEL 8 web console

You can view the serial console of a selected virtual machine (VM) in the RHEL 8 web console. This is useful when the host machine or the VM is not configured with a graphical interface.

Prerequisites
To be able to use the RHEL 8 web console to manage VMs, you must install the web console VM plug-in.

Procedure

1. Click a row with the name of the VM whose serial console you want to view.
   The row expands to reveal the Overview pane with basic information about the selected VM and controls for shutting down and deleting the VM.

2. Click Consoles.
   The graphical console appears in the web interface.

3. Select the Serial Console Console Type.
   The serial 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

- For details on viewing the graphical console in the RHEL 8 web console, see Section 5.10.1, “Viewing the virtual machine graphical console in the RHEL 8 web console”.

---

To disconnect the serial console from the VM, click Disconnect.

To reconnect the serial console to the VM, click Reconnect.
5.11. CREATING STORAGE POOLS USING THE RHEL 8 WEB CONSOLE

You can create storage pools using the RHEL 8 web console.

Prerequisites

To be able to use the RHEL 8 web console to manage virtual machines, you must install the web console virtual machine plug-in.

If the web console plug-in is not installed, see Section 5.2, “Setting up the RHEL 8 web console to manage virtual machines” for information about installing the web console virtual machine plug-in.

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.

![Storage Pools window](image)

2. Click **Create Storage Pool**. The Create Storage Pool dialog appears.
Enter the following information in the Create Storage Pool dialog:

- **Connection** - The connection to the host to be used by the storage pool.
- **Name** - The name of the storage pool.
- **Type** - The type of the storage pool: Filesystem Directory, Network File System
- **Target Path** - The storage pool path on the host’s file system.
- **Startup** - Whether or not the storage pool starts when the host boots.

4. 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.

**Related information**

- For information on viewing information about storage pools using the RHEL 8 web console, see Section 5.6.2, "Viewing storage pool information using the RHEL 8 web console".

### 5.12. DIFFERENCES BETWEEN VIRTUALIZATION FEATURES IN VIRTUAL MACHINE MANAGER AND THE RHEL 8 WEB CONSOLE

The deprecated Virtual Machine Manager (virt-manager) and its replacement, the RHEL 8 web console, do not have the same functionality. The following table highlights the features that are supported in virt-manager and not supported in the RHEL 8.0 web console.

If a feature is supported in a later version of RHEL, the minimum RHEL version appears in the **RHEL 8 web console** column.

<table>
<thead>
<tr>
<th>Feature</th>
<th>RHEL 8 web console</th>
<th>Alternative method using CLI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>Status</td>
<td>Command</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>--------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Creating new storage pools of the following types:</td>
<td>UNAVAILABLE</td>
<td>virsh pool-define</td>
</tr>
<tr>
<td>• Partition-based</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• GlusterFS-based</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• LVM-based</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• vHBA-based with SCSI devices</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Multipath-based</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• RBD-based</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• iSCSI-direct-based</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creating a new storage volume</td>
<td>UNAVAILABLE</td>
<td>virsh vol-create</td>
</tr>
<tr>
<td>Adding a new virtual network</td>
<td>UNAVAILABLE</td>
<td>virsh net-create</td>
</tr>
<tr>
<td>Deleting a virtual network</td>
<td>UNAVAILABLE</td>
<td>virsh net-define</td>
</tr>
<tr>
<td>Deleting a virtual network</td>
<td>UNAVAILABLE</td>
<td>virsh net-undefine</td>
</tr>
<tr>
<td>Creating a bridge from a host machine’s interface to a virtual</td>
<td>UNAVAILABLE</td>
<td>virsh iface-bridge</td>
</tr>
<tr>
<td>machine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creating a snapshot</td>
<td>UNAVAILABLE</td>
<td>virsh snapshot-create-as</td>
</tr>
<tr>
<td>Reverting to a snapshot</td>
<td>UNAVAILABLE</td>
<td>virsh snapshot-revert</td>
</tr>
<tr>
<td>Deleting a snapshot</td>
<td>UNAVAILABLE</td>
<td>virsh snapshot-delete</td>
</tr>
<tr>
<td>Cloning a virtual machine</td>
<td>UNAVAILABLE</td>
<td>virt-clone</td>
</tr>
<tr>
<td>Migrating a virtual machine to another host machine</td>
<td>UNAVAILABLE</td>
<td>virsh migrate</td>
</tr>
</tbody>
</table>
CHAPTER 6. 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 how they can be attached, modified, or removed from a VM.

6.1. HOW VIRTUAL DEVICES WORK

The basics
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.

Virtual devices attached to a VM can be configured when creating the VM, but can also be managed on an existing VM. Generally, the virtual devices attached to a VM can only be configured when the VM is shut off, but some can be added or removed when the VM is running. This feature is also 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 8 web console.

Performance or flexibility
For some types of devices, RHEL 8 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 virtual machine using a variety of controllers: emulated controller, virtio-scsi, or 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 virtual machine. Finally, virtio-blk provides better performance than both virtio-scsi and emulated controllers, but a more limited range of use-cases.
For more information on types of virtual devices, see Section 6.5, “Types of virtual devices”.

Additional resources

- For instructions how to attach, remove, or modify VM storage devices using the CLI, see Chapter 7, Managing storage for virtual machines.
- For instructions how to manage VM disks using the RHEL 8 web console, see Section 5.8, “Managing virtual machine disks using the RHEL 8 web console”.
- For instructions how to manage VM NICs using the RHEL 8 web console, see Section 5.9, “Using the RHEL 8 web console for managing virtual machine vNICs”.
- For instructions how to create and manage NVIDIA vGPUs, see Chapter 8, Managing NVIDIA vGPU devices.

6.2. ATTACHING DEVICES TO VIRTUAL MACHINES

The following provides general information for creating and attaching virtual devices to your virtual machines (VMs) using the command-line interface (CLI). Some devices can also be attached to VMs using the RHEL 8 web console.

Prerequisites

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

  ```bash
  # 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 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:

     ```bash
     # 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:

     ```bash
     # 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.

2. **[Optional]** Verify the device has been added by doing 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.

- Run the VM and test if the device is present and works properly. For example, the following displays the configuration of the `testguest` VM and confirms that the 002.004 USB flash disk device has been added.

```bash
# 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>
[...]
```

**Additional resources**

- For further information on using the `virt-xml` command, use `man virt-xml`.

### 6.3. MODIFYING DEVICES ATTACHED TO VIRTUAL MACHINES

The following provides general information 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 8 web console.

**Prerequisites**

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

```bash
# 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
```
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`:

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

2. [Optional] Verify the device has been modified by doing 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 displays the configuration of the `testguest` VM and confirms that the CPU mode has been configured as `host-model`.

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

3. [Optional] 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.

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

**NOTE**

For small changes directly in 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 would prevent the VM from booting.

Additional resources

- For details on using the `virt-xml` command, use `man virt-xml`.

6.4. REMOVING DEVICES FROM VIRTUAL MACHINES
The following provides general information for removing 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 8 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:

```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 --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:

     ```bash
     # virsh 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 an USB flash drive device from the running `testguest2` VM:

     ```bash
     # virsh xml testguest2 --remove-device --update --hostdev type=usb
     Device hotunplug successful.
     Domain '7.4-workstation' defined successfully.
     ```

2. [Optional] 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.

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

Additional resources

- For details on using the `virt-xml` command, use `man virt-xml`.

6.5. TYPES OF VIRTUAL DEVICES

Virtualization in RHEL 8 can present three distinct types of virtual devices to virtual machines (VMs):

Emulated devices

Emulated devices are software implementations of widely used physical devices. Standard device drivers designed to interact with physical devices will work identically with emulated devices. Therefore, emulated devices do not need specific drivers. As such, 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 in comparison to the corresponding physical devices or to more optimized virtual devices.

The following types of emulated devices are supported:

- Virtual CPUs (vCPUs), with a large choice of CPU models being 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 or QXL 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 virtual machines, and thus significantly increase the device performance. RHEL 8 provides paravirtualized devices to virtual machines 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. Whenever possible, it is recommended to use paravirtualized devices instead of emulated devices for VM, 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 virtual machines 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 share information about guest memory usage with the hypervisor
- The paravirtualized random number generator (*virtio-rng*)
- The paravirtualized graphics card (*QXL*)

**Physically shared devices**
Certain hardware platforms enable virtual machines to directly access various hardware devices and components. This process is known as device assignment, or also as 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:

- Virtual Function I/O (VFIO) device assignment - safely exposes devices to applications or virtual machines using hardware-enforced DMA and interrupt isolation.

- 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.
CHAPTER 7. MANAGING STORAGE FOR VIRTUAL MACHINES

You can manage virtual machine storage using the CLI or the web console.

This documentation provides information on how to manage virtual machine storage using the virsh command.

7.1. UNDERSTANDING VIRTUAL MACHINE STORAGE

The following sections provide information about storage for virtual machines (VMs), including information about storage pools, storage volumes, and how they are used to provide storage for VMs.

7.1.1. Virtual machine storage

The following provides information about how storage pools and storage volumes are used to create storage for VMs.

A storage pool is a quantity of storage managed by the host set aside for use by VMs. Storage volumes can be created from space in the storage pools. Each storage volume can be assigned to a VM as a block device on a guest bus.

Storage pools and volumes are managed using libvirt. With the libvirt remote protocol, you can manage all aspects of virtual machine storage. These operations can be performed on a remote host. As a result, a management application, such as the RHEL web console, using libvirt can enable a user to perform all the required tasks for configuring virtual machine storage.

The libvirt API can be used to query the list of volumes in the storage pool or to get information regarding the capacity, allocation, and available storage in the storage pool. A storage volume in the storage pool may be queried to get information such as allocation and capacity, which may differ for sparse volumes.

For storage pools that support it, the libvirt API can be used to create, clone, resize, and delete storage volumes. The APIs can also be used to upload data to storage volumes, download data from storage volumes, or wipe data from storage volumes.

Once a storage pool is started, a storage volume can be assigned to a VM using the storage pool name and storage volume name instead of the host path to the volume in the XML configuration files of the virtual machine.

7.1.2. Storage pools

A storage pool is a file, directory, or storage device, managed by libvirt to provide storage to virtual machines. Storage pools are divided into storage volumes that store virtual machine images or are attached to VMs as additional storage. 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.
- A transient storage pool only exists until the host reboots.

The virsh pool-define command is used to create a persistent storage pool, and the virsh pool-create command is used 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 large numbers of VMs.

- **Networked (shared) storage pools**
  Networked storage pools include storage devices shared over a network using standard protocols.

Storage pool actions
Storage pools can be stopped (destroyed). This removes the abstraction of the data, but keeps the data intact.

For example, an NFS server that uses `mount -t nfs nfs.example.com:/path/to/share /path/to/data`. A storage administrator responsible could define an NFS Storage Pool on the virtualization host to describe the exported server path and the client target path. This will allow `libvirt` to perform the mount either automatically when `libvirt` is started or as needed while `libvirt` is running. Files with the NFS Server exported directory are listed as storage volumes within the NFS storage pool.

When the storage volume is added to the VM, the administrator does not need to add the target path to the volume. He just needs to add the storage pool and storage volume by name. Therefore, if the target client path changes, it does not affect the virtual machine.

When the storage pool is started, `libvirt` mounts the share on the specified directory, just as if the system administrator logged in and executed `mount nfs.example.com:/path/to/share /vmdata`. If the storage pool is configured to autostart, `libvirt` ensures that the NFS shared disk is mounted on the directory specified when `libvirt` is started.

Once the storage pool is started, the files in the NFS shared disk are reported as storage volumes, and the storage volumes’ paths may be queried using the `libvirt` API. The storage volumes’ paths can then be copied into the section of a virtual machine’s XML definition that describes the source storage for the virtual machine’s block devices. In the case of NFS, an application that uses the `libvirt` API can 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).

Not all storage pool types support creating and deleting volumes. Stopping the 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 details, see `man virsh`.

Supported and unsupported 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
- GlusterFS storage pools
• iSCSI-based storage pools
• LVM-based storage pools
• NFS-based storage pools
• vHBA-based storage pools with SCSI devices
• Multipath-based storage pools
• RBD-based storage pools

The following is a list of libvirt storage pool types that are not supported by RHEL:

• Sheepdog-based storage pools
• Vstorage-based storage pools
• ZFS-based storage pools

7.1.3. 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 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, a volume named firstimage in the guest_images pool.

```bash
# virsh vol-info --pool guest_images firstimage
Name:             firstimage
Type:             block
Capacity:         20.00 GB
Allocation:       20.00 GB
```

7.2. MANAGING STORAGE FOR VIRTUAL MACHINES USING THE CLI

This documentation provides information on how to manage virtual machine storage using the virsh command.

You can add, remove, and modify virtual machine storage using the CLI. You can also view information about virtual machine storage.

NOTE

In many cases, storage for a VM is created at the same time the VM is created. Therefore, the following information primarily relates to advanced management of VM storage.

7.2.1. Viewing virtual machine storage information using the CLI

The following provides information about viewing information about storage pools and storage volumes using the CLI.
7.2.1.1. Viewing storage pool information using the CLI

Procedure

- Use the `virsh pool-list` command to view storage pool information.

```
# 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
RHEL8-Storage-Pool   running   yes        yes          214.62 GiB   93.02 GiB  168.60 GiB
```

Additional resources

- For information on the available `virsh pool-list` options, see the relevant `man` pages.

7.2.1.2. Viewing storage volume information using the CLI

The following provides information on viewing information about storage pools. You can view a list of all storage pools in a specified storage pool and details about a specified storage pool.

Procedure

1. Use the `virsh vol-list` command to list the storage volumes in a specified storage pool.

```
# virsh vol-list --pool RHEL8-Storage-Pool --details
Name                Path                                               Type   Capacity  Allocation
-------------------------------------------------------------------------------------------------------------
.bash_history        /home/VirtualMachines/.bash_history       file   18.70 KiB   20.00 KiB
.bash_logout         /home/VirtualMachines/.bash_logout        file    18.00 B    4.00 KiB
.bash_profile        /home/VirtualMachines/.bash_profile       file   193.00 B    4.00 KiB
.bashrc              /home/VirtualMachines/.bashrc             file   1.29 KiB    4.00 KiB
.git-prompt.sh       /home/VirtualMachines/.git-prompt.sh      file  15.84 KiB   16.00 KiB
.gitconfig           /home/VirtualMachines/.gitconfig          file   167.00 B    4.00 KiB
RHEL8_Volume.qcow2   /home/VirtualMachines/RHEL8_Volume.qcow2  file  60.00 GiB  13.93 GiB
```

NOTE

For information on the available `virsh vol-list` options, see the relevant `man` pages.

2. Use the `virsh vol-info` command to list the storage volumes in a specified storage pool.

```
# vol-info --pool RHEL8-Storage-Pool --vol RHEL8_Volume.qcow2
Name:     RHEL8_Volume.qcow2
Type:     file
Capacity: 60.00 GiB
Allocation: 13.93 GiB
```

NOTE

For information on the available `virsh vol-info` options, see the relevant `man` pages.
7.2.2. Creating and assigning storage for virtual machines using the CLI

The following is a high-level procedure for creating and assigning storage for virtual machines (VMs):

1. Create storage pools
   Create one or more storage pools from available storage media. For a list of supported storage pool types, see Storage pool types.
   - To create persistent storage pools, use the `virsh pool-define` and `virsh pool-define-as` commands. The `virsh pool-define` command uses an XML file for the pool options. The `virsh pool-define-as` command places the options in the command line.
   - To create temporary storage pools, use the `virsh pool-create` and `virsh pool-create-as` commands. The `virsh pool-create` command uses an XML file for the pool options. The `virsh pool-create-as` command places the options in the command line.

   **NOTE**

   All examples and procedures in this documentation are for creating persistent storage pools using the `virsh pool-define` command. For more information on the `virsh pool-create`, `virsh pool-define-as`, and `virsh pool-create-as` commands, see the relevant man pages.

2. Create storage volumes
   Create one or more storage volumes from the available storage pools.

   **NOTE**

   All examples and procedures in this documentation are for creating storage using the `virsh vol-create` command. For more information on the `virsh vol-create-as` command, see the relevant man pages.

3. Assign storage devices to a VM
   Assign one or more storage devices abstracted from storage volumes to a VM.

The following sections provide information on creating and assigning storage using the CLI:

- Directory-based storage
- Filesystem-based storage
- GlusterFS-based storage
- iSCSI-based storage
- LVM-based storage
- NFS-based storage
- vHBA-based storage

7.2.2.1. Creating and assigning directory-based storage for virtual machines using the CLI
The following provides information about creating directory-based storage pools and storage volumes and assigning volumes to virtual machines.

### 7.2.2.1.1. Creating directory-based storage pools using the CLI

The following provides instructions for creating directory-based storage pools.

#### Procedure

1. **Define the storage pool in an XML file**
   Create a temporary XML file containing the storage pool parameters required for the new device.

   ```
   # virsh pool-define ~/guest_images.xml
   Pool defined from guest_images_fs
   ``

   **NOTE**

   For information on the required parameters, refer to Parameters.

2. **Create a storage pool**
   Use the `virsh pool-define` command to create a persistent storage pool based on the XML file created in the previous step.

   ```
   # virsh pool-build guest_images_fs
   Pool guest_images_fs built
   ``

   **NOTE**

   You can delete the XML file created in step 1 after running the `virsh pool-define` command.

3. **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.

   ```
   # 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 ..
   ``

4. **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
   ```

5. **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.

4. **[Optional] Turn on autostart**

   By default, a storage pool defined with the `virsh` command is not set to automatically start each time `libvirtd` starts. 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
```

5. **Verify the Autostart state**

   Use the `virsh pool-list` command to verify the Autostart state.

```
# virsh pool-list --all

Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images_fs      inactive   yes
```

6. **Verify the storage pool**

   Verify that the storage pool was created correctly, the sizes reported are as expected, and the state is reported as `running`. Verify there is a `lost+found` directory in the target path on the file system, indicating that the device is mounted.

```
# 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

# 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 16384 May 31 14:18 lost+found
7.2.2.1.2. Directory-based storage pool parameters

The following provides information about the required parameters for a directory-based storage pool and an example.

Parameters

The following table provides a list of required parameters for the XML file for a directory-based storage pool.

Table 7.1. Directory-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='dir'&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 target. This will be the path used for the storage pool.</td>
<td>&lt;target&gt; &lt;path&gt;target_path&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 /guest_images directory:

```xml
<pool type='dir'>
  <name>dirpool</name>
  <target>
    <path>/guest_images</path>
  </target>
</pool>
```

7.2.2.2. Creating and assigning disk-based storage for virtual machines using the CLI

The following provides information about creating disk-based storage pools and storage volumes and assigning volumes to virtual machines.

7.2.2.2.1. Creating disk-based storage pools using the CLI

The following provides instructions for creating disk-based storage pools.

Recommendations

Be aware of the following before creating a disk-based 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. It is strongly recommended that you back up the data on the storage device before creating a storage pool.

- VMs should not be given write access to whole disks or block devices (for example, /dev/sdb). Use partitions (for example, /dev/sdb1) or LVM volumes. If you pass an entire block device to a 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.
Procedure

1. Relabeled the disk with a GUID Partition Table (GPT) disk label. GPT disk labels allow for creating up to 128 partitions on each device.

```
# parted /dev/sdb
GNU Parted 2.1
Using /dev/sdb
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) mklabel
New disk label type? gpt
(parted) quit
Information: You may need to update /etc/fstab.
```

2. Define the storage pool in an XML file
Create a temporary XML file containing the storage pool parameters required for the new device.

```
NOTE
For information on the required parameters, refer to Parameters.
```

3. Create a storage pool
Use the `virsh pool-define` command to create a persistent storage pool based on the XML file created in the previous step.

```
# virsh pool-define ~/guest_images.xml
Pool defined from guest_images_fs
```

```
NOTE
You can delete the XML file created in step 1 after running the `virsh pool-define` command.
```

1. 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
drw------- 2 root root 4096 May 31 19:38 .
dr-xr-xr-x 25 root root 4096 May 31 19:38 ..
```
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.

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_fs inactive no
   ```

3. **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.

4. **[Optional] Turn on autostart**
   By default, a storage pool defined with the `virsh` command is not set to automatically start each time libvirtd starts. 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
   ```

5. **Verify the Autostart state**
   Use the `virsh pool-list` command to verify the Autostart state.

   ```
   # virsh pool-list --all
   Name       State  Autostart
   ---------------------
   default     active  yes
   guest_images_fs inactive yes
   ```

6. **Verify the storage pool**
   Verify that the storage pool was created correctly, the sizes reported are as expected, and the state is reported as **running**. Verify there is a `lost+found` directory in the target path on the file system, indicating that the device is mounted.

   ```
   # virsh pool-info guest_images_fs
   ```
7.2.2.2.2. Disk-based storage pool parameters

The following provides information about the required parameters for a directory-based storage pool and an example.

Parameters

The following table provides a list of required parameters for the XML file for a disk-based storage pool.

<table>
<thead>
<tr>
<th>Description</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>The type of storage pool</td>
<td>&lt;pool type='disk'&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 storage device. For example, /dev/sdb.</td>
<td>&lt;source&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;path&gt;source_path&lt;/path&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/source&gt;</td>
</tr>
<tr>
<td>The path specifying the target device. This will be the path used</td>
<td>&lt;target&gt;</td>
</tr>
<tr>
<td>for the storage pool.</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 disk-based storage pool:

```xml
<pool type='disk'>
 <name>phy_disk</name>
 <source>
  <device path='/dev/sdb'/>  
  <format type='gpt'/>
 </source>
 <target>
</target>
```
7.2.2.3. Creating and assigning filesystem-based storage for virtual machines using the CLI

The following provides information about creating directory-based storage pools and storage volumes and assigning volumes to virtual machines.

7.2.2.3.1. Creating filesystem-based storage pools using the CLI

The following provides instructions for creating filesystem-based storage pools.

Recommendations
Do not use this procedure to assign an entire disk as a storage pool (for example, `/dev/sdb`). VMs should not be given write access to whole disks or block devices. This method should only be used to assign partitions (for example, `/dev/sdb1`) to storage pools.

Procedure

1. **Define the storage pool in an XML file**
   Create a temporary XML file containing the storage pool parameters required for the new device.

   ```xml
   <path>/dev</path>
   </target>
   </pool>
   ```

   **NOTE**
   For information on the required parameters, refer to Parameters.

2. **Create a storage pool**
   Use the `virsh pool-define` command to create a persistent storage pool based on the XML file created in the previous step.

   ```
   # virsh pool-define ~/guest_images.xml
   Pool defined from guest_images_fs
   ```

   **NOTE**
   You can delete the XML file created in step 1 after running the `virsh pool-define` command.

1. **Define the storage pool target path**
   Use the `virsh pool-build` command to create a storage pool target path for a pre-formatted filesystem 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 ..
   ```

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_fs inactive no
```

3. **Start the storage pool**
   Use the `virsh pool-start` command to mount the storage pool.
   ```bash
   # 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.

4. **[Optional] Turn on autostart**
   By default, a storage pool defined with the `virsh` command is not set to automatically start each time `libvirtd` starts. Use the `virsh pool-autostart` command to configure the storage pool to autostart.
   ```bash
   # virsh pool-autostart guest_images_fs
   Pool guest_images_fs marked as autostarted
   ```

5. **Verify the `Autostart` state**
   Use the `virsh pool-list` command to verify the `Autostart` state.
   ```bash
   # virsh pool-list --all

   Name           State   Autostart
               ---------------
   default        active  yes
   guest_images_fs inactive yes
   ```

6. **Verify the storage pool**
   Verify that the storage pool was created correctly, the sizes reported are as expected, and the state is reported as `running`. Verify there is a `lost+found` directory in the target path on the file system, indicating that the device is mounted.
   ```bash
   # 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
   ```
7.2.2.3.2. Filesystem-based storage pool parameters

The following provides information about the required parameters for a directory-based storage pool and an example.

**Parameters**

The following table provides a list of required parameters for the XML file for a filesystem-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='fs'&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 partition. For example, /dev/sdc1</td>
<td>`&lt;source&gt;</td>
</tr>
<tr>
<td></td>
<td><code>&lt;device path=device_path /&gt;</code></td>
</tr>
<tr>
<td>The filesystem type, for example ext4.</td>
<td><code>&lt;format type=fs_type /&gt;</code></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><code>&lt;path&gt;path-to-pool&lt;/path&gt;</code></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 /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>
```

7.2.2.4. Creating and assigning GlusterFS storage for virtual machines using the CLI
The following provides information about creating directory-based storage pools and storage volumes and assigning volumes to virtual machines.

7.2.2.4.1. Creating GlusterFS-based storage pools using the CLI

GlusterFS is a user space file system that uses File System in Userspace (FUSE).

The following provides instructions for creating GlusterFS-based storage pools.

Prerequisites

- Before a GlusterFS-based storage pool can be created on a host, a Gluster server must be prepared.
  1. Obtain the IP address of the Gluster server by listing its status with the following command:

```
# gluster volume status
Status of volume: gluster-vol1
Gluster process                           Port Online Pid
------------------------------------------------------------
Brick 222.111.222.111:/gluster-vol1       49155   Y    18634
```

Task Status of Volume gluster-vol1

```
There are no active volume tasks
```

2. If not installed, install the `glusterfs-fuse` package.

3. If not enabled, enable the `virt_use_fusefs` boolean. Check that it is enabled.

```
# setsebool virt_use_fusefs on
# getsebool virt_use_fusefs
virt_use_fusefs --> on
```

After ensuring that the required packages are installed and enabled, continue creating the storage pool.

Procedure

1. Define the storage pool in an XML file
   Create a temporary XML file containing the storage pool parameters required for the new device.

   **NOTE**

   For information on the required parameters, refer to Parameters.

2. Create a storage pool
   Use the `virsh pool-define` command to create a persistent storage pool based on the XML file created in the previous step.

   ```
   # virsh pool-define ~/guest_images.xml
   Pool defined from guest_images_fs
   ```

   You can delete the XML file created in step 1 after running the `virsh pool-define` command.
1. 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 ..
   ```

   **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.

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_fs      inactive   no
   ```

3. 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.

4. [Optional] Turn on autostart
   By default, a storage pool defined with the `virsh` command is not set to automatically start each time `libvirtd` starts. 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
   ```

5. Verify the **Autostart** state
   Use the `virsh pool-list` command to verify the **Autostart** state.
6. **Verify the storage pool**

Verify that the storage pool was created correctly, the sizes reported are as expected, and the state is reported as **running**. Verify there is a **lost+found** directory in the target path on the file system, indicating that the device is mounted.

```bash
# virsh pool-list --all
```

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
<th>Autostart</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>active</td>
<td>yes</td>
</tr>
<tr>
<td>guest_images_fs</td>
<td>inactive</td>
<td>yes</td>
</tr>
</tbody>
</table>

```bash
# 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
```

```bash
# mount | grep /guest_images
/dev/sdc1 on /guest_images type ext4 (rw)
```

```bash
# 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
```

### 7.2.2.4.2. GlusterFS-based storage pool parameters

The following provides information about the required parameters for a GlusterFS-based storage pool and an example.

**Parameters**

The following table provides a list of required parameters for the XML file for a GlusterFS-based storage pool.

**Table 7.4. GlusterFS-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='gluster'&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 hostname or IP address of the Gluster server | `<source>
    <name=gluster-name />
</source>` |
| The name of the Gluster server           | `<name>name</name>`                      |
The path on the Gluster server used for the storage pool.

<table>
<thead>
<tr>
<th>Description</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>The path on the Gluster server used for the storage pool.</td>
<td>&lt;dir path=&quot;gluster-path&quot;/&gt;</td>
</tr>
</tbody>
</table>

**Example**
The following is an example of an XML file for a storage pool based on the Gluster file system at 111.222.111.222:

```xml
<pool type='gluster'>
  <name>Gluster_pool</name>
  <source>
    <host name='111.222.111.222'/>
    <dir path='.'/>
    <name>gluster-vol1</name>
  </source>
</pool>
```

### 7.2.2.5. Creating and assigning iSCSI-based storage for virtual machines using the CLI

The following provides information about creating iSCSI-based storage pools and storage volumes, securing iSCSI-based storage pools with libvirt secrets, and assigning volumes to virtual machines.

**Recommendations**

Internet Small Computer System Interface (iSCSI) is a network protocol for sharing storage devices. iSCSI connects initiators (storage clients) to targets (storage servers) using SCSI instructions over the IP layer.

Using iSCSI-based devices to store virtual machines allows for more flexible storage options, such as using iSCSI as a block storage device. The iSCSI devices use a Linux-IO (LIO) target. This is a multi-protocol SCSI target for Linux. In addition to iSCSI, LIO also supports Fibre Channel and Fibre Channel over Ethernet (FCoE).

If you need to prevent access to an iSCSI storage pool, you can secure it using a libvirt secret.

**Prerequisites**

- Before an iSCSI-based storage pool can be created, iSCSI targets must be created. iSCSI targets are created with the `targetcli` package, which provides a command set for creating software-backed iSCSI targets. For more information and instructions on creating iSCSI targets, see the *Red Hat Enterprise Linux Storage Administration Guide*.

### 7.2.2.5.1. Creating iSCSI-based storage pools using the CLI

The following provides instructions for creating iSCSI-based storage pools.

**Procedure**

1. **Define the storage pool in an XML file**
   Create a temporary XML file containing the storage pool parameters required for the new device.
1. 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
```

2. 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
```

3. [Optional] Turn on autostart
   By default, a storage pool defined with the `virsh` command is not set to automatically start each time libvirtd starts. 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
```

4. Verify the **Autostart** state
   Use the `virsh pool-list` command to verify the **Autostart** state.

```
# virsh pool-list --all

Name                State  Autostart
-------------------------------
default             active  yes
guest_images_fs     inactive yes
```
5. **Verify the storage pool**

Verify that the storage pool was created correctly, the sizes reported are as expected, and the state is reported as *running*. Verify there is a **lost+found** directory in the target path on the file system, indicating that the device is mounted.

```
# 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
```

```
# 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 ..
drw------.  2 root root 16384 May 31 14:18 lost+found
```

### 7.2.2.5.2. iSCSI-based storage pool parameters

The following provides information about the required parameters for an iSCSI-based storage pool and an example.

**Parameters**

The following table provides a list of required parameters for the XML file for an iSCSI-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='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>
</tbody>
</table>
| The name of the host | `<source>
   <host name=hostname />
</source>` |
| The iSCSI IQN | `<device path= iSCSI_IQN />`
   `<source>
   <host name=hostname />
</source>` |
| The path specifying the target. This will be the path used for the storage pool. | `<target>
   <path>/dev/disk/by-path</path>
</target>` |
The IQN of the iSCSI initiator. This is only needed when the ACL restricts the LUN to a particular initiator.

```xml
<initiator>
  <iqn name='initiator0' />
</initiator>
```

**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:iscsirhel7guest'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

### 7.2.2.5.3. Securing iSCSI storage pools with libvirt secrets

User name and password parameters can be configured with `virsh` to secure an iSCSI storage pool. This can be configured before or after the pool is defined, 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.

**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:
   ```
   # MYSECRET=`printf *%s "password123" | base64`
   # virsh secret-set-value 2d7891af-20be-4e5e-af83-190e8a922360 $MYSECRET
   ```

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:
   ```xml
   <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.
   ```xml
   <auth username='redhat'>
    <secret type='iscsi' usage='iscsirhel7secret'/>
   </auth>
   ```

6. To activate the changes, the storage pool must be activated. If the pool has already been started, stop and restart the storage pool:
   ```
   # virsh pool-destroy iscsirhel7pool
   ```
# virsh pool-start iscsirhel7pool

7.2.2.6. Creating and assigning LVM-based storage for virtual machines using the CLI

The following provides information about creating LVM-based storage pools and storage volumes and assigning volumes to virtual machines.

7.2.2.6.1. Creating LVM-based storage pools using the CLI

The following provides instructions for creating 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 Logical Volume Manager commands or `virsh` commands. To manage volume groups using the `virsh` interface, use the `virsh` commands to create volume groups.
  For more 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 activating a new partition or device with these procedures, the partition will be formatted and all data will be erased. If using the host’s existing Volume Group (VG) nothing will be erased. It is recommended to back up the storage device before starting.

Procedure

1. **Define the storage pool in an XML file**
   Create a temporary XML file containing the storage pool parameters required for the new device.

   ```
   # virsh pool-define guest_images.fs
   Pool defined from guest_images_fs
   ```

   **NOTE**
   For information on the required parameters, refer to Parameters.

2. **Create a storage pool**
   Use the `virsh pool-define` command to create a persistent storage pool based on the XML file created in the previous step.

   ```
   # virsh pool-define ~/guest_images.xml
   ```

   **NOTE**
   You can delete the XML file created in step 1 after running the `virsh pool-define` command.

1. **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

2. **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.

3. **[Optional] Turn on autostart**
   By default, a storage pool defined with the **virsh** command is not set to automatically start each time libvirtd starts. 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

4. **Verify the Autostart state**
   Use the **virsh pool-list** command to verify the **Autostart** state.

   # virsh pool-list --all
   Name       State   Autostart
   -----------------------------------------
default    active yes
guest_images_fs inactive yes

5. **Verify the storage pool**
   Verify that the storage pool was created correctly, the sizes reported are as expected, and the state is reported as **running**. Verify there is a **lost+found** directory in the target path on the file system, indicating that the device is mounted.

   # 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

   # 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

7.2.2.6.2. LVM-based storage pool parameters

The following provides information about the required parameters for an LVM-based storage pool and an example.

Parameters
The following table provides a list of required parameters for the XML file for a LVM-based storage pool.

Table 7.6. 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;</code> <code>name</code>&lt;/name&gt;`</td>
</tr>
<tr>
<td>The path to the device for the storage pool</td>
<td><code>&lt;source&gt;</code> <code>&lt;device path='device_path'/&gt;</code></td>
</tr>
<tr>
<td>The name of the volume group</td>
<td><code>&lt;name&gt;</code> <code>VG-name</code>&lt;/name&gt;`</td>
</tr>
<tr>
<td>The virtual group format</td>
<td><code>&lt;format type='lvm2' /&gt;</code></td>
</tr>
<tr>
<td>The target path</td>
<td><code>&lt;target&gt;</code> <code>&lt;path=target_path /&gt;</code></td>
</tr>
</tbody>
</table>

**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:
7.2.2.7. Creating and assigning network-based storage for virtual machines using the CLI

The following provides information about creating network-based storage pools and assigning volumes to virtual machines.

7.2.2.7.1. Prerequisites

- To create an Network File System (NFS)-based storage pool, an NFS Server should already be configured to be used by the host machine. For more information about NFS, refer to the Red Hat Enterprise Linux Storage Administration Guide.

- Ensure that the required utilities for the file system used is installed on the host. For example, cifs-utils for Common Internet File Systems (CIFS) and glusterfs.fuse for GlusterFS.

7.2.2.7.2. Creating network-based storage pools using the CLI

The following provides instructions for creating network-based storage pools.

Procedure

1. Define the storage pool in an XML file
   Create a temporary XML file containing the storage pool parameters required for the new device.

   ```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>
   ```

   NOTE
   For information on the required parameters, refer to Parameters.

2. Create a storage pool
   Use the `virsh pool-define` command to create a persistent storage pool based on the XML file created in the previous step.

   ```sh
   # virsh pool-define ~/guest_images.xml
   Pool defined from guest_images_fs
   ```

   NOTE
   You can delete the XML file created in step 1 after running the `virsh pool-define` command.

1. 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.

```bash
# virsh pool-build guest_images_fs
Pool guest_images_fs built
```

```bash
# 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 ..
```

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_fs      inactive   no
```

3. **Start the storage pool**
   Use the `virsh pool-start` command to mount the storage pool.

```bash
# 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.

4. **[Optional] Turn on autostart**
   By default, a storage pool defined with the `virsh` command is not set to automatically start each time libvirtd starts. Use the `virsh pool-autostart` command to configure the storage pool to autostart.

```bash
# virsh pool-autostart guest_images_fs
Pool guest_images_fs marked as autostarted
```

5. **Verify the **Autostart** state**
   Use the `virsh pool-list` command to verify the **Autostart** state.

```bash
# virsh pool-list --all
Name                  State      Autostart
-----------------------------------------
default              active     yes
guest_images_fs      inactive   yes
```

6. **Verify the storage pool**
Verify that the storage pool was created correctly, the sizes reported are as expected, and the state is reported as `running`. Verify there is a `lost+found` directory in the target path on the file system, indicating that the device is mounted.

```bash
# 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

# 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 ..
drw------.  2 root root 16384 May 31 14:18 lost+found
```

### 7.2.2.7.3. NFS-based storage pool parameters

The following provides information about the required parameters for an NFS-based storage pool and an example.

**Parameters**

The following table provides a list of required parameters for the XML file for an NFS-based storage pool.

**Table 7.7. 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;</code> name&lt;/name&gt;`</td>
</tr>
<tr>
<td>The hostname of the network server where the</td>
<td><code>&lt;source&gt;</code></td>
</tr>
<tr>
<td>mount point is located. This can be a</td>
<td></td>
</tr>
<tr>
<td>hostname or an IP address.</td>
<td><code>&lt;host name='hostname'/&gt;</code></td>
</tr>
<tr>
<td>The format of the storage pool</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td><code>&lt;format type='nfs'/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;format type='glusterfs'/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;format type='cifs'/&gt;</code></td>
</tr>
<tr>
<td>The directory used on the network server</td>
<td><code>&lt;dir path='source_path'/&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/source&gt;</code></td>
</tr>
</tbody>
</table>
The path specifying the target. This will be the path used for the storage pool.

<table>
<thead>
<tr>
<th>Description</th>
<th>XML</th>
</tr>
</thead>
</table>
| The path specifying the target. This will be the path used for the storage pool. | `<target>
  <path>target_path</path>
</target>` |

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:

```
<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>
```

7.2.2.8. Creating and assigning vHBA-based storage for virtual machines using the CLI

The following provides information about creating vHBA-based storage pools and storage volumes and assigning volumes to virtual machines (VMs).

7.2.2.8.1. Recommendations

N_Port ID Virtualization (NPIV) is a software technology that allows sharing of a single physical Fibre Channel host bus adapter (HBA). This allows multiple VMs to see the same storage from multiple physical hosts, and thus allows for easier migration paths for the storage. As a result, there is no need for the migration to create or copy storage, as long as the correct storage path is specified.

In virtualization, the virtual host bus adapter, or vHBA, controls the Logical Unit Numbers (LUNs) for VMs. For a host to share one Fibre Channel device path between multiple VMs, a vHBA must be created for each VM. A single vHBA must not be used by multiple VMs.

Each vHBA for NPIV is identified by its parent HBA and its own World Wide Node Name (WWNN) and World Wide Port Name (WWPN). The path to the storage is determined by the WWNN and WWPN values. The parent HBA can be defined as `scsi_host#` or as a WWNN/WWPN pair.

**NOTE**

If a parent HBA is defined as `scsi_host#` and hardware is added to the host machine, the `scsi_host#` assignment may change. Therefore, it is recommended that you define a parent HBA using a WWNN/WWPN pair.

It is recommended that you define a libvirt storage pool based on the vHBA, because this preserves the vHBA configuration.

Using a libvirt storage pool has two primary advantages:
- The libvirt code can easily find the LUN’s path via virsh command output.

- Virtual machine migration requires only defining and starting a storage pool with the same vHBA name on the target machine. To do this, the vHBA LUN, libvirt storage pool and volume name must be specified in the VM’s XML configuration.

**NOTE**

Before creating a vHBA, it is recommended that you configure storage array (SAN)-side zoning in the host LUN to provide isolation between VMs and prevent the possibility of data corruption.

To create a persistent vHBA configuration, first create a libvirt 'scsi' storage pool XML file. For information on the XML file, see Creating vHBAs. When creating a single vHBA that uses a storage pool on the same physical HBA, it is recommended to use a stable location for the `<path>` value, such as one of the `/dev/disk/by-`{path|id|uuid|label}` locations on your system.

When creating multiple vHBAs that use storage pools on the same physical HBA, the value of the `<path>` field must be only `/dev/`, otherwise storage pool volumes are visible only to one of the vHBAs, and devices from the host cannot be exposed to multiple VMs with the NPIV configuration.

For more information on `<path>` and the elements in `<target>`, see upstream libvirt documentation.

### 7.2.2.8.2. Prerequisites

Before creating a vHBA-based storage pools with SCSI devices, create a vHBA.

### 7.2.2.8.3. Creating vHBAs

The following provides instructions on creating a virtual host bus adapter (vHBA).

**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:

```
# virsh nodedev-list --cap vports
scsi_host3
scsi_host4
```

2. View the HBA’s details, using the `virsh nodedev-dumpxml HBA_device` command.

```
# virsh nodedev-dumpxml scsi_host3
```

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>
  </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'>
      <max_vports>127</max_vports>
      <vports>0</vports>
    </capability>
  </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'>
      <max_vports>127</max_vports>
      <vports>0</vports>
    </capability>
  </capability>
</device>
```

**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 `noddev` 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
```

5. 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>
    <capability type='fc_host'>
        <wwnn>5001a4a93526d0a1</wwnn>
        <wwpn>5001a4ace3ee047d</wwpn>
        <fabric_wwn>2002000573de9a81</fabric_wwn>
    </capability>
</device>
```

---

7.2.2.8.4. Creating vHBA-based storage pools using the CLI

The following provides instructions for creating vHBA-based storage pools.

**Prerequisites**

- Ensure that there are vHBAs. For more information, see [Creating vHBAs](#).

**Procedure**

1. **Define the storage pool in an XML file**
   Create a temporary XML file containing the storage pool parameters required for the new device.

   ```
   NOTE
   For information on the required parameters, refer to Parameters.
   ```

2. **Create a storage pool**
   Use the `virsh pool-define` command to create a persistent storage pool based on the XML file created in the previous step.

   ```
   # virsh pool-define ~/guest_images.xml
   Pool defined from guest_images_fs
   ```

   You can delete the XML file created in step 1 after running the `virsh pool-define` command.

1. **Verify that the pool was created**
   Use the `virsh pool-list` command to verify that the pool was created.
# virsh pool-list --all

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
<th>Autostart</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>active</td>
<td>yes</td>
</tr>
<tr>
<td>guest_images_fs</td>
<td>inactive</td>
<td>no</td>
</tr>
</tbody>
</table>

## Start the storage pool

Use the `virsh pool-start` command to mount the storage pool.

```bash
# 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.

## [Optional] Turn on autostart

By default, a storage pool defined with the `virsh` command is not set to automatically start each time `libvirtd` starts. Use the `virsh pool-autostart` command to configure the storage pool to autostart.

```bash
# virsh pool-autostart guest_images_fs
Pool guest_images_fs marked as autostarted
```

## Verify the Autostart state

Use the `virsh pool-list` command to verify the Autostart state.

```bash
# virsh pool-list --all

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
<th>Autostart</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>active</td>
<td>yes</td>
</tr>
<tr>
<td>guest_images_fs</td>
<td>inactive</td>
<td>yes</td>
</tr>
</tbody>
</table>
```

## Verify the storage pool

Verify that the storage pool was created correctly, the sizes reported are as expected, and the state is reported as `running`. Verify there is a `lost+found` directory in the target path on the file system, indicating that the device is mounted.

```bash
# 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

# mount | grep /guest_images
```
7.2.2.8.5. vHBA-based storage pool parameters

The following provides information about the required parameters for a vHBA-based storage pool and an example.

Parameters

The following table provides a list of required parameters for the XML file for a vHBA-based storage pool.

Table 7.8. vHBA-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='scsi'&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 identifier of the vHBA. The parent attribute is optional.</td>
<td>&lt;source&gt;&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>&lt;target&gt;&lt;path=target_path /&gt;&lt;/target&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 vHBA-based storage pools.

The following is an example of 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' wnnn='5001a4a93526d0a1' wwpn='5001a4ace3ee047d'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

The following is an example of 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' wnnn='5001a4a93526d0a1' wwpn='5001a4ace3ee047d'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

7.2.2.8.6. Assigning and connecting SCSI LUN-based storage volumes to virtual machines using the CLI

The following provides information on how to configure a virtual machine to use a vHBA LUN and how to reconnect to an exposed LUN after a hardware failure.
7.2.8.1. Prerequisites

- Ensure that there are one or more vHBA storage pools.

7.2.8.2. Creating vHBA-based storage pools using the CLI

The following provides instructions for creating vHBA-based storage pools.

Prerequisites

- Ensure that there are vHBAs. For more information, see Creating vHBAs.

Procedure

1. Define the storage pool in an XML file
   Create a temporary XML file containing the storage pool parameters required for the new device.

   **NOTE**
   For information on the required parameters, refer to Parameters.

2. Create a storage pool
   Use the `virsh pool-define` command to create a persistent storage pool based on the XML file created in the previous step.

   ```
   # virsh pool-define ~/guest_images.xml
   Pool defined from guest_images_fs
   ```

   You can delete the XML file created in step 1 after running the `virsh pool-define` command.

1. 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
   ```

2. 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.

3. [Optional] Turn on autostart
By default, a storage pool defined with the **virsh** command is not set to automatically start each time libvirtd starts. 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
```

4. **Verify the Autostart state**

Use the **virsh pool-list** command to verify the **Autostart** state.

```
# virsh pool-list --all

Name           State      Autostart
-----------------------------------------
default         active     yes
guest_images_fs inactive   yes
```

5. **Verify the storage pool**

Verify that the storage pool was created correctly, the sizes reported are as expected, and the state is reported as **running**. Verify there is a **lost+found** directory in the target path on the file system, indicating that the device is mounted.

```
# 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
```

```
# 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
```

### 7.2.2.9. Creating and assigning storage volumes using the CLI

The following provides information on creating storage volumes from storage pools and assigning the storage volumes to virtual machines using the CLI. The procedure is the same for all types of storage pools.

#### 7.2.2.9.1. Prerequisites

- Storage pools on the host with unallocated space

#### 7.2.2.9.2. Procedure

1. Define a storage volume in an XML file
Create a temporary XML file containing the storage volume’s parameters.

The following is a list of required storage volume parameters:

- **name** - The name of the storage volume.
- **allocation** - The total storage allocation for the storage volume.
- **capacity** - The logical capacity of the storage volume. If the volume is sparse, this value can differ from the **allocation** value.
- **target** - The path to the storage volume on the host system and optionally its permissions and label.

The following shows an example a storage volume definition XML file. In this example, the file is saved to `~/guest_volume.xml`.

```xml
<volume>
  <name>volume1</name>
  <allocation>0</allocation>
  <capacity>20</capacity>
  <target>
    <path>/var/lib/virt/images/sparse.img</path>
  </target>
</volume>
```

2. **Create and assign the storage volume**

The `virsh vol-create` and `virsh vol-create-as` commands are used to create storage volumes from most storage pools types.

The following is a list of the storage pool types that do not support the `virsh vol-create` and `virsh vol-create-as` commands and the methods to use with each of them to create storage volumes:

- **GlusterFS-based** - Use the `qemu-img` command 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.

Use the `virsh vol-create` command to create and assign the storage volume based on the XML file. Specify the virtual machine to which the storage volume will be assigned in the `virsh vol-create` command.

```
# virsh vol-create guest_images_dir ~/guest_volume.xml
Vol volume1 created
```

**NOTE**

You can delete the XML file created in step 1 after running the `virsh vol-create` command.

For GlusterFS-based, multipath-based, and RBD-based storage pools, describe the storage volume using the following XML format and add it to the domain XML:
For multipath-based storage pools, describe the storage volume using the following XML format and add it to the domain XML:

```xml
<disk type='block' device='disk'>
  <driver name='qemu' type='raw'/>
  <source dev='/dev/mapper/mpatha' />
  <target dev='sdc' bus='scsi'/>
</disk>
```

For RBD-based storage pools, describe the storage volume using the following XML format and add it to the domain XML:

```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>
```

### 7.2.3. Deleting storage for virtual machines using the CLI

The following provides information about deleting storage pools and storage volumes using the CLI.

#### 7.2.3.1. Deleting storage pools using the CLI

The following provides information on deleting storage pools.

**Prerequisites**

- To avoid negatively affecting other virtual machines that use the storage pool you want to delete, it is recommended that you stop the storage pool and release any resources being used by it.

**Procedure**

1. List the defined storage pools using the `virsh pool-list` command.

```bash
  # virsh pool-list --all
  Name       State    Autostart
  -------------------------------
  default    active    yes
  Downloads   active    yes
  RHEL8-Storage-Pool active    yes
```
2. Stop the storage pool you want to delete using the `virsh pool-destroy` command.

```bash
# virsh pool-destroy Downloads
Pool Downloads destroyed
```

3. (Optional) For some types of storage pools, you can optionally remove the directory where the storage pool resides using the `virsh pool-delete` command. Note that to remove the directory where the storage pool resides, it must be empty.

```bash
# virsh pool-delete Downloads
Pool Downloads deleted
```

4. Delete the definition of the storage pool using the `virsh pool-undefine` command.

```bash
# virsh pool-undefine Downloads
Pool Downloads has been undefined
```

5. Confirm that the storage pool was deleted.

```bash
# virsh pool-list --all
Name                 State      Autostart
-------------------------------------------
default              active     yes
RHEL8-Storage-Pool   active     yes
```

### 7.2.3.2. Deleting storage volumes using the CLI

The following provides information on deleting storage volumes using the CLI.

#### Prerequisites

- To avoid negatively affecting virtual machines that use the storage volume you want to delete, it is recommended that you release any resources using it.

#### Procedure

1. List the defined storage volumes in a storage pool using the `virsh vol-list` command. The command must specify the name or path of a storage volume.

```bash
# virsh vol-list --pool RHEL8-Storage-Pool
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
RHEL8_Volume.qcow2   /home/VirtualMachines/RHEL8_Volume.qcow2
```

2. Delete storage volumes using the `virsh vol-delete` command. The command must specify the name or path of the storage volume and the storage pool from which the storage volume is abstracted.
# virsh vol-delete --pool RHEL-Storage-Pool RHEL8_Volume.qcow2
Pool RHEL8_Volume.qcow2 deleted
CHAPTER 8. 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 share the performance of a single physical GPU.

Note, however, that assigning a physical GPU to VMs, with or without using mediated devices, makes it impossible for the host to use the GPU.

8.1. SETTING UP NVIDIA VGPU DEVICES

To set up the NVIDIA vGPU feature, you need to obtain NVIDIA vGPU drivers for your GPU device, then create mediated devices, and assign them to the intended virtual machines. For detailed instructions, see below:

Prerequisites

- Creating mediated vGPU devices is only possible on a limited set of NVIDIA GPUs. For an up-to-date list of these devices, see the NVIDIA GPU 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.

```bash
# 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. Obtain 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 the /etc/modprobe.d/. Then, add the following lines in the file:

   ```bash
   blacklist nouveau
   options nouveau modeset=0
   ```

3. Regenerate the initial ramdisk for the current kernel, then reboot.
If you need to use a prior supported kernel version with mediated devices, regenerate the initial ramdisk for all installed kernel versions.

```
# dracut --force
# reboot
```

4. Check that the `nvidia_vgpu_vfio` module has been loaded by the kernel and that the `nvidia-vgpu-mgr.service` service is running.

```
# 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)
      [...]
```

5. Write a device UUID to the `/sys/class/mdev_bus/pci_dev/mdev_supported_types/type-id/create` file, where `pci_dev` is the PCI address of the host GPU, and `type-id` is an ID of the host GPU type.

The following example shows how to create a mediated device of the `nvidia-63` vGPU type on an NVIDIA Tesla P4 card:

```
# uuidgen
30820a6f-b1a5-4503-91ca-0c10ba58692a
# echo "30820a6f-b1a5-4503-91ca-0c10ba58692a" > /sys/class/mdev_bus/0000:01:00.0/mdev_supported_types/nvidia-63/create
```

NOTE

For `type-id` values for specific GPU devices, see the Virtual GPU software documentation. Note that only Q-series NVIDIA vGPUs, such as GRID P4-2Q, are supported as mediated device GPU types on Linux VMs.

6. Add the following lines to the `<devices/>` sections in the XML configurations of guests that you want to share the vGPU resources. Use the UUID value generated by the `uuidgen` command in the previous step. Each UUID can only be assigned to one guest at a time.

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci'>
  <source>
    <address uuid='30820a6f-b1a5-4503-91ca-0c10ba58692a'/>
  </source>
</hostdev>
```

Additional resources
• For the vGPU mediated devices to work properly on the assigned VMs, NVIDIA vGPU guest software licensing needs to be set up for the VMs. For further information and instructions, see the NVIDIA virtual GPU software documentation.

8.2. REMOVING NVIDIA VGPU DEVICES

To change the configuration of assigned vGPU mediated devices, the existing devices have to be removed from the assigned VMs. For instructions to do so, see below:

Procedure

• To remove a mediated vGPU device, use the following command when the device is inactive, and replace `uuid` with the UUID of the device, for example `30820a6f-b1a5-4503-91ca-0c10ba58692a`:

```
# echo 1 > /sys/bus/mdev/devices/uuid/remove
```

Note that attempting to remove a vGPU device that is currently in use by a VM triggers the following error:

```
echo: write error: Device or resource busy
```

8.3. OBTAINING NVIDIA VGPU INFORMATION ABOUT YOUR SYSTEM

To evaluate the capabilities of the vGPU features available to you, you can obtain additional information about mediated devices on your system, such as how many mediated devices of a given type can be created.

Procedure

• Use the `virsh nodedev-list --cap mdev_types` and `virsh nodedev-dumpxml` commands. For example, the following displays available vGPU types if you are using a physical Tesla P4 card:

```
$ virsh nodedev-list --cap mdev_types
pci_0000_01_00_0
$ virsh nodedev-dumpxml pci_0000_01_00_0
<...>
<capability type='mdev_types'>
  <type id='nvidia-70'>
    <name>GRID P4-8A</name>
    <deviceAPI>vfio-pci</deviceAPI>
    <availableInstances>1</availableInstances>
  </type>
  <type id='nvidia-69'>
    <name>GRID P4-4A</name>
    <deviceAPI>vfio-pci</deviceAPI>
    <availableInstances>2</availableInstances>
  </type>
  <type id='nvidia-67'>
    <name>GRID P4-1A</name>
    <deviceAPI>vfio-pci</deviceAPI>
    <availableInstances>8</availableInstances>
  </type>
</capability>
```
8.4. REMOTE DESKTOP STREAMING SERVICES FOR NVIDIA VGPU

The following remote desktop streaming services have been successfully tested for use with the NVIDIA vGPU feature in RHEL 8 hosts:

- **HP-RGS** - Note that it is currently not possible to use HP-RGS with RHEL 8 VMs.

- **Mechdyne TGX** - Note that it is currently not possible to use Mechdyne TGX with Windows Server 2016 VMs.

- **NICE DCV** - When using this streaming service, Red Hat recommends using fixed resolution settings, as using dynamic resolution in some cases results in a black screen. In addition, it is currently not possible to use NICE DCV with RHEL 8 VMs.

8.5. RELATED INFORMATION

- For further information on using NVIDIA vGPU on RHEL with KVM, see the NVIDIA GPU Software Documentation.
CHAPTER 9. OPTIMIZING VIRTUAL MACHINE PERFORMANCE IN RHEL 8

Virtual machines (VMs) always experience some degree of performance deterioration in comparison to the host. The following sections describe the reasons why that is and provide instructions how to minimize the performance impact of virtualization, so that your hardware infrastructure resources can be used as efficiently as possible.

9.1. 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 as the host.

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 and 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.

Nevertheless, a variety of features are available that 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.

**NOTE**

Tuning VM performance can have adverse effects on other virtualization functions. For example, it can make migrating the modified VM more difficult.

9.2. 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.
Red Hat recommends using the following profiles when using virtualization in RHEL 8:

- For RHEL 8 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 8 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 must be 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.
   
   # tuned-adm profile virtual-host
   
   - On a RHEL guest operating system, use the `virtual-guest` profile.
   
   # tuned-adm profile virtual-guest
   ```

**Additional resources**

- For more information about `tuned` and `tuned` profiles, see `Monitoring and managing system status and performance`.

**9.3. OPTIMIZING VIRTUAL MACHINE I/O PERFORMANCE**

The input and output (I/O) capabilities of a virtual machine (VM) can create a significant limitation to the VM’s overall efficiency. To address this, you can optimize a VM’s I/O by configuring block I/O parameters.
9.3.1. 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 blktune virtual_machine

   ```xml
   <domain>
     ...
     <blktune>
       <weight>800</weight>
       <device>
         <path>/dev/sda</path>
         <weight>1000</weight>
       </device>
       <device>
         <path>/dev/sdb</path>
         <weight>500</weight>
       </device>
     </blktune>
     ...
   </domain>
   ```

2. Edit the I/O weight of a specified device:

   ```
   # virsh blktune 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 blktune liftrul --device-weights /dev/sda, 500
   ```

9.3.2. 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 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 for a list of disk device names on a specified VM.

```bash
# virsh domblklist rollin-coal
Target   Source
----------
  vda      /var/lib/libvirt/images/rollin-coal.qcow2
  sda      -
  sdb      /home/horridly-demanding-processes.iso
```

2. Set I/O limits for a block device attached to a VM using the `virsh blkdeviotune` command:

```bash
# virsh blkdeviotune VM-name device --parameter limit
```

For example, to throttle the `sdb` device on the `rollin-coal` VM to 1000 I/O operations per second and 50 MB per second throughput:

```bash
# virsh blkdeviotune rollin-coal sdb --total-iops-sec 1000 --total-bytes-sec 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.

### 9.3.3. 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 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:

```xml
<controller type='scsi' index='0' model='virtio-scsi'>
  <driver queues='N'/>
</controller>
```

### 9.4. OPTIMIZING VIRTUAL MACHINE CPU PERFORMANCE

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. 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:

```bash
# virt-xml testguest1 --edit --cpu host-model
```

2. 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 to CPU and memory processes on 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 \[xref:sample-vcpu-performance-tuning-scenario_optimizing-virtual-machine-cpu-performance\].

### 9.4.1. 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 8 host.

#### Prerequisites

- The host must be 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 possible also 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
```

Additional resources

- Note that for best performance results, it is recommended to use all of the manual tuning methods listed above. For an example of such a configuration, see Section 9.4.2, “Sample vCPU performance tuning scenario”.

- To see the current NUMA configuration of your system, you can use the `numastat` utility. For details on using `numastat`, see Section 9.6, “Virtual machine performance monitoring tools”.

9.4.2. 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:

```
# 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:

```
# virsh capabilities
```

The output should include a section that looks similar to the following:

```
<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>
    </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>
  <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>
</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:
   
   ```bash
   #!/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()
   ```
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:

- Set the VM to use 8 static vCPUs. Use the `<vcpu/>` element to do this.

- Pin each of the vCPU threads to the corresponding host CPU threads that it mirrors in topology. To do so, use the `<vcpupin/>` elements in the `<cputune/>` section. Note that, as shown by `virsh capabilities` 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 Additional Resources section below.

- 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.

- Ensure the CPU mode is set to `host-passthrough`, and that the CPU uses cache in `passthrough` mode.

The resulting XML configuration should include a section similar to the following:

```
[...]  
  <memoryBacking>  
    <hugepages>  
    <page size='1' unit='GiB'/>  
    </hugepages>  
  </memoryBacking>  
  <vcpu placement='static'>8</vcpu>  
  <cputune>  
    <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'/>  
  </cputune>  
```
Test the performance of the VM using the applicable tools and utilities to evaluate the impact of the VM’s optimization.

### Additional resources

- The following tables illustrate the connections between the vCPUs and the host CPUs they should be pinned to:

#### Table 9.1. 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></td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td>3</td>
<td></td>
<td>4</td>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Sockets</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMA nodes</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Table 9.2. 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></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sockets</td>
<td>0</td>
<td></td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></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:

```
<vcpupin vcpu='0' cpuset='1'/>
<vcpupin vcpu='1' cpuset='4'/>
<vcpupin vcpu='2' cpuset='2'/>
<vcpupin vcpu='3' cpuset='5'/>
```

### 9.5. OPTIMIZING VIRTUAL MACHINE NETWORK PERFORMANCE

Due to the virtual nature of a VM’s network interface card (NIC), the VM loses a portion of the host network bandwidth it is allocated, 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:

```
# 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:
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 to 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:

```xml
<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.

Set up vhost zero-copy transmit

If using a network with large packet size, enable the vhost zero-copy transmit feature. Note that this feature only improves the performance when transmitting large packets between a guest network and an external network. It does not affect performance for guest-to-guest and guest-to-host workloads. In addition, it is likely to have a negative impact on the performance of small packet workloads.

Also, enabling zero-copy transmit can cause head-of-line blocking of packets, which may create a potential security risk.

To enable vhost zero-copy transmit:

1. On the host, disable the vhost-net kernel module:

   ```
   # modprobe -r vhost_net
   ```

2. Re-enable the vhost-net module with the zero-copy parameter turned on:

   ```
   # modprobe vhost-net experimental_zcopytx=1
   ```

3. Check whether zero-copy transmit was enabled successfully:

   ```
   # cat /sys/module/vhost_net/parameters/experimental_zcopytx
   1
   ```

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:

```
# ethtool -C tap0 rx-frames 128
```

Additional resources

- For additional information on virtual network connection types and tips for usage, see Chapter 11, *Understanding virtual networking*.
9.6. VIRTUAL MACHINE PERFORMANCE MONITORING TOOLS

To recognize what consumes the most VM resources and which aspect of VM performance needs optimization, both general and VM-specific performance diagnostic tools 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 8 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 8 host. To do so:

1. On the host, install the `perf` package:
   ```bash
   # yum install perf
   ```

2. Use the `perf kvm stat` command 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 powering down some of your active VMs, removing vCPUs from these VMs, or tuning the performance of the vCPUs.
   ```bash
   # perf kvm stat report
   ```

Analyze events for all VMs, all VCPUs:

```
+---------------------------------+-----------------+-----------------+-----------------+-----------------+----------------+-----------------+-----------------+
 | VM-EXIT | Samples | Samples% | Time% | Min Time | Max Time | Avg time |
 +---------------------------------+-----------------+-----------------+-----------------+-----------------+----------------+-----------------+
```
EXTERNAL_INTERRUPT 365634 31.59% 18.04% 0.42us 58780.59us 204.08us (+- 0.99%)
MSR_WRITE 293428 25.35% 0.13% 0.59us 17873.02us 1.80us (+- 4.63%)
PREEMPTION_TIMER 276162 23.86% 0.23% 0.51us 21396.03us 3.38us (+- 5.19%)
PAUSE_INSTRUCTION 189375 16.36% 11.75% 0.72us 29655.25us 256.77us (+- 0.70%)
HLT 20440 1.77% 69.83% 0.62us 79319.41us 14134.56us (+- 0.79%)
VMCALL 12426 1.07% 0.03% 1.02us 5416.25us 8.77us (+- 7.36%)
EXCEPTION_NMI 27 0.00% 0.00% 0.69us 1.34us 0.98us (+- 3.50%)
EPT_MISCONFIG 5 0.00% 0.00% 5.15us 10.85us 7.88us (+- 11.67%)

Total Samples:1157497, Total events handled time:413728274.66us.

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:

```bash
# 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</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</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</td>
<td>62</td>
<td>432</td>
<td>1661</td>
<td>506</td>
<td>4851</td>
<td>136</td>
<td>2</td>
<td>445</td>
<td>8116</td>
</tr>
<tr>
<td>53773</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>
</tbody>
</table>

Total 1769 463 2024 7462 10037 2672 169 7837 32434

In contrast, the following shows memory being provided to each VM by a single node, which is significantly more efficient.

```bash
# 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</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</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></td>
</tr>
<tr>
<td>53773</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8110</td>
<td>8118</td>
<td></td>
</tr>
</tbody>
</table>

146
<table>
<thead>
<tr>
<th>59065 (qemu-kvm)</th>
<th>0</th>
<th>0</th>
<th>8050</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>8051</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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>

CHAPTER 9. OPTIMIZING VIRTUAL MACHINE PERFORMANCE IN RHEL 8
CHAPTER 10. SHARING FILES BETWEEN THE HOST AND ITS VIRTUAL MACHINES

Sharing data between your host system and the virtual machines (VMs) it runs is frequently required. To do so quickly and efficiently, you can set up NFS or Samba file shares on your system.

10.1. SHARING FILES BETWEEN THE HOST AND LINUX VIRTUAL MACHINES

For efficient file sharing between your host system and the Linux VMs it is connected to, you can export an NFS share that your VMs can mount and access.

Prerequisites

- The `nfs-utils` package must be 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 set up the `macvlan` feature on the host first. To do so:

1. Create a network device file, for example called `/etc/systemd/network/vm-macvlan.netdev` in the host's `/etc/systemd/network/` directory.

```bash
# touch /etc/systemd/network/vm-macvlan.netdev
```

2. Edit the network device file to have the following content. If appropriate, replace `vm-macvlan` with the name you chose for your network device.

```bash
[NetDev]
Name=vm-macvlan
Kind=macvlan

[MACVLAN]
Mode=bridge
```

3. Create a network configuration file for your macvlan network device, for example `/etc/systemd/network/vm-macvlan.network`.

```bash
# touch /etc/systemd/network/vm-macvlan.network
```

4. Edit the network configuration file to have the following content. If appropriate, replace `vm-macvlan` with the name you chose for your network device.

```bash
[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`:

```
# touch /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 for the physical network to be a part of the macvlan interface, in this case `vm-macvlan`:

```
[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 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`.

```
# 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.
# exportfs -a

d. Ensure the NFS process is started:

```
# systemctl start nfs
```

e. Obtain the IP address of the host system. This will be used for mounting the shared directory on the VMs later.

```
# ip addr
[...]
5: virbr0: [BROADCAST,MULTICAST,UP,LOWER_UP] mtu 1500 qdisc noqueue state UP
    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 which the VMs you want to share files with use for connection to the host. 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
```

   c. 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
```

### 10.2. SHARING FILES BETWEEN THE HOST AND WINDOWS VIRTUAL MACHINES

For efficient file sharing between your host system and the Windows VMs it is connected to, you can prepare a Samba server that your VMs can access.

**Prerequisites**

- The `samba` packages are installed on your host. If they are not:

```
# yum 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 set up the macvlan feature on the host first. To do so:

1. Create a network device file, for example called `vm-macvlan.netdev` in the host’s `/etc/systemd/network/` directory.

   ```bash
   # touch /etc/systemd/network/vm-macvlan.netdev
   ```

2. Edit the network device file to have the following content. If appropriate, replace `vm-macvlan` with the name you chose for your network device.

   ```
   [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
   # touch /etc/systemd/network/vm-macvlan.network
   ```

4. Edit the network configuration file to have the following content. If appropriate, replace `vm-macvlan` with the name you chose for your network device.

   ```
   [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
   # touch /etc/systemd/network/enp4s0.network
   ```

   If you are unsure what interface 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 for the physical network to be a part of the macvlan interface, in this case `vm-macvlan`:

   ```
   [Match]
   Name=enp4s0
   
   [Network]
   MACVLAN=vm-macvlan
   ```

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.

   ```
   # firewall-cmd --permanent --zone=public --add-service=samba
   success
   # firewall-cmd --reload
   success
   ```

   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**.

   ![Add a network location menu](image1)

   c. In the **Add Network Location** wizard that opens, select “Choose a custom network location” and click **Next**.

   ![Add Network Location wizard](image2)

   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**.

   ![Add Network Location wizard](image3)

   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**.

   ![Add Network Location wizard](image4)
f. If accessing the network location was successful, you can now click **Finish** and open the shared directory.
CHAPTER 11. UNDERSTANDING VIRTUAL NETWORKING

The connection of virtual machines to other devices and locations on a network has to be facilitated by the host hardware.

Virtual networking uses the concept of a virtual network switch. A virtual network switch is a software construct that operates on a host physical machine server. Virtual machines (VMs) connect to the network through the virtual network switch.

The following figure shows a virtual network switch connecting two virtual machines to the network:

From the perspective of a guest operating system, a virtual network connection is the same as a physical network connection.

Host physical machine servers view virtual network switches as network interfaces. When the libvirtd daemon (libvirtd) is first installed and started, the default network interface representing the virtual network switch is `virbr0`.

This interface can be viewed with the `ip` command like any other network interface.

```
$ ip addr show virbr0
3: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
   link/ether 1b:c4:94:cf:fd:17 brd ff:ff:ff:ff:ff:ff
   inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
```

By default, all VMs on a single host are connected to the same libvirt virtual network, named `default`. VMs on this network can make the following connections:

- **With each other and with the virtualization host**
  Both inbound and outbound traffic is possible, but is affected by the firewalls in the guest operating system’s network stack and by libvirt network filtering rules attached to the guest interface.

- **With other hosts on the network beyond the virtualization host**
  Only outbound traffic is possible and is affected by Network Address Translation (NAT) rules, as well as the host system’s firewall.

For basic outbound-only network access from virtual machines, no additional network setup is usually needed, because the default network is installed along with the libvirt package, and automatically started when the libvirtd service is started.
If more advanced functionality is needed, additional networks can be created and configured using `virsh`, and the VM’s XML configuration file can be edited to use one of these new networks.

For more information on the default configuration, see Section 11.7, “Virtual networking default configuration”.

If needed, guest interfaces can instead be set to one of the following modes:

- Routed mode
- Bridged mode
- Isolated mode
- Open mode

The virtual network uses network address translation (NAT) to assign IP address ranges to virtual networks and `dnsmasq` to automatically assign IP addresses to virtual machine network interface cards (NICs) and connect to a domain name service (DNS).

The following features are available for virtual networking:

- Network address translation (NAT)
- DNS and DHCP

## 11.1. 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, all of the virtual machines are in their own subnet, routed through a virtual switch. This enables incoming connections, but requires extra routing-table entries for systems on the external network. Routed mode operates at Layer 3 of the OSI networking model.

**Common topologies**

The following are two common topologies in which routed mode is used:
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).

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 is for the virtual machines to connect through. Each virtual machine has its own public IP address, but the host machines use private IP addresses so that management of the virtual machines can only be performed by internal administrators.

11.2. VIRTUAL NETWORKING IN BRIDGED MODE
When using Bridged mode, virtual machines (VMs) are connected to a bridge device that is also
connected directly to a physical ethernet device connected to the local ethernet. 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.

A VM in bridged mode has to connect to an existing Linux bridge on the host, and therefore requires a network bridge to be created on the host interface. In contrast, other VM networking modes automatically create and connect to the `virbr0` virtual bridge.

All of the VMs appear within the same subnet as the host machine. All other physical machines on the same physical network are aware of the VMs, and can access them. Bridging operates on Layer 2 of the OSI networking model.

It is possible to use multiple physical interfaces on the hypervisor by joining them together with a bond. The bond is then added to a bridge and then VMs are added onto the bridge as well. However, the bonding driver has several modes of operation, and only a few of these modes work with a bridge where VMs are in use.

**WARNING**

When using bridged mode, the only bonding modes that should be used with a VM are Mode 1, Mode 2, and Mode 4. 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.

**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 transparent 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 services within an existing broadcast domain, such as DHCP.

- Connecting VMs to an existing network where VLANs are used.

Additional resources

- For a detailed explanation of bridge_opts parameters, used to configure bridged networking mode, see the Red Hat Virtualization Administration Guide.

11.3. VIRTUAL NETWORKING IN ISOLATED MODE

When using Isolated mode, VMs 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.

11.4. VIRTUAL NETWORKING NETWORK ADDRESS TRANSLATION

By default, virtual network switches operate in 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 IP address for communication to any external network. Computers external to the host cannot communicate to the VMs inside when the virtual network switch is operating in NAT mode.
WARNING

Virtual network switches use NAT configured by iptables rules. Editing these rules while the switch is running is not recommended, because incorrect rules may result in the switch being unable to communicate.

If the switch is not running, you can set the public IP range for forward mode NAT in order to create a port masquerading range by running:

```
# iptables -j SNAT --to-source [start]-[end]
```

11.5. VIRTUAL NETWORKING IN OPEN MODE

When using Open mode for networking, `libvirt` does not generate any `iptables` rules for the network. As a result, `iptables` rules added outside the scope of `libvirt` are not overwritten, and the user can therefore manually manage `iptables` rules.

11.6. VIRTUAL NETWORKING DNS AND DHCP

The `libvirt` package includes `dnsmasq` to provide a Dynamic Host Configuration Protocol (DHCP) server and a Domain Name System (DNS) forwarder for virtual networks.

The `dnsmasq` DHCP service can assign a pool of addresses to a virtual network switch. IP information can be assigned to virtual machines via DHCP.

`dnsmasq` accepts DNS queries from virtual machines on the virtual network and forwards them to a real DNS server.

An instance of `dnsmasq` is automatically configured and started by `libvirt` for each virtual network switch that needs it.
11.7. VIRTUAL NETWORKING DEFAULT CONFIGURATION

When the libvirtd daemon (libvirtd) is first installed, it contains an initial virtual network switch configuration in NAT mode. This configuration is used so that installed VMs can communicate to the external network through the host machine. The following figure shows the default configuration for libvirtd:

![Virtual Network Configuration Diagram]

**NOTE**

A virtual network can be restricted to a specific physical interface. This may be useful on a physical system that has several interfaces (for example, eth0, eth1, and eth2). This is only useful in routed and NAT modes, and can be defined in the `dev=<interface>` option, or in the RHEL 8 web console when creating a new virtual network.
CHAPTER 12. SECURING VIRTUAL MACHINES IN RHEL 8

As an administrator of a 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 8 host and provides a list of methods to increase the security of your VMs.

12.1. 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 12.1. 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 8 provides for VM security are always active and do not have to be enabled or configured. For details, see Section 12.4, “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 Section 12.2, “Best practices for securing virtual machines”.

### 12.2. 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 8, see Configuring and managing security in RHEL 8 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 Configuring and managing security guide for instructions to activate Enforcing mode.
NOTE

SE Linux Enforcing mode also enables the sVirt RHEL 8 feature. This is a set of specialized SE Linux 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 Section 12.3, “Creating a SecureBoot virtual machine”.

- Do not use qemu-* commands, such as qemu-img.
  QEMU is an essential component of the virtualization architecture in RHEL 8, 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, it is highly recommended to interact with QEMU using libvirt utilities, such as virsh, virt-install, and virt-xml, as these orchestrate QEMU according to the best practices.

Additional resources

- For detailed information on modifying your virtualization booleans, see Section 12.5, “Virtualization booleans in RHEL 8”.

12.3. CREATING A SECUREBOOT VIRTUAL MACHINE

The following provides instructions on creating a Linux virtual machine (VM) that uses the SecureBoot feature, which ensures that your VM is running a cryptographically signed OS. If the guest OS of a VM has been altered by malware, 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 installed:

  # yum install edk2-ovmf

- An operating system (OS) installation source, which can be one of the following, and can be available locally or on a network:
  - A physical installation medium, such as a DVD
  - An ISO image of an installation medium
  - A disk image of an existing VM installation
- Optionally, a Kickstart file can also be provided for faster and easier configuration of the installation.

Procedure

1. Use the virt-install command to create a VM as detailed in Section 2.2.2, “Creating virtual machines using the command-line interface”. For the --boot option, use the
uefi_nvram_template=/usr/share/OVMF/OVMF_VARS.secboot.fd

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 rhel8.0 --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-8.0-installation.iso
```

2. Follow the OS installation procedure according to the instructions on the screen.

3. 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.

4. Verify that SecureBoot is enabled by using the mokutil --sb-state command:

```
# mokutil --sb-state
SecureBoot enabled
```

12.4. AUTOMATIC FEATURES FOR VIRTUAL MACHINE SECURITY

In addition to manual means of improving the security of your virtual machines listed in Section 12.2, “Best practices for securing virtual machines”, a number of security features are provided by the libvirt software suite and automatically enabled when using virtualization in RHEL 8. These include:

**System and user sessions**

To access all the available utilities for virtual machine management in RHEL 8, you need to use the system session of libvirt. 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 user session of libvirt, which has to respect the access rights of the local user when accessing resources. For example, in the user session, you cannot detect or access VMs created in the system session or by other users. Also, available VM networking configuration options are significantly limited.

**NOTE**

The RHEL 8 documentation assumes you have libvirt system session 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.

12.5. VIRTUALIZATION BOOLEANS IN RHEL 8
For fine-grained configuration of virtual machines security on a RHEL 8 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:

```
$ 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 8 and what they do when enabled:

Table 12.1. 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>SELinux Boolean</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</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>
CHAPTER 13. INSTALLING AND MANAGING WINDOWS VIRTUAL MACHINES ON RHEL 8

The following provides information on installing Windows VMs on RHEL 8 machines, optimizing Windows VMs on RHEL 8 machines, and installing and configuring drivers on Windows VMs on RHEL 8 machines.

13.1. INSTALLING WINDOWS VIRTUAL MACHINES ON RHEL 8

The following provides information on how to create a fully-virtualized Windows machine on RHEL 8, launch the Windows’ installer inside the virtual machine (VM), and access the graphical installer.

You can create a VM and install it using the `virt-install` command or the RHEL 8 web console.

Prerequisites

- If the VM needs to use KVM Windows paravirtualized (virtio) drivers, remember to install the `virtio` drivers during or after installing the Windows operating system on the VM. For more information, see [Installing KVM paravirtualized drivers for RHEL 8 virtual machines](#).

Procedure

Installing a Windows operating system involves two steps:

1. Create the VM. For instructions, see [Creating virtual machines](#).

2. Install the Windows operating system on the VM.
   For information on how to install a Windows operating system, refer to the relevant Microsoft installation documentation.

Related information

- [Creating virtual machines](#)

13.2. OPTIMIZING WINDOWS VIRTUAL MACHINES ON RHEL 8

You can optimize Windows virtual machines on RHEL 8 by doing the following:

- Enabling Hyper-V enlightenments. For more information, see [Enabling Hyper-V enlightenments](#).

- Using paravirtualized drivers. For more information, see [Installing KVM paravirtualized drivers for RHEL 8 virtual machines](#).

13.2.1. 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 on RHEL 8.

The following provides information about the supported Hyper-V enlightenments and how to enable them.

13.2.1.1. Enabling Hyper-V enlightenments on a Windows virtual machine in RHEL 8

Hyper-V enlightenments provide better performance in a Windows virtual machine (VM).

Procedure
1. Edit the XML configuration of the VM, adding the Hyper-V enlightenments.

```
# virt-xml $VMNAME --edit --features
hyperv_relaxed=on,hyperv_vapic=on,hyperv_spinlocks=on,hyperv_spinlocks_retries=8191,hyperv_vpindex=on,hyperv_runtime=on,hyperv_sync=on,hyperv_stimer=on,hyperv_frequencies=on
```

```
# virt-xml $VMNAME --edit --clock hypervclock_present=yes
```

**NOTE**

You can also use the `virsh edit` command to add the snippets to the VM’s XML configuration.

This results in the following snippets being added to the VM’s XML configuration:

```
<hyperv>
  <relaxed state='on'/>
  <vapic state='on'/>
  <spinlocks state='on' retries='8191'/>
  <vpindex state='on'/>
  <runtime state='on'/>
  <sync state='on'/>
  <stimer state='on'/>
  <frequencies state='on'/>
</hyperv>

<clock offset='localtime'>
  <timer name='hypervclock' present='yes'/>
</clock>
```

2. Restart the VM.

   The Hyper-V enlightenments are enabled on the VM.

**13.2.1.2. Supported Hyper-V enlightenments**

The following is a list of Hyper-V enlightenments with a description of the functionality of each enlightenment:

**Hyper-V enlightenments**

- **clock**
  
  Defines the virtual time source and offset to use with the Windows virtual machine.

- **frequencies**
  
  Enables Hyper-V frequency Machine Specific Registers (MSRs).

- **relaxed**
  
  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.

- **runtime**
  
  Specifies the virtual processor’s run time.
spinlock

- Used by a virtual machine’s operating system to notify the hypervisor that the calling virtual processor is attempting to acquire a resource that is potentially held by another virtual processor within the same partition.

- Used by the hypervisor 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 the hypervisor.

sync

Together with stimer, activates the synthetic timer. Windows 8 uses this feature in periodic mode.

vapic

Provides accelerated MSR access to high-usage memory-mapped Advanced Programmable Interrupt Controller (APIC) registers.

vpindex

Specifies the virtual processor’s index.

13.2.2. Installing KVM paravirtualized drivers for RHEL 8 virtual machines

Paravirtualized drivers enhance the performance of virtual machines (VMs), decreasing I/O latency and increasing throughput to near bare-metal levels. It is recommended that you use paravirtualized drivers for fully virtualized VMs running 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 included in the virtio-win package. The virtio-win package supports block (storage) devices, network interface controllers, video controllers, memory ballooning device, paravirtual serial port device, entropy source device, paravirtual panic device, and a small set of emulated devices.

NOTE

For additional information about emulated, virtio, and assigned devices, refer to Guest Virtual Machine Device Configuration.

Using KVM virtio drivers, the following Microsoft Windows versions are expected to run similarly to bare-metal-based systems:

- Windows Server versions: See Certified guest operating systems for Red Hat Enterprise Linux with KVM
- 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)
NOTE

Network connectivity issues sometimes arise when attempting to use older virtio drivers with newer versions of QEMU. Keeping the drivers up to date is therefore recommended.

To install virtio drivers:

1. Prepare the install media on the host machine. For more information, see Preparing virtio installation media on a host.

2. Do one of the following:
   - Install the virtio drivers on an existing Windows VM.
   - Install the virtio drivers while installing Windows on the VM.

For more information, see Installing KVM drivers on a Windows virtual machine.

13.2.2.1. Preparing KVM driver installation media on a host machine

The installation media for KVM drivers on a Windows virtual machine must be prepared on the host machine. This is done by installing the virtio-win package on the host machine. The installation creates the following types of media that can be used for installing the virtio drivers on a Windows virtual machine:

- An ISO file that can be accessed using a virtualized CD-ROM device or a USB drive.
- A directory with the driver installation files that can be accessed from the Windows virtual machine using a local or shared network drive.

Prerequisites

- Ensure that virtualization is enabled in RHEL.

Procedure

1. Download the drivers
   a. Browse to Download Red Hat Enterprise Linux.
   b. Select the product variant from the menu:[Product Variant] list.
   c. Select the version from the menu:[Version] list.
   d. Select the architecture from the menu:[Architecture] list.
   e. In the Packages for RHEL section, search for virtio-win.
   f. Click Download Latest. The RPM file downloads.

2. Install the virtio-win package from the download directory. For example:

   $ sudo yum install ~/Downloads/virtio-win-x.x.x-x.xxx.noarch.rpm
   Installed:
   virtio-win-x.x.x-x.xxx.noarch
The virtio-win driver files are prepared in the file: `/usr/share/virtio` directory. These include ISO files and a drivers directory with the driver files in directories, one for each architecture and Windows version.

13.2.2.2. Installing KVM drivers on a Windows guest

The KVM virtio drivers can be loaded on the virtual machine during the Windows installation or installed after Windows is installed on the virtual machine.

Prerequisites

- The installation media for KVM drivers on a Windows virtual machine is prepared on the host machine. For more information, see Preparing KVM driver installation media on a host machine.

13.2.2.2.1. Installing KVM drivers on an existing Windows virtual machine

You can install KVM virtio drivers on an existing Windows virtual machine (VM) from any storage medium.

Procedure

1. Add the storage medium to the Windows VM.
   For more information, see Chapter 7, Managing storage for virtual machines.

2. Reboot the VM.

3. Open the Windows device manager.
   For information on opening Windows Device Manager, refer to the Windows documentation.

4. View the available devices by clicking the arrow to the left of Other devices.

5. Locate the appropriate device
   There are up to eight devices available: the memory ballooning device, the paravirtual serial port device, the network device, the block device, the SCSI block device, the entropy source device, the emulated PCI serial port device, and the paravirtual panic device. Additionally, the video device is available in the Display adapters group.

   - Balloon, the balloon driver, applies to PCI Device in the Other devices group.

   - vioserial, the paravirtual serial driver, applies to PCI Simple Communications Controller in the Other devices group.
- NetKVM, the network driver, applies to Ethernet Controller in the Other devices group. This driver is only available if a virtio NIC is configured. Configurable parameters for this driver are documented in Appendix E, NetKVM Driver Parameters.

- viostor, the block driver, applies to SCSI Controller in the Other devices group.

- viocsci, the SCSI block driver, applies to SCSI Controller in the Other devices group.

- viorng, the entropy source driver, applies to PCI Device in the Other devices group.

- qemupciserial, the emulated PCI serial driver, applies to PCI Serial Port in the Other devices group.

- qxl, the paravirtual video driver, applied to Microsoft Basic Display Adapter in the Display adapters group.

- pvpanic, the paravirtual panic driver, applies to Unknown device in the Other devices group.

**NOTE**

Both the memory balloon and entropy source devices are initially represented by PCI Device in the Other devices group. Similarly, both block devices are represented by SCSI Controller. The following procedure is used to tell them apart. ---

Right-click the device that you wish to identify and select the [btn]::`Details` tab. Select the Hardware Ids property in the list. Match the hardware ID with one of the IDs below:

- **VEN_1AF4&DEV_1002** or **VEN_1AF4&DEV_1045**, the balloon device.
- **VEN_1AF4&DEV_1003** or **VEN_1AF4&DEV_1043**, the paravirtual serial port device.
- **VEN_1AF4&DEV_1000** or **VEN_1AF4&DEV_1041**, the network device.
- **VEN_1AF4&DEV_1001** or **VEN_1AF4&DEV_1042**, the block device.
- **VEN_1AF4&DEV_1004** or **VEN_1AF4&DEV_1048**, the SCSI block device.
- **VEN_1AF4&DEV_1005** or **VEN_1AF4&DEV_1044**, the entropy source device.
- **VEN_1B36&DEV_0002**, the emulated PCI serial driver.
- **VEN_1B36&DEV_0100**, the video device.
- **VEN_QEMU&DEV_0001**, the guest panic device. ---

6. Right-click the device whose driver you want to update, and select **Update Driver Software** from the pop-up menu.
The Update Driver Software wizard starts.

The first page of the driver update wizard asks how you want to search for driver software.

7. Click **Browse my computer for driver software**.

8. Click **Browse**.
   A file browser opens.

9. Browse to the location of the driver.
   A separate driver is provided for each combination of operating systems and architectures. The drivers are arranged hierarchically according to their driver type, the operating system, and the architecture on which they will be installed: `driver_type/os/arch`.

   For example, the Balloon driver for a Windows 10 operating system with an x86 (32-bit) architecture, resides in the `Balloon/w10/x86` directory.

   Once you have navigated to the correct location, click **OK**.

10. Click **Next**.
The following screen is displayed while the driver installs:

![Driver Installation Screen](image1)

When the installation is complete, the following screen is displayed:

![Driver Update Screen](image2)

11. Click **Close**.
12. Reboot the VM to complete the driver installation.

NOTE
After the driver is successfully installed, the device no longer appears in the Other devices group. Find the device in the Multifunction adapters group (lx QEMU PCI Serial Card), Network adapters group (Red Hat VirtIO Ethernet Adapter), Storage controllers group (Red Hat VirtIO SCSI controller and Red Hat VirtIO SCSI pass-through controller), and System devices group (QEMU PPV Panic Device, VirtIO Balloon Driver, VirtIO RNG Device, and VirtIO Serial Driver).

13.2.2.2.2. Installing KVM drivers while installing Windows on a virtual machine

You can install the **virtio** drivers while installing Windows on a virtual machine (VM). This enables a Windows VM to use the **virtio** drivers for the default storage device.

Procedure

1. Create the VM using the `virt-install` utility.  
   For more information, see Section 2.2.2, “Creating virtual machines using the command-line interface”.

   **IMPORTANT**
   Do not start the VM.

   Add the following parameters to the `virt-install` command:

   - The required parameter to add the storage medium with the KVM **virtio** drivers to the VM.  
     For example, if the **virtio** drivers are on a virtual disk, add the following parameter:

     ```
     --disk path=/usr/share/virtio-win/virtio-win_*arch*qcow2,device=disk,bus=virtio
     ```

   **IMPORTANT**
   If the device you want to add is a disk (that is, not a CD-ROM), add the `bus=virtio` option to the end of the `--disk` parameter:

     ```
     --disk path=/usr/share/virtio-win/virtio-win_*arch*.qcow2,device=disk,bus=virtio
     ```

   - The required parameter for the Windows version you will install on the VM.  
     For example, for Windows 10:

     ```
     --os-variant win10
     ```

     For a list of Windows versions and the appropriate option, run the following command:

     ```
     # osinfo-query os
     ```

2. Start the VM and install the Windows operating system.  
   For more information, see Starting virtual machines.

3. Install the **virtio** drivers.
For information on installing these drivers, refer to the documentation for the version of Windows that you are installing.

13.3. INSTALLING AND CONFIGURING DRIVERS AND UTILITIES FOR WINDOWS VIRTUAL MACHINES ON RHEL 8

The following provides information about installing and configuring Windows virtual machines on RHEL 8:

- KVM paravirtualized drivers
- The NetKVM drivers
- The libguestfs library and tools

13.3.1. Installing KVM paravirtualized drivers for RHEL 8 virtual machines

Paravirtualized drivers enhance the performance of virtual machines (VMs), decreasing I/O latency and increasing throughput to near bare-metal levels. It is recommended that you use paravirtualized drivers for fully virtualized VMs running 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 included in the **virtio-win** package. The **virtio-win** package supports block (storage) devices, network interface controllers, video controllers, memory ballooning device, paravirtual serial port device, entropy source device, paravirtual panic device, and a small set of emulated devices.

**NOTE**

For additional information about emulated, **virtio**, and assigned devices, refer to Guest Virtual Machine Device Configuration.

Using KVM virtio drivers, the following Microsoft Windows versions are expected to run similarly to bare-metal-based systems:

- Windows Server versions: See Certified guest operating systems for Red Hat Enterprise Linux with KVM
- 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)

**NOTE**

Network connectivity issues sometimes arise when attempting to use older virtio drivers with newer versions of QEMU. Keeping the drivers up to date is therefore recommended.

To install **virtio** drivers:
1. Prepare the install media on the host machine. For more information, see Preparing virtio installation media on a host.

2. Do one of the following:
   - Install the virtio drivers on an existing Windows VM.
   - Install the virtio drivers while installing Windows on the VM.

For more information, see Installing KVM drivers on a Windows virtual machine.

13.3.1.1. Preparing KVM driver installation media on a host machine

The installation media for KVM drivers on a Windows virtual machine must be prepared on the host machine. This is done by installing the virtio-win package on the host machine. The installation creates the following types of media that can be used for installing the virtio drivers on a Windows virtual machine:

- An ISO file that can be accessed using a virtualized CD-ROM device or a USB drive.
- A directory with the driver installation files that can be accessed from the Windows virtual machine using a local or shared network drive.

Prerequisites

- Ensure that virtualization is enabled in RHEL.

Procedure

1. Download the drivers
   a. Browse to Download Red Hat Enterprise Linux.
   b. Select the product variant from the menu:[Product Variant] list.
   c. Select the version from the menu:[Version] list.
   d. Select the architecture from the menu:[Architecture] list.
   e. In the Packages for RHEL section, search for virtio-win.
   f. Click Download Latest.
      The RPM file downloads.

2. Install the virtio-win package from the download directory. For example:

   ```
   $ sudo yum install ~/Downloads/virtio-win-x.x.x-xxx.noarch.rpm
   Installed:
   virtio-win-x.x.x-xxx.noarch
   ```

   The virtio-win driver files are prepared in the [filename]:`/usr/share/virtio` directory. These include ISO files and a drivers directory with the driver files in directories, one for each architecture and Windows version.

13.3.1.2. Installing KVM drivers on a Windows guest
The KVM virtio drivers can be loaded on the virtual machine during the Windows installation or installed after Windows is installed on the virtual machine.

Prerequisites

- The installation media for KVM drivers on a Windows virtual machine is prepared on the host machine. For more information, see [Preparing KVM driver installation media on a host machine](#).

13.3.1.2.1. Installing KVM drivers on an existing Windows virtual machine

You can install KVM **virtio** drivers on an existing Windows virtual machine (VM) from any storage medium.

**Procedure**

1. Add the storage medium to the Windows VM.
   For more information, see [Chapter 7, Managing storage for virtual machines](#).

2. Reboot the VM.

3. Open the Windows device manager.
   For information on opening Windows Device Manager, refer to the Windows documentation.

4. View the available devices by clicking the arrow to the left of `[btn]`: ‘Other devices’.

5. Locate the appropriate device
   There are up to eight devices available: the memory ballooning device, the paravirtual serial port device, the network device, the block device, the SCSI block device, the entropy source device, the emulated PCI serial port device, and the paravirtual panic device. Additionally, the video device is available in the Display adapters group.

   - Balloon, the balloon driver, applies to PCI Device in the Other devices group.
   - vioserial, the paravirtual serial driver, applies to PCI Simple Communications Controller in the Other devices group.
   - NetKVM, the network driver, applies to Ethernet Controller in the Other devices group. This driver is only available if a virtio NIC is configured. Configurable parameters for this driver are documented in Appendix E, NetKVM Driver Parameters.
   - viostor, the block driver, applies to SCSI Controller in the Other devices group.
   - viocsci, the SCSI block driver, applies to SCSI Controller in the Other devices group.
   - viorng, the entropy source driver, applies to PCI Device in the Other devices group.
• `qemupciserial`, the emulated PCI serial driver, applies to PCI Serial Port in the Other devices group.

• `qxl`, the paravirtual video driver, applied to Microsoft Basic Display Adapter in the Display adapters group.

• `pvpanic`, the paravirtual panic driver, applies to Unknown device in the Other devices group.

**NOTE**

Both the memory balloon and entropy source devices are initially represented by PCI Device in the Other devices group. Similarly, both block devices are represented by SCSI Controller. The following procedure is used to tell them apart. ---

Right-click the device that you wish to identify and select the `Details` tab. Select the Hardware Ids property in the list. Match the hardware ID with one of the IDs below:

• `VEN_1AF4&DEV_1002` or `VEN_1AF4&DEV_1045`, the balloon device.

• `VEN_1AF4&DEV_1003` or `VEN_1AF4&DEV_1043`, the paravirtual serial port device.

• `VEN_1AF4&DEV_1000` or `VEN_1AF4&DEV_1041`, the network device.

• `VEN_1AF4&DEV_1001` or `VEN_1AF4&DEV_1042`, the block device.

• `VEN_1AF4&DEV_1004` or `VEN_1AF4&DEV_1048`, the SCSI block device.

• `VEN_1AF4&DEV_1005` or `VEN_1AF4&DEV_1044`, the entropy source device.

• `VEN_1B36&DEV_0002`, the emulated PCI serial driver.

• `VEN_1B36&DEV_0100`, the video device.

• `VEN_QEMU&DEV_0001`, the guest panic device. ---

6. Right-click the device whose driver you want to update, and select **Update Driver Software** from the pop-up menu.

The Update Driver Software wizard starts.
The first page of the driver update wizard asks how you want to search for driver software.

7. Click **Browse my computer for driver software**.

8. Click **Browse**.
   A file browser opens.

9. Browse to the location of the driver.
   A separate driver is provided for each combination of operating systems and architectures. The drivers are arranged hierarchically according to their driver type, the operating system, and the architecture on which they will be installed: `driver_type/os/arch/`.

   For example, the Balloon driver for a Windows 10 operating system with an x86 (32-bit) architecture, resides in the `Balloon/w10/x86` directory.

   Once you have navigated to the correct location, click **OK**.

10. Click **Next**.
    The following screen is displayed while the driver installs:
When the installation is complete, the following screen is displayed:

11. Click Close.

12. Reboot the VM to complete the driver installation.
NOTE

After the driver is successfully installed, the device no longer appears in the Other devices group. Find the device in the Multifunction adapters group (1x QEMU PCI Serial Card), Network adapters group (Red Hat VirtIO Ethernet Adapter), Storage controllers group (Red Hat VirtIO SCSI controller and Red Hat VirtIO SCSI pass-through controller), and System devices group (QEMU PV Panic Device, VirtIO Balloon Driver, VirtIO RNG Device, and VirtIO Serial Driver).

13.3.1.2.2. Installing KVM drivers while installing Windows on a virtual machine

You can install the virtio drivers while installing Windows on a virtual machine (VM). This enables a Windows VM to use the virtio drivers for the default storage device.

Procedure

1. Create the VM using the virt-install utility.
   For more information, see Section 2.2.2, “Creating virtual machines using the command-line interface”.

   IMPORTANT

   Do not start the VM.

   Add the following parameters to the virt-install command:

   • The required parameter to add the storage medium with the KVM virtio drivers to the VM. For example, if the virtio drivers are on a virtual disk, add the following parameter:

     --disk path=/usr/share/virtio-win/virtio-win_*arch*.qcow2,device=disk,bus=virtio

     IMPORTANT

     If the device you want to add is a disk (that is, not a CD-ROM), add the bus=virtio option to the end of the --disk parameter:

     --disk path=/usr/share/virtio-win/virtio-win_*arch*.qcow2,device=disk,bus=virtio

   • The required parameter for the Windows version you will install on the VM. For example, for Windows 10:

     --os-variant win10

     For a list of Windows versions and the appropriate option, run the following command:

     # osinfo-query os

2. Start the VM and install the Windows operating system.
   For more information, see Starting virtual machines.

3. Install the virtio drivers.
   For information on installing these drivers, refer to the documentation for the version of Windows that you are installing.
13.3.2. Configuring the NetKVM driver for Windows virtual machines on RHEL 8

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 re-load 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 RHEL 8 virtual machines.

**13.3.2.1. Configuring NetKVM driver parameters**

The following provides information about configuring NetKVM driver parameters on a Windows virtual machine.

**Prerequisites**

- The NetKVM driver is installed on the virtual machine.

**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**.

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.

**13.3.2.2. NetKVM driver parameters**

The following table provides information on the configurable NetKVM driver logging parameters.

**Table 13.1. Logging parameters**
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description 2</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>
<tr>
<td></td>
<td><strong>NOTE</strong></td>
</tr>
<tr>
<td></td>
<td>High logging levels will slow down your virtual machine.</td>
</tr>
</tbody>
</table>

The following table provides information on the configurable NetKVM driver initial parameters.

**Table 13.2. 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>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.</td>
</tr>
<tr>
<td></td>
<td>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>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Init.MaxRxBuffers  | An integer that represents the number of RX ring descriptors that will be allocated.  
The default value is 256.  
Valid values are: 16, 32, 64, 128, 256, 512, and 1024. |
| Offload.Tx.Checksum| Specifies the TX checksum offloading mode.  
In Red Hat Enterprise Linux 8, the valid values for this parameter are:  
* All (the default) which enables IP, TCP, and UDP checksum offloading for both IPv4 and IPv6  
* TCP/UDP(v4,v6) which enables TCP and UDP checksum offloading for both IPv4 and IPv6  
* TCP/UDP(v4) which enables TCP and UDP checksum offloading for IPv4 only  
* TCP(v4) which enables only TCP checksum offloading for IPv4 only |

13.3.3. Using libguestfs with Windows virtual machines on RHEL 8

**libguestfs** utilities enable accessing, editing, and creating guest virtual machine disks or other disk images.

**Procedure**

- Install the **libguestfs-tools** and **libguestfs-winsupport** packages on the host machine.

  ```bash
  $ sudo yum install libguestfs-tools libguestfs-winsupport
  ```
CHAPTER 14. FEATURE SUPPORT AND LIMITATIONS IN RHEL 8 VIRTUALIZATION

This document provides information on feature support and restrictions in Red Hat Enterprise Linux 8 (RHEL 8) virtualization.

14.1. HOW RHEL 8 VIRTUALIZATION SUPPORT WORKS

A set of support limitations applies to virtualization in Red Hat Enterprise Linux 8 (RHEL 8). This means that when you use certain features or exceed a certain amount of allocated resources when using virtual machines in RHEL 8, Red Hat will not support these guests unless you have a specific subscription plan.

Features listed in Section 14.2, “Recommended features in RHEL 8 virtualization” have been tested and certified by Red Hat to work with the KVM hypervisor on a RHEL 8 system. Therefore, they are fully supported and recommended for use in virtualization in RHEL 8.

Features listed in Section 14.3, “Unsupported features in RHEL 8 virtualization” may work, but are not supported and not intended for use in RHEL 8. Therefore, Red Hat strongly recommends not using these features in RHEL 8 with KVM.

Section 14.4, “Resource allocation limits in RHEL 8 virtualization” lists the maximum amount of specific resources supported on a KVM guest in RHEL 8. 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 8 virtualization are supported. However, some of these have not been completely tested and therefore may not be fully optimized.

NOTE

Many of these limitations do not apply to other virtualization solutions provided by Red Hat, such as Red Hat Virtualization (RHV) or Red Hat OpenStack Platform (RHOSP).

14.2. RECOMMENDED FEATURES IN RHEL 8 VIRTUALIZATION

The following features are recommended for use with the KVM hypervisor included with Red Hat Enterprise Linux 8 (RHEL 8):

- **Host system architectures**
  Red Hat Enterprise Linux with KVM is supported only on the following host architectures:
  - AMD64 and Intel 64
  - IBM Z - IBM z13 systems and later
  - IBM POWER8
  - IBM POWER9
NOTE

RHEL 8 documentation primarily describes AMD64 and Intel 64 features and usage. For information about the specific of using RHEL 8 virtualization on different architectures, see:

- Chapter 3, Getting started with virtualization in RHEL 8 on IBM POWER
- Chapter 4, Getting started with virtualization in RHEL 8 on IBM Z.

- **Guest operating systems**
  Red Hat supports KVM virtual machines that use the following operating systems:
  - Red Hat Enterprise Linux 6 and later
  - Microsoft Windows 10 and later
  - Microsoft Windows Server 2016 and later

- **Q35 guests**
  The recommended machine type for KVM guest virtual machines is QEMU Q35, which emulates the ICH9 chipset.

**Additional resources**

- For information about unsupported guest OS types and features in RHEL 8 virtualization, see Section 14.3, “Unsupported features in RHEL 8 virtualization”.

- For information about the maximum supported amounts of resources that can be allocated to a virtual machine, see Section 14.4, “Resource allocation limits in RHEL 8 virtualization”.

14.3. **UNSUPPORTED FEATURES IN RHEL 8 VIRTUALIZATION**

The following features are not supported by the KVM hypervisor included with Red Hat Enterprise Linux 8 (RHEL 8):

NOTE

Many of these limitations do not apply to other virtualization solutions provided by Red Hat, such as Red Hat Virtualization (RHV) or Red Hat OpenStack Platform (RHOSP).

- **Guest operating systems**
  KVM virtual machines using the following guest operating systems on a RHEL 8 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
vCPU hot unplug
Removing a virtual CPU (vCPU) from a running KVM virtual machine, also referred to as a vCPU hot unplug, is unsupported in RHEL 8.

Memory hot unplug
Decreasing the memory limit allocated to running a KVM virtual machine, also referred to as a memory hot unplug, is unsupported in RHEL 8.

SR-IOV networking
Single-root I/O virtualization (SR-IOV) networking is not supported in RHEL 8.

I/O throttling
Configuring maximum input and output levels for operations on virtual disk, also known as I/O throttling, is not supported in RHEL 8.

Storage live migration
Migrating a disk image of a running virtual machine between hosts is not supported in RHEL 8.

Live snapshots
Creating or loading a snapshot of a running virtual machine, also referred to as a live snapshot, is not supported in RHEL 8.

Vhost-user
RHEL 8 does not support the implementation of a user-space vHost interface.

S3 and S4 system power states
Suspending a virtual machine 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 virtual machine not supportable by Red Hat.

S3-PR on a multipathed vDisk
SCSI3 persistent reservation (S3-PR) on a multipathed vDisk is not supported in RHEL 8. As a consequence, Windows Cluster is not supported in RHEL 8. In case you need Windows Cluster support, use Red Hat Virtualization (RHV) instead.

virtio-crypto
The drivers for the virtio-crypto device are available in the RHEL 8 kernel, and the device can thus be enabled on a KVM hypervisor under certain circumstances. However, using the virtio-crypto device in RHEL 8 is not supported and highly discouraged.

Additional resources
- For information about supported guest OS types and recommended features in RHEL 8 virtualization, see Section 14.2, “Recommended features in RHEL 8 virtualization”.
- For information about the maximum supported amounts of resources that can be allocated to a virtual machine, see Section 14.4, “Resource allocation limits in RHEL 8 virtualization”.

14.4. RESOURCE ALLOCATION LIMITS IN RHEL 8 VIRTUALIZATION

The following limits apply to virtualized resources that can be allocated to a single virtual machine (VM) on a Red Hat Enterprise Linux 8 (RHEL 8) host.
NOTE

Many of these limitations do not apply to other virtualization solutions provided by Red Hat, such as Red Hat Virtualization (RHV) or Red Hat OpenStack Platform (RHOSP).

- **Maximum virtual machines per host**
  A single RHEL 8 host supports up to 4 VMs running at the same time.

- **Maximum vCPUs per VM**
  RHEL 8 supports up to 384 vCPUs allocated to a single KVM virtual machine.

- **PCI devices per VM**
  RHEL 8 supports 32 PCI device slots per virtual machine 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 virtual machine, 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 virtual machine.