Red Hat Enterprise Linux 9.0 Beta

Configuring and managing virtualization

Setting up your host, creating and administering virtual machines, and understanding virtualization features in Red Hat Enterprise Linux 9
Red Hat Enterprise Linux 9.0 Beta Configuring and managing virtualization

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

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

MAKING OPEN SOURCE MORE INCLUSIVE ......................................................... 6

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION ............................. 7

CHAPTER 1. INTRODUCING VIRTUALIZATION IN RHEL ............................... 8
  1.1. WHAT IS VIRTUALIZATION? ............................................................. 8
  1.2. ADVANTAGES OF VIRTUALIZATION ............................................. 8
  1.3. VIRTUAL MACHINE COMPONENTS AND THEIR INTERACTION ............. 9
  1.4. TOOLS AND INTERFACES FOR VIRTUALIZATION MANAGEMENT ...... 10
  1.5. RED HAT VIRTUALIZATION SOLUTIONS ..................................... 11

CHAPTER 2. ENABLING VIRTUALIZATION .................................................... 13

CHAPTER 3. CREATING VIRTUAL MACHINES ............................................ 15
  3.1. CREATING VIRTUAL MACHINES USING THE COMMAND-LINE INTERFACE 15
  3.2. CREATING VIRTUAL MACHINES AND INSTALLING GUEST OPERATING SYSTEMS USING THE WEB CONSOLE 18
      3.2.1. Creating virtual machines using the web console 18
      3.2.2. Creating virtual machines by importing disk images using the web console 20
      3.2.3. Installing guest operating systems using the web console 21

CHAPTER 4. STARTING VIRTUAL MACHINES ............................................. 23
  4.1. STARTING A VIRTUAL MACHINE USING THE COMMAND-LINE INTERFACE 23
  4.2. STARTING VIRTUAL MACHINES USING THE WEB CONSOLE ............... 24
  4.3. STARTING VIRTUAL MACHINES AUTOMATICALLY WHEN THE HOST STARTS 24

CHAPTER 5. CONNECTING TO VIRTUAL MACHINES .................................. 27
  5.1. INTERACTING WITH VIRTUAL MACHINES USING THE WEB CONSOLE 27
      5.1.1. Viewing the virtual machine graphical console in the web console 27
      5.1.2. Viewing the graphical console in a remote viewer using the web console 28
      5.1.3. Viewing the virtual machine serial console in the web console 31
  5.2. OPENING A VIRTUAL MACHINE GRAPHICAL CONSOLE USING VIRT VIEWER 32
  5.3. CONNECTING TO A VIRTUAL MACHINE USING SSH ......................... 33
  5.4. OPENING A VIRTUAL MACHINE SERIAL CONSOLE .......................... 34
  5.5. SETTING UP EASY ACCESS TO REMOTE VIRTUALIZATION HOSTS .......... 36

CHAPTER 6. SHUTTING DOWN VIRTUAL MACHINES .................................. 39
  6.1. SHUTTING DOWN A VIRTUAL MACHINE USING THE COMMAND-LINE INTERFACE 39
  6.2. SHUTTING DOWN AND RESTARTING VIRTUAL MACHINES USING THE WEB CONSOLE 39
      6.2.1. Shutting down virtual machines in the web console 39
      6.2.2. Restarting virtual machines using the web console 40
      6.2.3. Sending non-maskable interrupts to VMs using the web console 41

CHAPTER 7. DELETING VIRTUAL MACHINES ............................................ 42
  7.1. DELETING VIRTUAL MACHINES USING THE COMMAND LINE INTERFACE 42
  7.2. DELETING VIRTUAL MACHINES USING THE WEB CONSOLE ................. 42

CHAPTER 8. MANAGING VIRTUAL MACHINES IN THE WEB CONSOLE ............ 44
  8.1. OVERVIEW OF VIRTUAL MACHINE MANAGEMENT USING THE WEB CONSOLE 44
  8.2. SETTING UP THE WEB CONSOLE TO MANAGE VIRTUAL MACHINES ....... 44

CHAPTER 9. MIGRATING VIRTUAL MACHINES ........................................... 46
  9.1. HOW MIGRATING VIRTUAL MACHINES WORKS ................................. 46
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2. BENEFITS OF MIGRATING VIRTUAL MACHINES</td>
<td>47</td>
</tr>
<tr>
<td>9.3. LIMITATIONS FOR MIGRATING VIRTUAL MACHINES</td>
<td>47</td>
</tr>
<tr>
<td>9.4. SHARING VIRTUAL MACHINE DISK IMAGES WITH OTHER HOSTS</td>
<td>48</td>
</tr>
<tr>
<td>9.5. MIGRATING A VIRTUAL MACHINE USING THE COMMAND-LINE INTERFACE</td>
<td>49</td>
</tr>
<tr>
<td>9.6. LIVE MIGRATING A VIRTUAL MACHINE USING THE WEB CONSOLE</td>
<td>52</td>
</tr>
<tr>
<td>9.7. SUPPORTED HOSTS FOR VIRTUAL MACHINE MIGRATION</td>
<td>54</td>
</tr>
<tr>
<td>CHAPTER 10. MANAGING VIRTUAL DEVICES</td>
<td>55</td>
</tr>
<tr>
<td>10.1. HOW VIRTUAL DEVICES WORK</td>
<td>55</td>
</tr>
<tr>
<td>10.2. VIEWING DEVICES ATTACHED TO VIRTUAL MACHINES USING THE WEB CONSOLE</td>
<td>56</td>
</tr>
<tr>
<td>10.3. ATTACHING DEVICES TO VIRTUAL MACHINES</td>
<td>57</td>
</tr>
<tr>
<td>10.4. MODIFYING DEVICES ATTACHED TO VIRTUAL MACHINES</td>
<td>58</td>
</tr>
<tr>
<td>10.5. REMOVING DEVICES FROM VIRTUAL MACHINES</td>
<td>59</td>
</tr>
<tr>
<td>10.6. TYPES OF VIRTUAL DEVICES</td>
<td>60</td>
</tr>
<tr>
<td>10.7. MANAGING VIRTUAL USB DEVICES</td>
<td>62</td>
</tr>
<tr>
<td>10.7.1. Attaching USB devices to virtual machines</td>
<td>62</td>
</tr>
<tr>
<td>10.7.2. Removing USB devices from virtual machines</td>
<td>63</td>
</tr>
<tr>
<td>10.7.3. Additional resources</td>
<td>64</td>
</tr>
<tr>
<td>10.8. MANAGING VIRTUAL OPTICAL DRIVES</td>
<td>64</td>
</tr>
<tr>
<td>10.8.1. Attaching optical drives to virtual machines</td>
<td>64</td>
</tr>
<tr>
<td>10.8.2. Replacing ISO images in virtual optical drives</td>
<td>65</td>
</tr>
<tr>
<td>10.8.3. Removing ISO images from virtual optical drives</td>
<td>66</td>
</tr>
<tr>
<td>10.8.4. Removing optical drives from virtual machines</td>
<td>66</td>
</tr>
<tr>
<td>10.8.5. Additional resources</td>
<td>67</td>
</tr>
<tr>
<td>10.9. MANAGING SR-IOV DEVICES</td>
<td>67</td>
</tr>
<tr>
<td>10.9.1. What is SR-IOV?</td>
<td>67</td>
</tr>
<tr>
<td>10.9.2. Attaching SR-IOV networking devices to virtual machines</td>
<td>69</td>
</tr>
<tr>
<td>10.9.3. Supported devices for SR-IOV assignment</td>
<td>72</td>
</tr>
<tr>
<td>10.10. ATTACHING DASD DEVICES TO VIRTUAL MACHINES ON IBM Z</td>
<td>72</td>
</tr>
<tr>
<td>CHAPTER 11. CONFIGURING VIRTUAL MACHINE NETWORK CONNECTIONS</td>
<td>76</td>
</tr>
<tr>
<td>11.1. UNDERSTANDING VIRTUAL NETWORKING</td>
<td>76</td>
</tr>
<tr>
<td>11.1.1. How virtual networks work</td>
<td>76</td>
</tr>
<tr>
<td>11.1.2. Virtual networking default configuration</td>
<td>77</td>
</tr>
<tr>
<td>11.2. USING THE WEB CONSOLE FOR MANAGING VIRTUAL MACHINE NETWORK INTERFACES</td>
<td>78</td>
</tr>
<tr>
<td>11.2.1. Viewing and editing virtual network interface information in the web console</td>
<td>78</td>
</tr>
<tr>
<td>11.2.2. Adding and connecting virtual network interfaces in the web console</td>
<td>79</td>
</tr>
<tr>
<td>11.2.3. Disconnecting and removing virtual network interfaces in the web console</td>
<td>80</td>
</tr>
<tr>
<td>11.3. RECOMMENDED VIRTUAL MACHINE NETWORKING CONFIGURATIONS USING THE COMMAND-LINE INTERFACE</td>
<td>80</td>
</tr>
<tr>
<td>11.3.1. Configuring externally visible virtual machines using the command-line interface</td>
<td>81</td>
</tr>
<tr>
<td>11.3.2. Isolating virtual machines from each other using the command-line interface</td>
<td>82</td>
</tr>
<tr>
<td>11.4. RECOMMENDED VIRTUAL MACHINE NETWORKING CONFIGURATIONS USING THE WEB CONSOLE</td>
<td>84</td>
</tr>
<tr>
<td>11.4.1. Configuring externally visible virtual machines using the web console</td>
<td>84</td>
</tr>
<tr>
<td>11.4.2. Isolating virtual machines from each other using the web console</td>
<td>86</td>
</tr>
<tr>
<td>11.5. TYPES OF VIRTUAL MACHINE NETWORK CONNECTIONS</td>
<td>87</td>
</tr>
<tr>
<td>11.5.1. Virtual networking with network address translation</td>
<td>87</td>
</tr>
<tr>
<td>11.5.2. Virtual networking in routed mode</td>
<td>88</td>
</tr>
<tr>
<td>11.5.3. Virtual networking in bridged mode</td>
<td>89</td>
</tr>
<tr>
<td>11.5.4. Virtual networking in isolated mode</td>
<td>91</td>
</tr>
<tr>
<td>11.5.5. Virtual networking in open mode</td>
<td>91</td>
</tr>
<tr>
<td>11.5.6. Direct attachment of the virtual network device</td>
<td>91</td>
</tr>
</tbody>
</table>
11.5.7. Comparison of virtual machine connection types .......................... 92
11.6. BOOTING VIRTUAL MACHINES FROM A PXE SERVER .................. 92
  11.6.1. Setting up a PXE boot server on a virtual network ...................... 93
  11.6.2. Booting virtual machines using PXE and a virtual network .......... 94
  11.6.3. Booting virtual machines using PXE and a bridged network ......... 95
11.7. ADDITIONAL RESOURCES ............................................. 96

CHAPTER 12. OPTIMIZING VIRTUAL MACHINE PERFORMANCE ....................... 97
  12.1. WHAT INFLUENCES VIRTUAL MACHINE PERFORMANCE ................. 97
    The impact of virtualization on system performance ........................ 97
    Reducing VM performance loss ........................................... 97
  12.2. OPTIMIZING VIRTUAL MACHINE PERFORMANCE USING TUNED .......... 98
  12.3. CONFIGURING VIRTUAL MACHINE MEMORY ................................ 99
    12.3.1. Adding and removing virtual machine memory using the web console 99
    12.3.2. Adding and removing virtual machine memory using the command-line interface 100
    12.3.3. Additional resources ............................................. 102
  12.4. OPTIMIZING VIRTUAL MACHINE I/O PERFORMANCE ....................... 102
    12.4.1. Tuning block I/O in virtual machines ................................ 102
    12.4.2. Disk I/O throttling in virtual machines ............................. 103
    12.4.3. Enabling multi-queue virtio-scsi .................................. 104
  12.5. OPTIMIZING VIRTUAL MACHINE CPU PERFORMANCE ....................... 104
    12.5.1. Adding and removing virtual CPUs using the command-line interface 105
    12.5.2. Managing virtual CPUs using the web console ........................ 106
    12.5.3. Configuring NUMA in a virtual machine ............................. 107
    12.5.4. Sample vCPU performance tuning scenario ............................ 109
    12.5.5. Deactivating kernel same-page merging .............................. 114
  12.6. OPTIMIZING VIRTUAL MACHINE NETWORK PERFORMANCE ............... 115
  12.7. VIRTUAL MACHINE PERFORMANCE MONITORING TOOLS .................... 116
  12.8. ADDITIONAL RESOURCES ............................................. 118

CHAPTER 13. SHARING FILES BETWEEN THE HOST AND ITS VIRTUAL MACHINES .... 119
  13.1. SHARING FILES BETWEEN THE HOST AND ITS VIRTUAL MACHINES USING VIRTIOFS 119
  13.2. SHARING FILES BETWEEN THE HOST AND LINUX VIRTUAL MACHINES .... 120
  13.3. SHARING FILES BETWEEN THE HOST AND WINDOWS VIRTUAL MACHINES ... 123

CHAPTER 14. FEATURE SUPPORT AND LIMITATIONS IN RHEL 9 VIRTUALIZATION ........ 127
  14.1. HOW RHEL VIRTUALIZATION SUPPORT WORKS ............................. 127
  14.2. RECOMMENDED FEATURES IN RHEL 9 VIRTUALIZATION .................... 127
  14.3. UNSUPPORTED FEATURES IN RHEL 9 VIRTUALIZATION ...................... 128
  14.4. RESOURCE ALLOCATION LIMITS IN RHEL 9 VIRTUALIZATION .............. 133
  14.5. AN OVERVIEW OF VIRTUALIZATION FEATURES SUPPORT IN RHEL 9 ....... 133
RHEL BETA RELEASE

Red Hat provides Red Hat Enterprise Linux Beta access to all subscribed Red Hat accounts. The purpose of Beta access is to:

- Provide an opportunity to customers to test major features and capabilities prior to the general availability release and provide feedback or report issues.

- Provide Beta product documentation as a preview. Beta product documentation is under development and is subject to substantial change.

Note that Red Hat does not support the usage of RHEL Beta releases in production use cases. For more information, see What does Beta mean in Red Hat Enterprise Linux and can I upgrade a RHEL Beta installation to a General Availability (GA) release?.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message.
We appreciate your input on our documentation. Please let us know how we could make it better.

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

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

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

1.1. WHAT IS VIRTUALIZATION?

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

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

VMs enable you to safely test software configurations and features, run legacy software, or optimize the workload efficiency of your hardware. For more information on the benefits, see Section 1.2, “Advantages of virtualization”.

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

Next steps

- To start using virtualization in Red Hat Enterprise Linux 9, see Chapter 2, Enabling virtualization.
- In addition to Red Hat Enterprise Linux 9 virtualization, Red Hat offers a number of specialized virtualization solutions, each with a different user focus and features. For more information, see 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 on the software level, and is therefore very flexible. A VM uses a configurable fraction of the host memory, CPUs, or storage space, and that configuration can specify very fine-grained resource requests.

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

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

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

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

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

  **NOTE**
  Not all operating systems are supported as a guest OS in a RHEL 9 host. For details, see Section 14.2, “Recommended features in RHEL 9 virtualization”.

### 1.3. VIRTUAL MACHINE COMPONENTS AND THEIR INTERACTION

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

#### Hypervisor

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

The hypervisor includes the Kernel-based Virtual Machine (KVM) module and virtualization kernel drivers, 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) determines all settings and devices in a specific VM. The configuration includes:

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

For more information on the contents of an XML configuration, see Viewing information about virtual machines.

#### Component interaction

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

**NOTE**

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

For more information on the host-based interfaces, see Section 1.4, “Tools and interfaces for virtualization management”.

**Figure 1.1. RHEL 9 virtualization architecture**

![RHEL 9 virtualization architecture diagram](image)

### 1.4. TOOLS AND INTERFACES FOR VIRTUALIZATION MANAGEMENT

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

**Command-line interface**

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

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

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

- **virt-install** - A CLI utility for creating new VMs. For more information, see the **virt-install(1)** man page.
- **virt-xml** - A utility for editing the configuration of a VM.
- **guestfish** - A utility for examining and modifying VM disk images. For more information, see the **guestfish(1)** man page.

**Graphical interfaces**

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

- The **RHEL 9 web console**, also known as **Cockpit**, provides a remotely accessible and easy to use graphical user interface for managing VMs and virtualization hosts.
  For instructions on basic virtualization management with the web console, see Chapter 8, *Managing virtual machines in the web console*.

### 1.5. RED HAT VIRTUALIZATION SOLUTIONS

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

**Red Hat Virtualization (RHV)**

RHV is designed for enterprise-class scalability and performance, and enables the management of your entire virtual infrastructure, including hosts, virtual machines, networks, storage, and users from a centralized graphical interface.

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 the Red Hat Customer Portal.

**Red Hat OpenStack Platform (RHOSP)**

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

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

---

**NOTE**

For details on virtualization features not supported on RHEL but supported on RHV or RHOSP, see Unsupported features in RHEL 9 virtualization.

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. ENABLING VIRTUALIZATION

To use virtualization in RHEL 9, you must install virtualization packages, and ensure your system is configured to host virtual machines (VMs).

Prerequisites

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

Procedure

1. Install the `virt-install` and `virt-viewer` packages:

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

2. Start the `libvirtd` service.

   ```bash
   # systemctl start libvirtd
   ```

Verification

1. 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)
   ```

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

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.

Next steps

- Create a virtual machine on your RHEL 9 host
CHAPTER 3. CREATING VIRTUAL MACHINES

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

3.1. CREATING VIRTUAL MACHINES USING THE COMMAND-LINE INTERFACE

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

**Prerequisites**

- Virtualization is **enabled** on your host system.
- You have sufficient a amount of system resources to allocate to your VMs, such as disk space, RAM, or CPUs. The recommended values may vary significantly depending on the intended tasks and workload of the VMs.
- An operating system (OS) installation source is available locally or on a network. This can be one of the following:
  - An ISO image of an installation medium
  - A disk image of an existing VM installation

**WARNING**

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

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

**Procedure**

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

- The name of the new machine (**--name**)
- The amount of allocated memory (**--memory**)
- The number of allocated virtual CPUs (**--vcpus**)
- The type and size of the allocated storage (**--disk**)
- The type and location of the OS installation source (**--cdrom** or **--location**)
Based on the chosen installation method, the necessary options and values can vary. See below for examples:

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

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

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

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

- The following creates a RHEL 9 VM named `demo-guest3` that connects to an existing disk image, `/home/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 rhel9.0 --import --disk /home/username/backup/disk.qcow2
  ```

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

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

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

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

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

- The following creates a VM named `demo-guest6`, which has the same configuration as demo-guest5, but resides on the 10.0.0.1 remote host.
# virt-install --connect qemu+ssh://root@10.0.0.1/system --name demo-guest6 --memory 16384 --vcpus 16 --disk size=280 --os-variant rhel9.0 --location RHEL9.iso --graphics none --extra-args='console=ttys0'

- The following creates a VM named **demo-guest7**, which has the same configuration as demo-guest5, but for its storage, it uses a DASD mediated device **mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8**, and assigns it device number **1111**.

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

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

**Verification**

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

**Troubleshooting**

- If **virt-install** fails with a **cannot find default network** error:
  
  a. Ensure that the **libvirt-daemon-config-network** package is installed:

     ```
     # dnf info libvirt-daemon-config-network
     Installed Packages
     Name : libvirt-daemon-config-network
     [...] 
     ```

  b. Verify that the **libvirt** default network is active and configured to start automatically:

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

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

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

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

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

     ```
     error: failed to get network 'default'
     error: Network not found: no network with matching name 'default'
     ```
To fix this, re-install `libvirt-daemon-config-network`.

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

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

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

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

### Additional resources

- `man virt-install`

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

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

#### 3.2.1. Creating virtual machines using the web console

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

**Prerequisites**

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

**Procedure**

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

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Unique name</td>
</tr>
<tr>
<td>Installation type</td>
<td>Download an OS</td>
</tr>
<tr>
<td>Operating system</td>
<td>Choose an operating system</td>
</tr>
<tr>
<td>Storage</td>
<td>Create new volume</td>
</tr>
<tr>
<td>Size</td>
<td>10 GiB</td>
</tr>
<tr>
<td>Memory</td>
<td>1 GiB</td>
</tr>
<tr>
<td>Run unattended installation</td>
<td>False</td>
</tr>
<tr>
<td>Immediately start VM</td>
<td>True</td>
</tr>
</tbody>
</table>

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

- **Name** – The name of the VM.
- **Connection** – The type of libvirt connection, system or session. For more details, see System and session connections.
- **Installation type** – The installation can use a local installation medium, a URL, a PXE network boot, or download an OS from a limited set of operating systems.
- **Operating system** – The VM’s operating system. Note that Red Hat provides support only for a limited set of guest operating systems.
- **Storage** – The type of storage with which to configure the VM.
- **Size** – The amount of storage space with which to configure the VM.
- **Memory** – The amount of memory with which to configure the VM.
- **Run unattended installation** – Whether or not to run the installation unattended. This option is available only when the Installation type is Download an OS.
- **Immediately Start VM** – Whether or not the VM will start immediately after it is created.

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

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

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

**Prerequisites**

- The web console VM plug-in is installed on your system.
- You have sufficient a amount of system resources to allocate to your VMs, such as disk space, RAM, or CPUs. The recommended values can vary significantly depending on the intended tasks and workload of the VMs.
- Make sure you have a disk image of an existing VM installation

**Procedure**

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

   ![Import a virtual machine dialog](image)

   - **Name** - The name of the VM.
   - **Disk image** - The path to the existing disk image of a VM on the host system.
   - **Operating system** - The VM’s operating system. Note that Red Hat provides support only for a limited set of guest operating systems.
   - **Memory** - The amount of memory with which to configure the VM.
   - **Immediately start VM** - Whether or not the VM will start immediately after it is created.

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

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

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

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

**NOTE**

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

**Prerequisites**

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

**Procedure**

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

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

   **NOTE**

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

   a. Click the firmware.

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

   c. Click **Save**.

3. Click **Install**.
The installation routine of the operating system runs in the VM console.

Troubleshooting

- If the installation routine fails, the VM must be deleted and recreated.
CHAPTER 4. STARTING VIRTUAL MACHINES

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

Prerequisites

- Before a VM can be started, it must be created and, ideally, also installed with an OS. For instruction to do so, see Chapter 3, Creating virtual machines.

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

You can use the command line interface to start a shut-down virtual machine (VM) or restore a saved VM. Follow the procedure below.

Prerequisites

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

Procedure

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

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

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

  ```bash
  # virsh -c qemu+ssh://root@192.168.123.123/system start demo-guest1
  root@192.168.123.123's password:
  Last login: Mon Feb 18 07:28:55 2019
  Domain demo-guest1 started
  ```

Additional Resources

- For more `virsh start` arguments, use `virsh start --help`.
- For simplifying VM management on remote hosts, see modifying your libvirt and SSH configuration.
- You can use the `virsh autostart` utility to configure a VM to start automatically when the host boots up. For more information about autostart, see starting virtual machines automatically when the host starts.

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

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

**Prerequisites**

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

**Procedure**

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

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

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

**Additional resources**

- For information on shutting down a VM, see Section 6.2.1, “Shutting down virtual machines in the web console”.
- For information on restarting a VM, see Section 6.2.2, “Restarting virtual machines using the web console”.

### 4.3. STARTING VIRTUAL MACHINES AUTOMATICALLY WHEN THE HOST STARTS

When a host with a running virtual machine (VM) restarts, the VM must be started again manually by default. However, you might want your VM to be always available when the host is running. In that case, you can use the `virsh autostart` utility to configure the VM to be started automatically.

**Prerequisites**

- A created VM, for more information see creating virtual machines.

**Procedure**

1. Use the `virsh autostart` utility to configure the VM to start automatically when the host starts.
   For example, the following command configures the demo-guest1 VM to start automatically.
2. If you use network interfaces that are not managed by libvirt, you must also make additional changes to the systemd configuration. Otherwise, the affected VMs might fail to start.

**NOTE**

These interfaces include for example:

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

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

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

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

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

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

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

**Verification**

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

```
# virsh dominfo demo-guest1
Id:             2
Name:           demo-guest1
UUID:           e46bc81c-74e2-406e-bd7a-67042bae80d1
OS Type:        hvm
State:          running
CPU(s):         2
CPU time:       385.9s
Max memory:     4194304 KiB
Used memory:    4194304 KiB
Persistent:     yes
Autostart:      enable
Managed save:   no
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c873,c919 (enforcing)
```
2. If you use network interfaces that are not managed by libvirt, check if the content of the `10-network-online.conf` file matches the following output.

```
$ cat /etc/systemd/system/libvirtd.service.d/10-network-online.conf
[Unit]
After=network-online.target
```

Additional resources

- `$ virsh autostart --help`

- You can also enable autostart using the web console, see `starting virtual machines using the web console`. 

CHAPTER 5. CONNECTING TO VIRTUAL MACHINES

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

- When using the web console interface, use the Virtual Machines pane in the web console interface. For more information, see Section 5.1, “Interacting with virtual machines using the web console”.

- If you need to interact with a VM graphical display without using the web console, use the Virt Viewer application. For details, see Section 5.2, “Opening a virtual machine graphical console using Virt Viewer”.

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

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

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

Prerequisites

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

5.1. INTERACTING WITH VIRTUAL MACHINES USING THE WEB CONSOLE

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

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

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

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

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

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

Prerequisites

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

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

Procedure

1. In the Virtual Machines interface, click the VM whose graphical console you want to view. A new page opens with an Overview and a Console section for the VM.
2. Select **VNC console** in the console drop down menu. The VNC console appears below the menu in the web interface.

The graphical console appears in the web interface.

3. Click **Expand**

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

**NOTE**

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

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

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

**Additional resources**

- For instructions on viewing the graphical console in a remote viewer, see Section 5.1.2, “Viewing the graphical console in a remote viewer using the web console”.

- For instructions on viewing the serial console in the web console, see Section 5.1.3, “Viewing the virtual machine serial console in the web console”.

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

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

**NOTE**

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

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

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

2. Open the file to launch Virt Viewer.

NOTE

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

Procedure

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

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

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

NOTE

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

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

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

Troubleshooting

- If launching the Remote Viewer in the web console does not work or is not optimal, you can manually connect with any viewer application using the following protocols:

  - Address - The default address is 127.0.0.1. You can modify the vnc_listen or the spice_listen parameter in /etc/libvirt/qemu.conf to change it to the host’s IP address.

  - SPICE port - 5900
5.1.3. Viewing the virtual machine serial console in the web console

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

For more information about the serial console, see Section 5.4, “Opening a virtual machine serial console”.

Prerequisites

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

Procedure

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

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

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

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

- To reconnect the serial console to the VM, click Reconnect.

Additional resources

- For instructions on viewing the graphical console in the web console, see Section 5.1.1, “Viewing the virtual machine graphical console in the web console”.

- For instructions on viewing the serial console in the web console, see Section 5.1.3, “Viewing the virtual machine serial console in the web console”.

VNC port - 5901
5.2. OPENING A VIRTUAL MACHINE GRAPHICAL CONSOLE USING VIRT VIEWER

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

Prerequisites

- Your system, as well as the VM you are connecting to, must support graphical displays.
- If the target VM is located on a remote host, connection and root access privileges to the host are needed.
- Optional: If the target VM is located on a remote host, set up your libvirt and SSH for more convenient access to remote hosts.

Procedure

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

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

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

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

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 VMs in an interactive GUI in RHEL 9, you can use the web console interface. For more information, see Section 5.1, “Interacting with virtual machines using the web console”.

### 5.3. CONNECTING TO A VIRTUAL MACHINE USING SSH

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

**Prerequisites**
• You have network connection and root access privileges to the target VM.

• If the target VM is located on a remote host, you also have connection and root access privileges to that host.

• The **libvirt-nss** component is installed and enabled on the VM’s host. If it is not, do the following:

  a. Install the **libvirt-nss** package:

     ```bash
     # dnf install libvirt-nss
     ```

  b. Edit the `/etc/nsswitch.conf` file and add **libvirt_guest** to the **hosts** line:

     ```
     [...]
     passwd: compat
     shadow: compat
     group: compat
     hosts:     files libvirt_guest dns
     [...]
     ```

**Procedure**

1. **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:

   ```bash
   # 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 the “testguest1” VM using its root credentials:

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

**Troubleshooting**

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

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

**5.4. 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 a 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` VM, if the libvirt driver supports safe console handling:

```
# virsh console guest1 --safe
Connected to domain guest1
Escape character is '^]'
```

Subscription-name
Kernel 3.10.0-948.el7.x86_64 on an x86_64

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

Additional resources

- For more information about the VM serial console, see the virsh man page.

5.5. SETTING UP EASY ACCESS TO REMOTE VIRTUALIZATION HOSTS

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

```
# virsh -c qemu+ssh://root@10.0.0.1/system list
```

```
root@10.0.0.1's password:
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:

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

3. As a result, you can manage remote VMs by using libvirt-based utilities on the local system with an added -c qemu-host-alias parameter. This automatically performs the commands over SSH on the remote host.

For example, 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, you can avoid having to provide the root password to the remote system. To do so, use one or more of the following methods:

- Set up key-based SSH access to the remote host.
- Use SSH connection multiplexing to connect to the remote system.
- Set up a kerberos authentication ticket on the remote system. For instructions, see Kerberos authentication in Identity Management.

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`
CHAPTER 6. SHUTTING DOWN VIRTUAL MACHINES

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

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

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

- Use a shutdown command appropriate to the guest OS while connected to the guest.
- Use the `virsh shutdown` command on the host:
  - If the VM is on a local host:
    ```
    # virsh shutdown demo-guest1
    Domain demo-guest1 is being shutdown
    ```
  - If the VM is on a remote host, in this example 10.0.0.1:
    ```
    # virsh -c qemu+ssh://root@10.0.0.1/system shutdown demo-guest1
    root@10.0.0.1's password:
    Last login: Mon Feb 18 07:28:55 2019
    Domain demo-guest1 is being shutdown
    ```

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

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

**NOTE**

The `virsh destroy` command does not actually delete or remove the VM configuration or disk images. It only 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.

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

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

6.2.1. Shutting down virtual machines in the web console

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

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

Procedure

1. In the Virtual Machines interface, click the row 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.

Troubleshooting

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

- To shut down an unresponsive VM, you can also send a non-maskable interrupt. For more information, see Section 6.2.3, “Sending non-maskable interrupts to VMs using the web console”.

Additional resources

- For information on starting a VM, see Section 4.2, “Starting virtual machines using the web console”.

- For information on restarting a VM, see Section 6.2.2, “Restarting virtual machines using the web console”.

6.2.2. Restarting virtual machines using the web console

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

Prerequisites

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

Procedure

1. In the Virtual Machines interface, click the row 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.

Troubleshooting

- If the VM does not restart, click the Menu button next to the Restart button and select Force Restart.

- To restart an unresponsive VM, you can also send a non-maskable interrupt. For more information, see Section 6.2.3, “Sending non-maskable interrupts to VMs using the web console”.
6.2.3. Sending non-maskable interrupts to VMs using the web console

Sending a non-maskable interrupt (NMI) may cause an unresponsive running virtual machine (VM) to respond or shut down. For example, you can send the Ctrl+Alt+Del NMI to a VM that is not responding to standard input.

Prerequisites

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

Procedure

1. In the Virtual Machines interface, click the row 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 Menu button ⋮ 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 4.2, “Starting virtual machines using the web console”.

- For information on restarting a VM, see Section 6.2.2, “Restarting virtual machines using the web console”.

- For information on shutting down a VM, see Section 6.2.1, “Shutting down virtual machines in the web console”.

• For information on starting a VM, see Section 4.2, “Starting virtual machines using the web console”.

• For information on shutting down a VM, see Section 6.2.1, “Shutting down virtual machines in the web console”.

• For information on starting a VM, see Section 4.2, “Starting virtual machines using the web console”.

• For information on restarting a VM, see Section 6.2.2, “Restarting virtual machines using the web console”.

• For information on shutting down a VM, see Section 6.2.1, “Shutting down virtual machines in the web console”.
CHAPTER 7. DELETING VIRTUAL MACHINES

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

7.1. DELETING VIRTUAL MACHINES USING THE COMMAND LINE INTERFACE

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

Prerequisites

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

Procedure

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

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

Additional resources

- For other `virsh undefine` arguments, use `virsh undefine --help` or see the `virsh` man page.

7.2. DELETING VIRTUAL MACHINES USING THE WEB CONSOLE

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

Prerequisites

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

Procedure

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

![Confirm deletion of Grid_v2](image)

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

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

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

8.1. OVERVIEW OF VIRTUAL MACHINE MANAGEMENT USING THE WEB CONSOLE

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

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

Next steps

- For instructions on enabling VMs management in your web console, see Setting up the web console to manage virtual machines.

8.2. SETTING UP THE WEB CONSOLE TO MANAGE VIRTUAL MACHINES

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

Prerequisites

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

```
# systemctl status cockpit.socket
cockpit.socket - Cockpit Web Service Socket
Loaded: loaded (/usr/lib/systemd/system/cockpit.socket
[...]
```
If this command returns **Unit cockpit.socket could not be found**, follow the Installing the web console document to enable the web console.

**Procedure**

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

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

**Verification**

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

2. Log in.

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

**Additional resources**

- For instructions on connecting to the web console, as well as other information on using the web console, see the *Managing systems using the RHEL 9 web console* document.
CHAPTER 9. MIGRATING VIRTUAL MACHINES

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

9.1. HOW MIGRATING VIRTUAL MACHINES WORKS

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

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

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

Table 9.1. VM migration types

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

Additional resources

- For more information on the benefits of VM migration, see Section 9.2, “Benefits of migrating virtual machines”.

46
For instructions on setting up shared storage for migrating VMs, see Section 9.4, “Sharing virtual machine disk images with other hosts”.

9.2. BENEFITS OF MIGRATING VIRTUAL MACHINES

Migrating virtual machines (VMs) can be useful for:

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

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

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

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

9.3. LIMITATIONS FOR MIGRATING VIRTUAL MACHINES

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

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

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

- VMs with assigned host devices will not work correctly if migrated, or the migration will fail. Such configurations include:
  - Device passthrough
  - SR-IOV device assignment
  - Mediated devices, such as vGPUs

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

- The emulated CPUs, both on the source VM and the destination VM, must be identical, otherwise the migration might fail. Any differences between the VMs in the following CPU related areas can cause problems with the migration:
  - CPU model
  - Firmware settings
  - Microcode version
  - BIOS version
9.4. SHARING VIRTUAL MACHINE DISK IMAGES WITH OTHER HOSTS

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

Prerequisites

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

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

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

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

Procedure

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

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

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

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

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

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

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

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

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

**Verification**

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

**9.5. MIGRATING A VIRTUAL MACHINE USING THE COMMAND-LINE INTERFACE**

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

**Prerequisites**

- The source host and the destination host both use the KVM hypervisor.
- The source host and the destination host are able to reach each other over the network. Use the `ping` utility to verify this.
- For the migration to be supportable by Red Hat, the source host and destination host must be using specific operating systems and machine types. To ensure this is the case, see Section 9.7, “Supported hosts for virtual machine migration”.
- The disk images of VMs that will be migrated are located on a separate networked location accessible to both the source host and the destination host. This is optional for offline migration, but required for migrating a running VM. For instructions to set up such shared VM storage, see Section 9.4, “Sharing virtual machine disk images with other hosts”.
- When migrating a running VM, your network bandwidth must be higher than the rate in which the VM generates dirty memory pages. To obtain the dirty page rate of your VM before you start the live migration, do the following:
  a. Monitor the rate of dirty page generation of the VM for a short period of time.
b. After the monitoring finishes, obtain its results:

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

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

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

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

- When performing a VM migration, the `virsh` client on the source host can use one of several protocols to connect to the libvirt daemon on the destination host. Examples in the following procedure use an SSH connection, but you can choose a different one.
  
  - If you want libvirt to use an SSH connection, ensure that the `virtqemud` socket is enabled and running on the destination host.

```
# systemctl enable --now virtqemud.socket
```

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

```
# systemctl enable --now virtproxyd-tls.socket
```

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

```
# systemctl enable --now virtproxyd-tcp.socket
```

Procedure

1. Use the `virsh migrate` command with options appropriate for your migration requirements.

- The following migrates the `wanderer1` VM from your local host to the system connection of the `dest-example` host using an SSH tunnel. The VM will remain running during the migration.

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

- The following enables you to make manual adjustments to the configuration of the `wanderer2` VM running on your local host, and then migrates the VM to the `dest-example` host. The migrated VM will automatically use the updated configuration.
This procedure can be useful for example when the destination host needs to use a different path to access the shared VM storage or when configuring a feature specific to the destination host.

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

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

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

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

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

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

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

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

**Verification**

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

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

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

**Troubleshooting**

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

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

```
# virsh suspend wanderer1
```

Additional resources

- For further options and examples for virtual machine migration, use `virsh migrate --help` or see the `virsh` man page.

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

If you wish to migrate a virtual machine (VM) that is performing tasks which require it to be constantly running, you can migrate that VM to another KVM host without shutting it down. This is also known as live migration. The following instructions explain how to do so using the web console.

**WARNING**

For tasks that modify memory pages faster than KVM can transfer them, such as heavy I/O load tasks, it is recommended that you do not live migrate the VM.

**Prerequisites**

- The web console VM plug-in is installed on your system.
- The source and destination hosts are running.
- The VM’s disk images are located on a shared storage that is accessible to the source host as well as the destination host.
- When migrating a running VM, your network bandwidth must be higher than the rate in which the VM generates dirty memory pages.

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

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

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

b. After the monitoring finishes, obtain its results:

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

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

**Procedure**

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

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

3. Enter the URI of the destination host.

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

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

**Verification**

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

- Confirm whether the VM appears in the list of VMs available on the destination host.
- Start the migrated VM and observe if it boots up.

### 9.7. SUPPORTED HOSTS FOR VIRTUAL MACHINE MIGRATION

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

**Table 9.2. Live migration compatibility**

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

**NOTE**

Support level is different for other virtualization solutions provided by Red Hat, including RHV, RHOSP, and OpenShift Virtualization.
One of the most effective ways to manage the functionality, features, and performance of a virtual machine (VM) is to adjust its virtual devices.

The following sections provide a general overview of what virtual devices are, and instructions on how they can be attached, modified, or removed from a VM.

### 10.1. HOW VIRTUAL DEVICES WORK

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

#### The basics

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

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

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

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

#### Performance or flexibility

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

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

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

For more information on types of virtual devices, see Section 10.6, “Types of virtual devices”.

### 10.2. VIEWING DEVICES ATTACHED TO VIRTUAL MACHINES USING THE WEB CONSOLE

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

#### Prerequisites

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

#### Procedure

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

2. Scroll to the **Host devices** section.

<table>
<thead>
<tr>
<th>Type</th>
<th>Class</th>
<th>Model</th>
<th>Vendor</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>usb</td>
<td></td>
<td>CHERRY Model</td>
<td>Cherry GmbH</td>
<td>Device 002</td>
</tr>
<tr>
<td>usb</td>
<td></td>
<td>Optical Mouse</td>
<td>Lenovo</td>
<td>Device 003</td>
</tr>
<tr>
<td>pci</td>
<td></td>
<td>Ethernet Connection (1)</td>
<td>Intel Corporation</td>
<td>Slot 0000:00:01</td>
</tr>
</tbody>
</table>

#### Additional resources

- For information about attaching or removing virtual devices, see Chapter 10, *Managing virtual devices*. 
10.3. ATTACHING DEVICES TO VIRTUAL MACHINES

You can add a specific functionality to your virtual machines (VMs) by attaching a new virtual device. For example, you can increase the storage capacity of a VM by attaching a new virtual disk device to it.

The following procedure demonstrates how to create and attach virtual devices to your virtual machines (VMs) using the command-line interface (CLI). Some devices can also be attached to VMs using the RHEL 8 web console.

Prerequisites

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

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

Procedure

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

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

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

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

```
# virt-xml testguest2 --add-device --update --hostdev 002.004
Device hotplug successful.
Domain 'testguest2' defined successfully.
```

   The bus-device combination for defining the USB can be obtained using the `lsusb` command.

Verification

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

- Use the `virsh dumpxml` command and see if the device’s XML definition has been added to the `<devices>` section in the VM’s XML configuration.

  For example, the following output shows the configuration of the `testguest` VM and confirms that the 002.004 USB flash disk device has been added.
# virsh dumpxml testguest

```xml
<hostdev mode='subsystem' type='usb' managed='yes'>
  <source>
    <vendor id='0x4146'/>
    <product id='0x902e'/>
    <address bus='2' device='4'/>
  </source>
  <alias name='hostdev0'/>
  <address type='usb' bus='0' port='3'/>
</hostdev>
```

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

### Additional resources

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

## 10.4. MODIFYING_devices_attached_to_virtual_machines

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

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

### Prerequisites

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

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

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

Verification

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

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

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

Troubleshooting

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

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

NOTE

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

Additional resources

- `man virt-xml`

10.5. REMOVING DEVICES FROM VIRTUAL MACHINES

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

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

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

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

Procedure

1. Use the `virt-xml --remove-device` command, including a definition of the device. For example:
   - The following removes the storage device marked as `vdb` from the running `testguest` VM after it shuts down:
     ```
     # virt-xml testguest --remove-device --disk target=vdb
     Domain 'testguest' defined successfully.
     Changes will take effect after the domain is fully powered off.
     ```
   - The following immediately removes a USB flash drive device from the running `testguest2` VM:
     ```
     # virt-xml testguest2 --remove-device --update --hostdev type=usb
     Device hotunplug successful.
     Domain 'testguest2' defined successfully.
     ```

Troubleshooting

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

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

Additional resources

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

10.6. TYPES OF VIRTUAL DEVICES

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

**Emulated devices**

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

The following types of emulated devices are supported:

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

Paravirtualized devices

Paravirtualization provides a fast and efficient method for exposing virtual devices to VMs. Paravirtualized devices expose interfaces that are designed specifically for use in VMs, and thus significantly increase device performance. RHEL 9 provides paravirtualized devices to VMs using the virtio API as a layer between the hypervisor and the VM. The drawback of this approach is that it requires a specific device driver in the guest operating system.

It is recommended to use paravirtualized devices instead of emulated devices for VM whenever possible, notably if they are running I/O intensive applications. Paravirtualized devices decrease I/O latency and increase I/O throughput, in some cases bringing them very close to bare-metal performance. Other paravirtualized devices also add functionality to VMs that is not otherwise available.

The following types of paravirtualized devices are supported:

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

Physically shared devices

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

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

The following types of passthrough devices are supported:

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

10.7. MANAGING VIRTUAL USB DEVICES

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

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

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

10.7.1. Attaching USB devices to virtual machines

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

Prerequisites

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

Procedure

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

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

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

**NOTE**

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

**Verification**

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

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

**Additional resources**

- `man virt-xml`

### 10.7.2. Removing USB devices from virtual machines

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

**Procedure**

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

```bash
# lsusb
[...]
Bus 001 Device 003: ID 2567:0a2b Intel Corp.
Bus 001 Device 005: ID 0407:6252 Kingston River 2.0
[...]
```

2. Use the `virt-xml` utility along with the `--remove-device` argument.
For example, the following command removes a USB flash drive, attached to the host as device 005 on bus 001, from the **Library** VM.

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

**NOTE**

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

**Verification**

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

**Additional resources**

- `man virt-xml`

---

### 10.7.3. Additional resources

- For information about managing other types of devices, see Section 10.3, "Attaching devices to virtual machines".

---

### 10.8. MANAGING VIRTUAL OPTICAL DRIVES

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

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

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

#### 10.8.1. Attaching optical drives to virtual machines

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

**Prerequisites**

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

**Procedure**

- Use the `virt-xml` utility with the **--add-device** argument.
For example, the following command attaches the **Doc10** ISO image, stored in the `/MC/tank/` directory, to the **DN1** VM.

```
# virt-xml DN1 --add-device --disk /MC/tank/Doc10.iso,device=cdrom
Domain 'DN1' defined successfully.
```

**Verification**

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

**Additional resources**

- man virt-xml

### 10.8.2. Replacing ISO images in virtual optical drives

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

**Prerequisites**

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

**Procedure**

1. Locate the target device where the CD-ROM is attached to the VM. You can find this information in the VM’s XML configuration file.
   For example, the following command displays the **DN1** VM’s XML configuration file, where the target device for CD-ROM is `sda`.
   ```
   # virsh dumpxml DN1
   ...
   <disk>
   ...
   <source file='/MC/tank/Doc10.iso'/>
   <target dev='sda' bus='sata'/>
   ...
   </disk>
   ...
   ```

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

**Verification**

- Run the VM and test if the device is replaced and works as expected.
10.8.3. Removing ISO images from virtual optical drives

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

Procedure

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

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

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

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

Verification

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

Additional resources

- man virt-xml

10.8.4. Removing optical drives from virtual machines

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

Procedure

1. Locate the target device where the CD-ROM is attached to the VM. You can find this information in the VM's XML configuration file.
   For example, the following command displays the DN1 VM's XML configuration file, where the target device for CD-ROM is sda.
2. Use the `virt-xml` utility with the `--remove-device` argument. For example, the following command removes the optical drive attached as target `sda` from the `DN1` VM.

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

**Verification**

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

**Additional resources**

- `man virt-xml`

### 10.9. MANAGING SR-IOV DEVICES

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

#### 10.9.1. What is SR-IOV?

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

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

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

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

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

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

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

**Figure 10.1. SR-IOV architecture**

Benefits

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

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

For example, a VF attached to a VM as a vNIC performs at almost the same level as a physical NIC, and much better than paravirtualized or emulated NICs. In particular, when multiple VFs are used simultaneously on a single host, the performance benefits can be significant.

Disadvantages
To modify the configuration of a PF, you must first change the number of VFs exposed by the PF to zero. Therefore, you also need to remove the devices provided by these VFs from the VM to which they are assigned.

A VM with an VFIO-assigned devices attached, including SR-IOV VFs, cannot be migrated to another host. In some cases, you can work around this limitation by pairing the assigned device with an emulated device. For example, you can bond an assigned networking VF to an emulated vNIC, and remove the VF before the migration.

In addition, VFIO-assigned devices require pinning of VM memory, which increases the memory consumption of the VM and prevents the use of memory ballooning on the VM.

Additional resources

- Section 10.9.3, “Supported devices for SR-IOV assignment”

10.9.2. Attaching SR-IOV networking devices to virtual machines

To attach an SR-IOV networking device to a virtual machine (VM) on an Intel or AMD host, you must create a virtual function (VF) from an SR-IOV capable network interface on the host and assign the VF as a device to a specified VM. For details, see the following instructions.

Prerequisites

- The CPU and the firmware of your host support the I/O Memory Management Unit (IOMMU).
  - If using an Intel CPU, it must support the Intel Virtualization Technology for Directed I/O (VT-d).
  - If using an AMD CPU, it must support the AMD-Vi feature.
- The host system uses Access Control Service (ACS) to provide direct memory access (DMA) isolation for PCIe topology. Verify this with the system vendor.
  For additional information, see Hardware Considerations for Implementing SR-IOV.
- The physical network device supports SR-IOV. To verify if any network devices on your system support SR-IOV, use the `lspci -v` command and look for Single Root I/O Virtualization (SR-IOV) in the output.

```
# lspci -v
[...]
02:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
Subsystem: Intel Corporation Gigabit ET Dual Port Server Adapter
Flags: bus master, fast devsel, latency 0, IRQ 16, NUMA node 0
Memory at fcba0000 (32-bit, non-prefetchable) [size=128K]
[...]
Capabilities: [150] Alternative Routing-ID Interpretation (ARI)
Capabilities: [160] Single Root I/O Virtualization (SR-IOV)
Kernel driver in use: igb
Kernel modules: igb
[...]
```

- The host network interface you want to use for creating VFs is running. For example, to activate the eth1 interface and verify it is running:
### For SR-IOV device assignment to work, the IOMMU feature must be enabled in the host BIOS and kernel. To do so:

- On an Intel host, enable VT-d:
  - If your Intel host uses multiple boot entries:
    - A. Edit the `/etc/default/grub` file and add the `intel_iommu=on` and `iommu=pt` parameters at the end of the `GRUB_CMDLINE_LINUX` line:
      ```
      GRUB_CMDLINE_LINUX="crashkernel=auto resume=/dev/mapper/rhel_dell-per730-27-swap rd.lvm.lv=rhel_dell-per730-27/root rd.lvm.lv=rhel_dell-per730-27swap console=ttyS0,115200n81 intel_iommu=on iommu=pt"
      ```
    - B. Regenerate the GRUB configuration:
      ```bash
      # grub2-mkconfig -o /boot/grub2/grub.cfg
      ```
    - C. Reboot the host.
  - If your Intel host uses a single boot entry:
    - A. Regenerate the GRUB configuration with the `intel_iommu=on iommu=pt` parameter:
      ```bash
      # grubby --args="intel_iommu=on iommu=pt" --update-kernel DEFAULT
      ```
    - B. Reboot the host.

- On an AMD host, enable AMD-Vi:
  - If your AMD host uses multiple boot entries:
    - A. Edit the `/etc/default/grub` file and add the `iommu=pt` parameter at the end of the `GRUB_CMDLINE_LINUX` line:
      ```
      GRUB_CMDLINE_LINUX="crashkernel=auto resume=/dev/mapper/rhel_dell-per730-27-swap rd.lvm.lv=rhel_dell-per730-27/root rd.lvm.lv=rhel_dell-per730-27swap console=ttyS0,115200n81 iommu=pt"
      ```
    - B. Regenerate the GRUB configuration:
      ```bash
      # grub2-mkconfig -o /boot/grub2/grub.cfg
      ```
    - C. Reboot the host.
If your AMD host uses a single boot entry:

A. Regenerate the GRUB configuration with the \texttt{iommu=pt} parameter:

\begin{verbatim}
# grubby --args="iommu=pt" --update-kernel DEFAULT
\end{verbatim}

B. Reboot the host.

Procedure

1. \textbf{Optional:} Confirm the maximum number of VFs your network device can use. To do so, use the following command and replace \textit{eth1} with your SR-IOV compatible network device.

\begin{verbatim}
# cat /sys/class/net/eth1/device/sriov_totalvfs
7
\end{verbatim}

2. Use the following command to create a virtual function (VF):

\begin{verbatim}
# echo VF-number > /sys/class/net/network-interface/device/sriov_numvfs
\end{verbatim}

In the command, replace:

- \textit{VF-number} with the number of VFs you want to create on the PF.
- \textit{network-interface} with the name of the network interface for which the VFs will be created.

The following example creates 2 VFs from the \textit{eth1} network interface:

\begin{verbatim}
# echo 2 > /sys/class/net/eth1/device/sriov_numvfs
\end{verbatim}

3. Verify the VFs have been added:

\begin{verbatim}
# lspci | grep Ethernet
82:00.0 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)
82:00.1 Ethernet controller: Intel Corporation 82599ES 10-Gigabit SFI/SFP+ Network Connection (rev 01)
82:10.0 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
82:10.2 Ethernet controller: Intel Corporation 82599 Ethernet Controller Virtual Function (rev 01)
\end{verbatim}

4. Make the created VFs persistent by creating a udev rule for the network interface you used to create the VFs. For example, for the \textit{eth1} interface, create the \texttt{/etc/udev/rules.d/eth1.rules} file, and add the following line:

\begin{verbatim}
ACTION=="add", SUBSYSTEM=="net", ENV{ID_NET_DRIVER}=="ixgbe",
ATTR{device/sriov_numvfs}=="2"
\end{verbatim}

This ensures that the two VFs that use the \texttt{ixgbe} driver will automatically be available for the \textit{eth1} interface when the host starts. If you do not require persistent SR-IOV devices, skip this step.
Currently, the setting described above does not work correctly when attempting to make VFs persistent on Broadcom NetXtreme II BCM57810 adapters. In addition, attaching VFs based on these adapters to Windows VMs is currently not reliable.

5. Hot-plug one of the newly added VF interface devices to a running VM.

```
# virsh attach-interface testguest1 hostdev 0000:82:10.0 --managed --live --config
```

**Verification**

- If the procedure is successful, the guest operating system detects a new network interface card.

### 10.9.3. Supported devices for SR-IOV assignment

Not all devices can be used for SR-IOV. The following devices have been tested and verified as compatible with SR-IOV in RHEL 9.

**Networking devices**

- Intel 82599ES 10 Gigabit Ethernet Controller - uses the `ixgbe` driver
- Intel Ethernet Controller XL710 Series - uses the `i40e` driver
- Mellanox ConnectX-5 Ethernet Adapter Cards - use the `mlx5_core` driver
- Intel Ethernet Network Adapter XXV710 - uses the `i40e` driver
- Intel 82576 Gigabit Ethernet Controller - uses the `igb` driver
- Broadcom NetXtreme II BCM57810 - uses the `bnx2x` driver

### 10.10. ATTACHING DASD DEVICES TO VIRTUAL MACHINES ON IBM Z

Using the `vfio-ccw` feature, you can assign direct-access storage devices (DASDs) as mediated devices to your virtual machines (VMs) on IBM Z hosts. This for example makes it possible for the VM to access a z/OS dataset, or to provide the assigned DASDs to a z/OS machine.

**Prerequisites**

- Your host system is using the IBM Z hardware architecture and supports the FICON protocol.
- The target VM is using a Linux guest operating system.
- The `mdevctl` package is installed.

```
# dnf install mdevctl
```
- The `driverctl` package is installed.
  
  ```
  # dnf install driverctl
  ```

- The necessary kernel modules have been loaded on the host. To verify, use:
  
  ```
  # lsmod | grep vfio
  ```

  The output should contain the following modules:

  - `vfio_ccw`
  - `vfio_mdev`
  - `vfio_iommu_type1`

- You have a spare DASD device for exclusive use by the VM, and you know the device’s identifier.

  This procedure uses **0.0.002c** as an example. When performing the commands, replace **0.0.002c** with the identifier of your DASD device.

**Procedure**

1. Obtain the subchannel identifier of the DASD device.

   ```
   # lscss -d 0.0.002c
   Device Subchan. DevType CU Type Use  PIM PAM POM  CHPIDs
   0.0.002c 0.0.29a8 3390/0c 3990/e9 yes 10 10 ff 02111221 00000000
   ```

   In this example, the subchannel identifier is detected as **0.0.29a8**. In the following commands of this procedure, replace **0.0.29a8** with the detected subchannel identifier of your device.

2. If the `lscss` command in the previous step only displayed the header output and no device information, perform the following steps:

   a. Remove the device from the `cio_ignore` list.

      ```
      # cio_ignore -r 0.0.002c
      ```

   b. In the guest OS, edit the kernel command line of the VM and add the device identifier with a ! mark to the line that starts with `cio_ignore=`, if it is not present already.

      ```
      cio_ignore=all,!condev,!0.0.002c
      ```

   c. Repeat step 1 on the host to obtain the subchannel identifier.

3. Bind the subchannel to the `vfio_ccw` passthrough driver.

   ```
   # driverctl -b css set-override 0.0.29a8 vfio_ccw
   ```
NOTE

This binds the 0.0.29a8 subchannel to vfio_ccw persistently, which means the DASD will not be usable on the host. If you need to use the device on the host, you must first remove the automatic binding to ‘vfio_ccw’ and rebind the subchannel to the default driver:

```
# driverctl -b css unset-override 0.0.29a8
```

4. Generate an UUID.

```
# uuidgen
30820a6f-b1a5-4503-91ca-0c10ba12345a
```

5. Create the DASD mediated device using the generated UUID.

```
# mdevctl start --uuid 30820a6f-b1a5-4503-91ca-0c10ba12345a --parent 0.0.29a8 --type vfio_ccw-io
```

6. Make the mediated device persistent.

```
# mdevctl define --auto --uuid 30820a6f-b1a5-4503-91ca-0c10ba12345a
```

7. Shut down the VM, if it is running.

8. Attach the mediated device to the VM. To do so, use the virsh edit utility to edit the XML configuration of the VM, add the following section to the XML, and replace the uuid value with the UUID you generated in the previous step.

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-ccw'>
    <source>
        <address uuid='30820a6f-b1a5-4503-91ca-0c10ba12345a'/>
    </source>
</hostdev>
```

Verification

1. Obtain the identifier that libvirtn assigned to the mediated DASD device. To do so, display the XML configuration of the VM and look for a vfio-ccw device.

```
# virsh dumpxml vm-name
```

```
<domain>
  [...]
  <hostdev mode='subsystem' type='mdev' managed='no' model='vfio-ccw'>
    <source>
        <address uuid='10620d2f-ed4d-437b-8aff-beda461541f9'/>
    </source>
    <alias name='hostdev0'/>
    <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0009'/>
  </hostdev>
  [...]
</domain>
```
In this example, the assigned identifier of the device is 0.0.0009.

2. Start the VM and log in to its guest OS.

3. In the guest OS, confirm that the DASD device is listed. For example:

   ```
   # lscss | grep 0.0.0009
   0.0.0009 0.0.0007 3390/0c 3990/e9 f0 f0 ff 12212231 00000000
   ```

4. In the guest OS, set the device online. For example:

   ```
   # chccwdv -e 0.0009
   Setting device 0.0.0009 online
   Done
   ```

Additional resources

- IBM documentation on cio_ignore
- Configuring kernel parameters at runtime
CHAPTER 11. CONFIGURING VIRTUAL MACHINE NETWORK CONNECTIONS

For your virtual machines (VMs) to connect over a network to your host, to other VMs on your host, and to locations on an external network, the VM networking must be configured accordingly. To provide VM networking, the RHEL 9 hypervisor and newly created VMs have a default network configuration, which can also be modified further. For example:

- You can enable the VMs on your host to be discovered and connected to by locations outside the host, as if the VMs were on the same network as the host.
- You can partially or completely isolate a VM from inbound network traffic to increase its security and minimize the risk of any problems with the VM impacting the host.

The following sections explain the various types of VM network configuration and provide instructions for setting up selected VM network configurations.

11.1. UNDERSTANDING VIRTUAL NETWORKING

The connection of virtual machines (VMs) to other devices and locations on a network has to be facilitated by the host hardware. The following sections explain the mechanisms of VM network connections and describe the default VM network setting.

11.1.1. How virtual networks work

Virtual networking uses the concept of a virtual network switch. A virtual network switch is a software construct that operates on a host machine. VMs connect to the network through the virtual network switch. Based on the configuration of the virtual switch, a VM can use an existing virtual network managed by the hypervisor, or a different network connection method.

The following figure shows a virtual network switch connecting two VMs to the network:

![Virtual Network Switch Diagram]

From the perspective of a guest operating system, a virtual network connection is the same as a physical network connection. Host machines view virtual network switches as network interfaces. When the libvirtd service is first installed and started, it creates virbr0, the default network interface for VMs.

To view information about this interface, use the ip utility on the host.
By default, all VMs on a single host are connected to the same NAT-type virtual network, named **default**, which uses the **virbr0** interface. For details, see Section 11.1.2, “Virtual networking default configuration”.

For basic outbound-only network access from VMs, no additional network setup is usually needed, because the default network is installed along with the **libvirt** package, and is automatically started when the **libvirtd** service is started.

If a different VM network functionality is needed, you can create additional virtual networks and network interfaces and configure your VMs to use them. In addition to the default NAT, these networks and interfaces can be configured to use one of the following modes:

- Routed mode
- Bridged mode
- Isolated mode
- Open mode

### 11.1.2. Virtual networking default configuration

When the **libvirtd** service is first installed on a virtualization host, it contains an initial virtual network configuration in network address translation (NAT) mode. By default, all VMs on the host are connected to the same **libvirt** virtual network, named **default**. VMs on this network can connect to locations both on the host and on the network beyond the host, but with the following limitations:

- VMs on the network are visible to the host and other VMs on the host, but the network traffic is affected by the firewalls in the guest operating system’s network stack and by the **libvirt** network filtering rules attached to the guest interface.

- VMs on the network can connect to locations outside the host but are not visible to them. Outbound traffic is affected by the NAT rules, as well as the host system’s firewall.

The following diagram illustrates the default VM network configuration:
11.2. USING THE WEB CONSOLE FOR MANAGING VIRTUAL MACHINE NETWORK INTERFACES

Using the RHEL 9 web console, you can manage the virtual network interfaces for the virtual machines to which the web console is connected. You can:

- View information about network interfaces and edit them.
- Add network interfaces to virtual machines, and disconnect or delete the interfaces.

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

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

**Prerequisites**

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

**Procedure**

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

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

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

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

- **Model type** - The model of the virtual network interface.

- **MAC Address** - The MAC address of the virtual network interface.

- **IP Address** - The IP address of the virtual network interface.

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

- **State** - The state of the virtual network interface.

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

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

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

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

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

### 11.2.2. Adding and connecting virtual network interfaces in the web console
Using the RHEL 9 web console, you can create a virtual network interface and connect a virtual machine (VM) to it.

**Prerequisites**

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

**Procedure**

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

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

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

### 11.2.3. Disconnecting and removing virtual network interfaces in the web console

Using the RHEL 9 web console, you can disconnect the virtual network interfaces connected to a selected virtual machine (VM).

**Prerequisites**

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

**Procedure**

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

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

3. Click Unplug in the row of the virtual network interface you want to disconnect. The selected virtual network interface disconnects from the VM.

### 11.3. RECOMMENDED VIRTUAL MACHINE NETWORKING CONFIGURATIONS USING THE COMMAND-LINE INTERFACE

In many scenarios, the default VM networking configuration is sufficient. However, if adjusting the configuration is required, you can use the command-line interface (CLI) to do so. The following sections describe selected VM network setups for such situations.
11.3.1. Configuring externally visible virtual machines using the command-line interface

By default, a newly created VM connects to a NAT-type network that uses virbr0, the default virtual bridge on the host. This ensures that the VM can use the host's network interface controller (NIC) for connecting to outside networks, but the VM is not reachable from external systems.

If you require a VM to appear on the same external network as the hypervisor, you must use bridged mode instead. To do so, attach the VM to a bridge device connected to the hypervisor’s physical network device. To use the command-line interface for this, follow the instructions below.

Prerequisites

- A shut-down existing VM with the default NAT setup.
- The IP configuration of the hypervisor. This varies depending on the network connection of the host. As an example, this procedure uses a scenario where the host is connected to the network using an ethernet cable, and the hosts' physical NIC MAC address is assigned to a static IP on a DHCP server. Therefore, the ethernet interface is treated as the hypervisor IP.

To obtain the IP configuration of the ethernet interface, use the `ip addr` utility:

```bash
# ip addr
[...]
enp0s25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
    group default qlen 1000
    link/ether 54:ee:75:49:dc:46 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.148/24 brd 10.0.0.255 scope global dynamic noprefixroute enp0s25
```

Procedure

1. Create and set up a bridge connection for the physical interface on the host. For instructions, see the Configuring a network bridge.
   Note that in a scenario where static IP assignment is used, you must move the IPv4 setting of the physical ethernet interface to the bridge interface.

2. Modify the VM’s network to use the created bridged interface. For example, the following sets testguest to use bridge0.

   ```bash
   # virt-xml testguest --edit --network bridge=bridge0
   Domain 'testguest' defined successfully.
   ```

3. Start the VM.

   ```bash
   # virsh start testguest
   ```

4. In the guest operating system, adjust the IP and DHCP settings of the system’s network interface as if the VM was another physical system in the same network as the hypervisor. The specific steps for this will differ depending on the guest OS used by the VM. For example, if the guest OS is RHEL 9, see Configuring an Ethernet connection.

Verification

1. Ensure the newly created bridge is running and contains both the host’s physical interface and the interface of the VM.
# ip link show master bridge0
2: enp0s25: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master bridge0 state UP mode DEFAULT group default qlen 1000
   link/ether 54:ee:75:49:dc:46 brd ff:ff:ff:ff:ff:ff
10: vnet0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master bridge0 state UNKNOWN mode DEFAULT group default qlen 1000
   link/ether fe:54:00:89:15:40 brd ff:ff:ff:ff:ff:ff

2. Ensure the VM appears on the same external network as the hypervisor:
   a. In the guest operating system, obtain the network ID of the system. For example, if it is a Linux guest:

   ```
   # ip addr
   [...]  
enp0s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
   link/ether 52:54:00:09:15:46 brd ff:ff:ff:ff:ff:ff
   inet 10.0.0.150/24 brd 10.0.0.255 scope global dynamic noprefixroute enp0s0
   ```

   b. From an external system connected to the local network, connect to the VM using the obtained ID.

   ```
   # ssh root@10.0.0.150
   root@10.0.0.150's password: 
   Last login: Mon Sep 24 12:05:36 2019
   root~#*
   ```

   If the connection works, the network has been configured successfully.

Additional resources

- For instructions on creating an externally visible VM using the web console, see Section 11.4.1, "Configuring externally visible virtual machines using the web console".

- For additional information on bridged mode, see Section 11.5.3, "Virtual networking in bridged mode".

- In certain situations, such as when a using client-to-site VPN while the VM is hosted on the client, using bridged mode for making your VMs available to external locations is not possible. To work around this problem, you can set a destination NAT for the VM. For details, see the Configuring and managing networking document.

11.3.2. Isolating virtual machines from each other using the command-line interface

To prevent a virtual machine (VM) from communicating with other VMs on your host, for example to avoid data sharing or to increase system security, you can completely isolate the VM from host-side network traffic.

By default, a newly created VM connects to a NAT-type network that uses `virbr0`, the default virtual bridge on the host. This ensures that the VM can use the host’s NIC for connecting to outside networks, as well as to other VMs on the host. This is a generally secure connection, but in some cases, connectivity to the other VMs may be a security or data privacy hazard. In such situations, you can isolate the VM by using direct `macvtap` connection in private mode instead of the default network.
In private mode, the VM is visible to external systems and can receive a public IP on the host’s subnet, but the VM and the host cannot access each other, and the VM is also not visible to other VMs on the host.

For instructions to set up **macvtap** private mode on your VM using the CLI, see below.

**Prerequisites**

- An **existing VM** with the default NAT setup.
- The name of the host interface that you want to use for the **macvtap** connection. The interface you must select will vary depending on your use case and the network configuration on your host. As an example, this procedure uses the host’s physical ethernet interface.

  To obtain the name of the targeted interface:

  ```
  $ ip addr
  [...]  
  2: enp0s31f6: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel
  state DOWN group default qlen 1000
  link/ether 54:e1:ad:42:70:45 brd ff:ff:ff:ff:ff:ff
  [...]  
  ```

**Procedure**

- Use the selected interface to set up private **macvtap** on the selected VM. The following example configures **macvtap** in private mode on the **enp0s31f6** interface for the VM named `panic-room`.

  ```
  # virt-xml panic-room --edit --network
type=direct,source=enp0s31f6,source.mode=private
  Domain panic-room XML defined successfully
  ```

**Verification**

1. Start the updated VM.

   ```
   # virsh start panic-room
   Domain panic-room started
   ```

2. List the interface statistics for the VM.

   ```
   # virsh domstats panic-room --interface
   Domain: 'panic-room'
   net.count=1
   net.0.name=macvtap0
   net.0.rx.bytes=0
   net.0.rx.pkts=0
   net.0.rx.errs=0
   net.0.rx.drop=0
   net.0.tx.bytes=0
   net.0.tx.pkts=0
   net.0.tx.errs=0
   net.0.tx.drop=0
   ```
If the command displays similar output, the VM has been isolated successfully.

Additional resources

- For instructions on isolating a VM using the web console, see Section 11.4.2, “Isolating virtual machines from each other using the web console”.
- For additional information about macvtap private mode, see Section 11.5.6, “Direct attachment of the virtual network device”.
- For additional security measures that you can set on a VM, see Securing virtual machines in RHEL 9.

11.4. RECOMMENDED VIRTUAL MACHINE NETWORKING CONFIGURATIONS USING THE WEB CONSOLE

In many scenarios, the default VM networking configuration is sufficient. However, if adjusting the configuration is required, you can use the RHEL 9 web console to do so. The following sections describe selected VM network setups for such situations.

11.4.1. Configuring externally visible virtual machines using the web console

By default, a newly created VM connects to a NAT-type network that uses virbr0, the default virtual bridge on the host. This ensures that the VM can use the host’s network interface controller (NIC) for connecting to outside networks, but the VM is not reachable from external systems.

If you require a VM to appear on the same external network as the hypervisor, you must use bridged mode instead. To do so, attach the VM to a bridge device connected to the hypervisor’s physical network device. To use the RHEL 9 web console for this, follow the instructions below.

Prerequisites

- The web console VM plug-in is installed on your system.
- A shut-down existing VM with the default NAT setup.
- The IP configuration of the hypervisor. This varies depending on the network connection of the host. As an example, this procedure uses a scenario where the host is connected to the network using an ethernet cable, and the hosts’ physical NIC MAC address is assigned to a static IP on a DHCP server. Therefore, the ethernet interface is treated as the hypervisor IP.

To obtain the IP configuration of the ethernet interface, go to the Networking tab in the web console, and see the Interfaces section.

Procedure

1. Create and set up a bridge connection for the physical interface on the host. For instructions, see Configuring network bridges in the web console.
   Note that in a scenario where static IP assignment is used, you must move the IPv4 setting of the physical ethernet interface to the bridge interface.

2. Modify the VM’s network to use the bridged interface. In the Network Interfaces tab of the VM:
   a. Click Add Network Interface
b. In the **Add Virtual Network Interface** dialog, set:
   - **Interface Type** to **Bridge to LAN**
   - **Source** to the newly created bridge, for example **bridge0**

c. Click **Add**

d. **Optional**: Click **Unplug** for all the other interfaces connected to the VM.

3. Click **Run** to start the VM.

4. In the guest operating system, adjust the IP and DHCP settings of the system’s network interface as if the VM was another physical system in the same network as the hypervisor. The specific steps for this will differ depending on the guest OS used by the VM. For example, if the guest OS is RHEL 9, see [Configuring an Ethernet connection](#).

**Verification**

1. In the **Networking** tab of the host’s web console, click the row with the newly created bridge to ensure it is running and contains both the host’s physical interface and the interface of the VM.

2. Ensure the VM appears on the same external network as the hypervisor.

   a. In the guest operating system, obtain the network ID of the system. For example, if it is a Linux guest:

   ```
   # ip addr
   [...] enp0s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
   link/ether 52:54:00:09:15:46 brd ff:ff:ff:ff:ff:ff
   inet 10.0.0.150/24 brd 10.0.0.255 scope global dynamic noprefixroute enp0s0
   ```

   b. From an external system connected to the local network, connect to the VM using the obtained ID.

   ```
   # ssh root@10.0.0.150
   root@110.34.5.18's password:
   Last login: Mon Sep 24 12:05:36 2019
   root~#*
   ```

   If the connection works, the network has been configured successfully.

**Additional resources**

- For instructions on creating an externally visible VM using the CLI, see Section 11.3.1, “Configuring externally visible virtual machines using the command-line interface”.

- For additional information on bridged mode, see Section 11.5.3, “Virtual networking in bridged mode”.

- In certain situations, such as when a using client-to-site VPN while the VM is hosted on the client, using bridged mode for making your VMs available to external locations is not possible. To work around this problem, you can set a destination NAT for the VM. For details, see the [Configuring and managing networking](#) document.
11.4.2. Isolating virtual machines from each other using the web console

To prevent a virtual machine (VM) from communicating with other VMs on your host, for example to avoid data sharing or to increase system security, you can completely isolate the VM from host-side network traffic.

By default, a newly created VM connects to a NAT-type network that uses `virbr0`, the default virtual bridge on the host. This ensures that the VM can use the host’s NIC for connecting to outside networks, as well as to other VMs on the host. This is a generally secure connection, but in some cases, connectivity to the other VMs may be a security or data privacy hazard. In such situations, you can isolate the VM by using direct `macvtap` connection in private mode instead of the default network.

In private mode, the VM is visible to external systems and can receive a public IP on the host’s subnet, but the VM and the host cannot access each other, and the VM is also not visible to other VMs on the host.

For instructions to set up `macvtap` private mode on your VM using the web console, see below.

Prerequisites

- The web console VM plug-in is installed on your system.
- An existing VM with the default NAT setup.

Procedure

1. In the Virtual Machines pane, click the row with the virtual machine you want to isolate. A pane with the basic information about the VM opens.

2. Click the Network Interfaces tab.

3. Click Edit. The Virtual Machine Interface Settings dialog opens.

4. Set Interface Type to Direct Attachment

5. Set Source to the host interface of your choice. Note that the interface you select will vary depending on your use case and the network configuration on your host.

Verification

1. Start the VM by clicking Run.

2. In the Terminal pane of the web console, list the interface statistics for the VM. For example, to view the network interface traffic for the `panic-room` VM:

```
# virsh domstats panic-room --interface
Domain: 'panic-room'
  net.count=1
  net.0.name=macvtap0
  net.0.rx.bytes=0
  net.0.rx.pkts=0
  net.0.rx.errs=0
  net.0.rx.drop=0
  net.0.tx.bytes=0
```
net.0.tx.pkts=0
net.0.tx.errs=0
net.0.tx.drop=0

If the command displays similar output, the VM has been isolated successfully.

Additional resources

- For instructions on isolating a VM using the command-line, see Section 11.3.2, “Isolating virtual machines from each other using the command-line interface”.

- For additional information about macvtap private mode, see Section 11.5.6, “Direct attachment of the virtual network device”.

- For additional security measures that you can set on a VM, see Securing virtual machines in RHEL 9.

11.5. TYPES OF VIRTUAL MACHINE NETWORK CONNECTIONS

To modify the networking properties and behavior of your VMs, change the type of virtual network or interface the VMs use. The following sections describe the connection types available to VMs in RHEL 9.

11.5.1. Virtual networking with network address translation

By default, virtual network switches operate in network address translation (NAT) mode. They use IP masquerading rather than Source-NAT (SNAT) or Destination-NAT (DNAT). IP masquerading enables connected VMs to use the host machine’s IP address for communication with any external network. When the virtual network switch is operating in NAT mode, computers external to the host cannot communicate with the VMs inside the host.
11.5.2. Virtual networking in routed mode

When using *Routed* mode, the virtual switch connects to the physical LAN connected to the host machine, passing traffic back and forth without the use of NAT. The virtual switch can examine all traffic and use the information contained within the network packets to make routing decisions. When using this mode, the virtual machines (VMs) are all in a single subnet, separate from the host machine. The VM subnet is routed through a virtual switch, which exists on the host machine. This enables incoming connections, but requires extra routing-table entries for systems on the external network.

Routed mode uses routing based on the IP address:

Common topologies that use routed mode include DMZs and virtual server hosting.

**DMZ**

You can create a network where one or more nodes are placed in a controlled sub-network for security reasons. Such a sub-network is known as a demilitarized zone (DMZ).
Host machines in a DMZ typically provide services to WAN (external) host machines as well as LAN (internal) host machines. Since this requires them to be accessible from multiple locations, and considering that these locations are controlled and operated in different ways based on their security and trust level, routed mode is the best configuration for this environment.

**Virtual server hosting**

A virtual server hosting provider may have several host machines, each with two physical network connections. One interface is used for management and accounting, the other for the VMs to connect through. Each VM has its own public IP address, but the host machines use private IP addresses so that only internal administrators can manage the VMs.

**11.5.3. Virtual networking in bridged mode**

In most VM networking modes, VMs automatically create and connect to the `virbr0` virtual bridge. In contrast, in bridged mode, the VM connects to an existing Linux bridge on the host. As a result, the VM is directly visible on the physical network. This enables incoming connections, but does not require any extra routing-table entries.

Bridged mode uses connection switching based on the MAC address:
In bridged mode, the VM appear within the same subnet as the host machine. All other physical machines on the same physical network can detect the VM and access it.

**Bridged network bonding**

It is possible to use multiple physical bridge interfaces on the hypervisor by joining them together with a bond. The bond can then be added to a bridge, after which the VMs can be added to the bridge as well. However, the bonding driver has several modes of operation, and not all of these modes work with a bridge where VMs are in use.

The following bonding modes are usable:

- mode 1
- mode 2
- mode 4

In contrast, using modes 0, 3, 5, or 6 is likely to cause the connection to fail. Also note that media-independent interface (MII) monitoring should be used to monitor bonding modes, as Address Resolution Protocol (ARP) monitoring does not work correctly.

For more information on bonding modes, refer to the Red Hat Knowledgebase.

**Common scenarios**

The most common use cases for bridged mode include:

- Deploying VMs in an existing network alongside host machines, making the difference between virtual and physical machines invisible to the end user.
- Deploying VMs without making any changes to existing physical network configuration settings.
- Deploying VMs that must be easily accessible to an existing physical network. Placing VMs on a physical network where they must access DHCP services.
- Connecting VMs to an existing network where virtual LANs (VLANs) are used.

**Additional resources**
For instructions on configuring your VMs to use bridged mode, see Configuring externally visible virtual machines using the command-line interface or Configuring externally visible virtual machines using the web console.

For a detailed explanation of bridge_opts parameters, used to configure bridged networking mode, see the Red Hat Virtualization Administration Guide.

11.5.4. Virtual networking in isolated mode

When using isolated mode, virtual machines connected to the virtual switch can communicate with each other and with the host machine, but their traffic will not pass outside of the host machine, and they cannot receive traffic from outside the host machine. Using dnsmasq in this mode is required for basic functionality such as DHCP.

11.5.5. Virtual networking in open mode

When using open mode for networking, libvirt does not generate any firewall rules for the network. As a result, libvirt does not overwrite firewall rules provided by the host, and the user can therefore manually manage the VM’s firewall rules.

11.5.6. Direct attachment of the virtual network device

You can use the macvtap driver to attach a virtual machine’s NIC directly to a specified physical interface of the host machine. The macvtap connection has a number of modes, including private mode.

In this mode, all packets are sent to the external switch and will only be delivered to a target VM on the same host machine if they are sent through an external router or gateway and these send them back to the host. Private mode can be used to prevent the individual VMs on a single host from communicating with each other.
Additional resources

- For instructions on configuring your VMs to use `macvtap` in private mode, see Isolating virtual machines from each other using the command-line interface or Isolating virtual machines from each other using the web console.

11.5.7. Comparison of virtual machine connection types

The following table provides information about the locations to which selected types of virtual machine (VM) network configurations can connect, and to which they are visible.

**Table 11.1. Virtual machine connection types**

<table>
<thead>
<tr>
<th>Connection to the host</th>
<th>Connection to other VMs on the host</th>
<th>Connection to outside locations</th>
<th>Visible to outside locations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bridged mode</strong></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td><strong>NAT</strong></td>
<td>YES</td>
<td>YES</td>
<td>no</td>
</tr>
<tr>
<td><strong>Routed mode</strong></td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td><strong>Isolated mode</strong></td>
<td>YES</td>
<td>YES</td>
<td>no</td>
</tr>
<tr>
<td><strong>Private mode</strong></td>
<td>no</td>
<td>no</td>
<td>YES</td>
</tr>
<tr>
<td><strong>Open mode</strong></td>
<td><em>Depends on the host’s firewall rules</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

11.6. BOOTING VIRTUAL MACHINES FROM A PXE SERVER

Virtual machines (VMs) that use Preboot Execution Environment (PXE) can boot and load their configuration from a network. This chapter describes how to use `libvirt` to boot VMs from a PXE server on a virtual or bridged network.
11.6.1. Setting up a PXE boot server on a virtual network

This procedure describes how to configure a *libvirt* virtual network to provide Preboot Execution Environment (PXE). This enables virtual machines on your host to be configured to boot from a boot image available on the virtual network.

**Prerequisites**

- A local PXE server (DHCP and TFTP), such as:
  - *libvirt* internal server
  - manually configured dhcpd and tftpd
  - *dnsmasq*
  - Cobbler server
- PXE boot images, such as *PXELINUX* configured by Cobbler or manually.

**Procedure**

1. Place the PXE boot images and configuration in `/var/lib/tftpboot` folder.
2. Set folder permissions:
   ```bash
   # chmod -R a+r /var/lib/tftpboot
   ```
3. Set folder ownership:
   ```bash
   # chown -R nobody: /var/lib/tftpboot
   ```
4. Update SELinux context:
   ```bash
   # chcon -R --reference /usr/sbin/dnsmasq /var/lib/tftpboot
   # chcon -R --reference /usr/libexec/libvirt_leaseshelper /var/lib/tftpboot
   ```
5. Shut down the virtual network:
   ```bash
   # virsh net-destroy default
   ```
6. Open the virtual network configuration file in your default editor:
   ```bash
   # virsh net-edit default
   ```
7. Edit the `<ip>` element to include the appropriate address, network mask, DHCP address range, and boot file, where `BOOT_FILENAME` is the name of the boot image file.

```
<ip address='192.168.122.1' netmask='255.255.255.0'>
  <tftp root='/var/lib/tftpboot'/>
  <dhcp>
    <range start='192.168.122.2' end='192.168.122.254'/>
    <bootp file='BOOT_FILENAME'/>
  </dhcp>
</ip>
```

8. Start the virtual network:

```
# virsh net-start default
```

**Verification**

- Verify that the `default` virtual network is active:

```
# virsh net-list

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
<th>Autostart</th>
<th>Persistent</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>active</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
```

**Additional resources**

- For more information about configuring TFTP and DHCP on a PXE server, see [*Preparing to install from the network using PXE*](#).

### 11.6.2. Booting virtual machines using PXE and a virtual network

To boot virtual machines (VMs) from a Preboot Execution Environment (PXE) server available on a virtual network, you must enable PXE booting.

**Prerequisites**

- A PXE boot server is set up on the virtual network as described in [*Section 11.6.1, “Setting up a PXE boot server on a virtual network”*](#).

**Procedure**

- Create a new VM with PXE booting enabled. For example, to install from a PXE, available on the `default` virtual network, into a new 10 GB qcow2 image file:

  ```
  # virt-install --pxe --network network=default --memory 2048 --vcpus 2 --disk size=10
  ```

- Alternatively, you can manually edit the XML configuration file of an existing VM:

  i. Ensure the `<os>` element has a `<boot dev='network'/>` element inside:

```
<os>
  <type arch='x86_64' machine='pc-i440fx-rhel7.0.0'>hvm</type>
  <boot dev='network'/>
</os>
```
ii. Ensure the guest network is configured to use your virtual network:

```xml
<interface type='network'>
  <mac address='52:54:00:66:79:14'/>
  <source network='default'/>
  <target dev='vnet0'/>
  <alias name='net0'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

**Verification**

- Start the VM using the `virsh start` command. If PXE is configured correctly, the VM boots from a boot image available on the PXE server.

### 11.6.3. Booting virtual machines using PXE and a bridged network

To boot virtual machines (VMs) from a Preboot Execution Environment (PXE) server available on a bridged network, you must enable PXE booting.

**Prerequisites**

- Network bridging is enabled.
- A PXE boot server is available on the bridged network.

**Procedure**

- Create a new VM with PXE booting enabled. For example, to install from a PXE, available on the `breth0` bridged network, into a new 10 GB qcow2 image file:

  ```bash
  # virt-install --pxe --network bridge=breth0 --memory 2048 --vcpus 2 --disk size=10
  ```

  Alternatively, you can manually edit the XML configuration file of an existing VM:

  i. Ensure the `<os>` element has a `<boot dev='network'/>` element inside:

  ```xml
  <os>
    <type arch='x86_64' machine='pc-i440fx-rhel7.0.0'>hvm</type>
    <boot dev='network'/>
    <boot dev='hd'/>
  </os>
  ```

  ii. Ensure the VM is configured to use your bridged network:

  ```xml
  <interface type='bridge'>
    <mac address='52:54:00:5a:ad:cb'/>
    <source bridge='breth0'/>
    <target dev='vnet0'/>
  </interface>
  ```
Verification

- Start the VM using the **virsh start** command. If PXE is configured correctly, the VM boots from a boot image available on the PXE server.

Additional resources

- For more information about bridged networking, see [*Configuring a network bridge*](#).

### 11.7. ADDITIONAL RESOURCES

- For additional information on networking configurations in RHEL 9, see the [*Configuring and managing networking*](#) document.

- Specific network interface cards can be attached to VMs as SR-IOV devices, which increases their performance. For details, see [*Managing SR-IOV devices*](#).
CHAPTER 12. OPTIMIZING VIRTUAL MACHINE PERFORMANCE

Virtual machines (VMs) always experience some degree of performance deterioration in comparison to the host. The following sections explain the reasons for this deterioration and provide instructions on how to minimize the performance impact of virtualization in RHEL 9, so that your hardware infrastructure resources can be used as efficiently as possible.

12.1. WHAT INFLUENCES VIRTUAL MACHINE PERFORMANCE

VMs are run as user-space processes on the host. The hypervisor therefore needs to convert the host’s system resources so that the VMs can use them. As a consequence, a portion of the resources is consumed by the conversion, and the VM therefore cannot achieve the same performance efficiency as the host.

The impact of virtualization on system performance

More specific reasons for VM performance loss include:

- Virtual CPUs (vCPUs) are implemented as threads on the host, handled by the Linux scheduler.
- VMs do not automatically inherit optimization features, such as NUMA or huge pages, from the host kernel.
- Disk and network I/O settings of the host might have a significant performance impact on the VM.
- Network traffic typically travels to a VM through a software-based bridge.
- Depending on the host devices and their models, there might be significant overhead due to emulation of particular hardware.

The severity of the virtualization impact on the VM performance is influenced by a variety factors, which include:

- The number of concurrently running VMs.
- The amount of virtual devices used by each VM.
- The device types used by the VMs.

Reducing VM performance loss

RHEL 9 provides a number of features you can use to reduce the negative performance effects of virtualization. Notably:

- The tuned service can automatically optimize the resource distribution and performance of your VMs.
- Block I/O tuning can improve the performances of the VM’s block devices, such as disks.
- NUMA tuning can increase vCPU performance.
- Virtual networking can be optimized in various ways.

IMPORTANT

Tuning VM performance can have adverse effects on other virtualization functions. For example, it can make migrating the modified VM more difficult.
12.2. OPTIMIZING VIRTUAL MACHINE PERFORMANCE USING TUNED

The tuned utility is a tuning profile delivery mechanism that adapts RHEL for certain workload characteristics, such as requirements for CPU-intensive tasks or storage-network throughput responsiveness. It provides a number of tuning profiles that are pre-configured to enhance performance and reduce power consumption in a number of specific use cases. You can edit these profiles or create new profiles to create performance solutions tailored to your environment, including virtualized environments.

To optimize RHEL 9 for virtualization, use the following profiles:

- For RHEL 9 virtual machines, use the virtual-guest profile. It is based on the generally applicable throughput-performance profile, but also decreases the swappiness of virtual memory.

- For RHEL 9 virtualization hosts, use the virtual-host profile. This enables more aggressive writeback of dirty memory pages, which benefits the host performance.

Prerequisites

- The tuned service is installed and enabled.

Procedure

To enable a specific tuned profile:

1. List the available tuned profiles.

```
# 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.

```
# 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.

12.3. CONFIGURING VIRTUAL MACHINE MEMORY

To improve the performance of a virtual machine (VM), you can assign additional host RAM to the VM. Similarly, you can decrease the amount of memory allocated to a VM so the host memory can be allocated to other VMs or tasks.

To perform these actions, you can use the web console or the command-line interface.

12.3.1. Adding and removing virtual machine memory using the web console

To improve the performance of a virtual machine (VM) or to free up the host resources it is using, you can use the web console to adjust amount of memory allocated to the VM.

Prerequisites

- The guest OS is running the memory balloon drivers. To verify this is the case:
  1. Ensure the VM’s configuration includes the memballoon device:

```
# virsh dumpxml testguest | grep memballoon
<memballoon model="virtio">
  </memballoon>
```

If this command displays any output and the model is not set to none, the memballoon device is present.

- Ensure the balloon drivers are running in the guest OS.
  - In Windows guests, the drivers are installed as a part of the virtio-win driver package. For instructions, see Installing paravirtualized KVM drivers for Windows virtual machines.
  - In Linux guests, the drivers are generally included by default and activate when the memballoon device is present.

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

Procedure

1. **Optional:** Obtain the information about the maximum memory and currently used memory for a VM. This will serve as a baseline for your changes, and also for verification.

```
# virsh dominfo testguest
Max memory: 2097152 KiB
Used memory: 2097152 KiB
```

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

3. Click edit next to the Memory line in the Overview pane. The Memory Adjustment dialog appears.
4. Configure the virtual CPUs for the selected VM.

   - **Maximum allocation** - Sets the maximum amount of host memory that the VM can use for its processes. You can specify the maximum memory when creating the VM or increase it later. You can specify memory as multiples of MiB or GiB. Adjusting maximum memory allocation is only possible on a shut-off VM.

   - **Current allocation** - Sets the actual amount of memory allocated to the VM. This value can be less than the Maximum allocation but cannot exceed it. You can adjust the value to regulate the memory available to the VM for its processes. You can specify memory as multiples of MiB or GiB. If you do not specify this value, the default allocation is the Maximum allocation value.

5. Click **Save**.

   The memory allocation of the VM is adjusted.

**Additional resources**

- For instructions for adjusting VM memory setting using the command-line interface, see Section 12.3.2, "Adding and removing virtual machine memory using the command-line interface".

- To optimize how the VM uses the allocated memory, you can modify your vCPU setting. For more information, see Section 12.5, “Optimizing virtual machine CPU performance”.

### 12.3.2. Adding and removing virtual machine memory using the command-line interface

To improve the performance of a virtual machine (VM) or to free up the host resources it is using, you can use the CLI to adjust amount of memory allocated to the VM.

**Prerequisites**

- The guest OS is running the memory balloon drivers. To verify this is the case:

  1. Ensure the VM’s configuration includes the `memballoon` device:

     ```
     # virsh dumpxml testguest | grep membaloion
     <memballoion model='virtio'>
     </memballoion>
     ```

     If this commands displays any output and the model is not set to `none`, the `memballoion` device is present.
2. Ensure the balloon drivers are running in the guest OS.

- In Windows guests, the drivers are installed as a part of the `virtio-win` driver package. For instructions, see Installing paravirtualized KVM drivers for Windows virtual machines.

- In Linux guests, the drivers are generally included by default and activate when the `memballoon` device is present.

### Procedure

1. **Optional:** Obtain the information about the maximum memory and currently used memory for a VM. This will serve as a baseline for your changes, and also for verification.

   ```
   # virsh dominfo testguest
   Max memory: 2097152 KiB
   Used memory: 2097152 KiB
   ```

2. Adjust the maximum memory allocated to a VM. Increasing this value improves the performance potential of the VM, and reducing the value lowers the performance footprint the VM has on your host. Note that this change can only be performed on a shut-off VM, so adjusting a running VM requires a reboot to take effect.

   For example, to change the maximum memory that the `testguest` VM can use to 4096 MiB:

   ```
   # virt-xml testguest --edit --memory memory=4096,currentMemory=4096
   Domain 'testguest' defined successfully.
   Changes will take effect after the domain is fully powered off.
   ```

1. **Optional:** You can also adjust the memory currently used by the VM, up to the maximum allocation. This regulates the memory load that the VM has on the host until the next reboot, without changing the maximum VM allocation.

   ```
   # virsh setmem testguest --current 2048
   ```

### Verification

1. Confirm that the memory used by the VM has been updated:

   ```
   # virsh dominfo testguest
   Max memory: 4194304 KiB
   Used memory: 2097152 KiB
   ```

2. **Optional:** If you adjusted the current VM memory, you can obtain the memory balloon statistics of the VM to evaluate how effectively it regulates its memory use.

   ```
   # virsh domstats --balloon testguest
   Domain: 'testguest'
   balloon.current=365624
   balloon.maximum=4194304
   balloon.swap_in=0
   balloon.swap_out=0
   balloon.major_fault=306
   balloon.minor_fault=156117
   balloon.unused=3834448
   balloon.available=4035008
   ```
balloon.usable=3746340
balloon.last-update=1587971682
balloon.disk_caches=75444
balloon.hugetlb_pgalloc=0
balloon.hugetlb_pgfail=0
balloon.rss=1005456

Additional resources

- For instructions for adjusting VM memory setting using the web console, see Section 12.3.1, “Adding and removing virtual machine memory using the web console”.
- To optimize how the VM uses the allocated memory, you can modify your vCPU setting. For more information, see Section 12.5, “Optimizing virtual machine CPU performance”.

12.3.3. Additional resources

- To increase the maximum memory of a running VM, you can attach a memory device to the VM. This is also referred to as memory hot plug. For details, see Attaching devices to virtual machines
  Note that removing a memory device from a VM, also known as memory hot unplug, is not supported in RHEL 9, and Red Hat highly discourages its use.

12.4. OPTIMIZING VIRTUAL MACHINE I/O PERFORMANCE

The input and output (I/O) capabilities of a virtual machine (VM) can significantly limit the VM’s overall efficiency. To address this, you can optimize a VM’s I/O by configuring block I/O parameters.

12.4.1. 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:
   ```bash
   # virsh dumpxml VM-name
   ```
   ```xml
   <domain>
   [...]
   <blkiotune>
   <weight>800</weight>
   <device>
   <path>/dev/sda</path>
   ```
2. Edit the I/O weight of a specified device:

```
# virsh blkiotune VM-name --device-weights device, I/O-weight
```

For example, the following changes the weight of the `/dev/sda` device in the `liftrul` VM to 500.

```
# virsh blkiotune liftrul --device-weights /dev/sda, 500
```

### 12.4.2. Disk I/O throttling in virtual machines

When several VMs are running simultaneously, they can interfere with system performance by using excessive disk I/O. Disk I/O throttling in KVM virtualization provides the ability to set a limit on disk I/O requests sent from the VMs to the host machine. This can prevent a VM from over-utilizing shared resources and impacting the performance of other VMs.

To enable disk I/O throttling, set a limit on disk I/O requests sent from each block device attached to VMs to the host machine.

**Procedure**

1. Use the `virsh domblklist` command to list the names of all the disk devices on a specified VM.

```
# virsh domblklist rollin-coal

Target   Source
--------- ---------------------------------------------
vda       /var/lib/libvirt/images/rollin-coal.qcow2
sda       -
         
```

2. Find the host block device where the virtual disk that you want to throttle is mounted.

For example, if you want to throttle the `sdb` virtual disk from the previous step, the following output shows that the disk is mounted on the `/dev/nvme0n1p3` partition.

```
$ lsblk
NAME                MAJ:MIN  RM  SIZE RO TYPE  MOUNTPOINT
zram0               252:0    0   4G  0  disk [SWAP]
nvme0n1             259:0    0 238.5G 0 disk
├─nvme0n1p1         259:1    0  600M 0 part /boot/efi
└─nvme0n1p2         259:2    0   1G  0 part /boot
└─nvme0n1p3         259:3    0 236.9G 0 part
        └─luks-a1123911-6f37-463c-b4eb-fxzy1ac12lea 253:0    0 236.9G 0 crypt /home
```

3. Set I/O limits for the block device using the `virsh blkiotune` command.
# virsh blkiotune VM-name --parameter device,limit

The following example throttles the sdb disk on the rollin-coal VM to 1000 read and write I/O operations per second and to 50 MB per second read and write throughput.

# virsh blkiotune rollin-coal --device-read-iops-sec /dev/nvme0n1p3,1000 --device-write-iops-sec /dev/nvme0n1p3,1000 --device-write-bytes-sec /dev/nvme0n1p3,52428800 --device-read-bytes-sec /dev/nvme0n1p3,52428800

Additional information

- Disk I/O throttling can be useful in various situations, for example when VMs belonging to different customers are running on the same host, or when quality of service guarantees are given for different VMs. Disk I/O throttling can also be used to simulate slower disks.
- I/O throttling can be applied independently to each block device attached to a VM and supports limits on throughput and I/O operations.
- Red Hat does not support using the virsh blkdeviotune command to configure I/O throttling in VMs. For more information on unsupported features when using RHEL 9 as a VM host, see Section 14.3, “Unsupported features in RHEL 9 virtualization”.

12.4.3. Enabling multi-queue virtio-scsi

When using virtio-scsi storage devices in your virtual machines (VMs), the multi-queue virtio-scsi feature provides improved storage performance and scalability. It enables each virtual CPU (vCPU) to have a separate queue and interrupt to use without affecting other vCPUs.

Procedure

- To enable multi-queue virtio-scsi support for a specific VM, add the following to the VM’s XML configuration, where \( N \) is the total number of vCPU queues:

  ```xml
  <controller type='scsi' index='0' model='virtio-scsi'>
    <driver queues='N'/>
  </controller>
  ```

12.5. 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. Adjust how many host CPUs are assigned to the VM. You can do this using the CLI or the web console.

2. Ensure that the vCPU model is aligned with the CPU model of the host. For example, to set the testguest1 VM to use the CPU model of the host:

   ```bash
   # virt-xml testguest1 --edit --cpu host-model
   ```

3. Deactivate kernel same-page merging (KSM).
4. If your host machine uses Non-Uniform Memory Access (NUMA), you can also configure NUMA for its VMs. This maps the host’s CPU and memory processes onto the CPU and memory processes of the VM as closely as possible. In effect, NUMA tuning provides the vCPU with a more streamlined access to the system memory allocated to the VM, which can improve the vCPU processing effectiveness. For details, see Configuring NUMA in a virtual machine and Sample vCPU performance tuning scenario.

12.5.1. Adding and removing virtual CPUs using the command-line interface

To increase or optimize the CPU performance of a virtual machine (VM), you can add or remove virtual CPUs (vCPUs) assigned to the VM.

When performed on a running VM, this is also referred to as vCPU hot plugging and hot unplugging. However, note that vCPU hot unplug is not supported in RHEL 9, and Red Hat highly discourages its use.

Prerequisites

- Optional: View the current state of the vCPUs in the targeted VM. For example, to display the number of vCPUs on the testguest VM:

  ```
  # virsh vcpucount testguest
  maximum config 4
  maximum live 2
  current config 2
  current live 1
  ```

  This output indicates that testguest is currently using 1 vCPU, and 1 more vCPU can be hot plugged to it to increase the VM’s performance. However, after reboot, the number of vCPUs testguest uses will change to 2, and it will be possible to hot plug 2 more vCPUs.

Procedure

1. Adjust the maximum number of vCPUs that can be attached to a VM, which takes effect on the VM’s next boot.
   For example, to increase the maximum vCPU count for the testguest VM to 8:

   ```
   # virsh setvcpus testguest 8 --maximum --config
   ```

   Note that the maximum may be limited by the CPU topology, host hardware, the hypervisor, and other factors.

2. Adjust the current number of vCPUs attached to a VM, up to the maximum configured in the previous step. For example:

   - To increase the number of vCPUs attached to the running testguest VM to 4:

     ```
     # virsh setvcpus testguest 4 --live
     ```

     This increases the VM’s performance and host load footprint of testguest until the VM’s next boot.

   - To permanently decrease the number of vCPUs attached to the testguest VM to 1:

     ```
     # virsh setvcpus testguest 1 --config
     ```
This decreases the VM’s performance and host load footprint of testguest after the VM’s next boot. However, if needed, additional vCPUs can be hot plugged to the VM to temporarily increase its performance.

Verification

- Confirm that the current state of vCPU for the VM reflects your changes.

```
# virsh vcpucount testguest
maximum config 8
maximum live 4
current config 1
current live 4
```

Additional resources

- For information on adding and removing vCPUs using the web console, see Section 12.5.2, "Managing virtual CPUs using the web console".

12.5.2. Managing virtual CPUs using the web console

Using the RHEL 9 web console, you can review and configure virtual CPUs used by virtual machines (VMs) to which the web console is connected.

Prerequisites

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

Procedure

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

2. Click edit next to the number of vCPUs in the Overview pane. The vCPU details dialog appears.

1. Configure the virtual CPUs for the selected VM.

   - vCPU Count - The number of vCPUs currently in use.
NOTE
The vCPU count cannot be greater than the vCPU Maximum.

- **vCPU Maximum** - The maximum number of virtual CPUs that can be configured for the VM. If this value is higher than the **vCPU Count**, additional vCPUs can be attached to the VM.

- **Sockets** - The number of sockets to expose to the VM.

- **Cores per socket** - The number of cores for each socket to expose to the VM.

- **Threads per core** - The number of threads for each core to expose to the VM.

   Note that the Sockets, Cores per socket, and Threads per core options adjust the CPU topology of the VM. This may be beneficial for vCPU performance and may impact the functionality of certain software in the guest OS. If a different setting is not required by your deployment, keep the default values.

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

NOTE
Changes to virtual CPU settings only take effect after the VM is restarted.

Additional resources:

- For information on managing your vCPUs using the command-line interface, see Section 12.5.1, “Adding and removing virtual CPUs using the command-line interface”.

### 12.5.3. Configuring NUMA in a virtual machine

The following methods can be used to configure Non-Uniform Memory Access (NUMA) settings of a virtual machine (VM) on a RHEL 9 host.

**Prerequisites**

- The host is a NUMA-compatible machine. To detect whether this is the case, use the `virsh nodeinfo` command and see the **NUMA cell(s)** line:

```
# virsh nodeinfo
CPU model:       x86_64
CPU(s):          48
CPU frequency:   1200 MHz
CPU socket(s):   1
Core(s) per socket: 12
Thread(s) per core: 2
NUMA cell(s):    2
Memory size:     67012964 KiB
```

If the value of the line is 2 or greater, the host is NUMA-compatible.

**Procedure**
For ease of use, you can set up a VM’s NUMA configuration using automated utilities and services. However, manual NUMA setup is more likely to yield a significant performance improvement.

**Automatic methods**

- Set the VM’s NUMA policy to **Preferred**. For example, to do so for the `testguest5` VM:
  
  ```
  # virt-xml testguest5 --edit --vcpus placement=auto
  # virt-xml testguest5 --edit --numatune mode=preferred
  ```

- Enable automatic NUMA balancing on the host:
  
  ```
  # echo 1 > /proc/sys/kernel/numa_balancing
  ```

- Use the **numad** command to automatically align the VM CPU with memory resources.
  
  ```
  # numad
  ```

**Manual methods**

1. Pin specific vCPU threads to a specific host CPU or range of CPUs. This is also possible on non-NUMA hosts and VMs, and is recommended as a safe method of vCPU performance improvement.

   For example, the following commands pin vCPU threads 0 to 5 of the `testguest6` VM to host CPUs 1, 3, 5, 7, 9, and 11, respectively:
   
   ```
   # virsh vcpupin testguest6 0 1
   # virsh vcpupin testguest6 1 3
   # virsh vcpupin testguest6 2 5
   # virsh vcpupin testguest6 3 7
   # virsh vcpupin testguest6 4 9
   # virsh vcpupin testguest6 5 11
   ```

   Afterwards, you can verify whether this was successful:
   
   ```
   # 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:
   
   ```
   # 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:

```
# 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 12.5.4, “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 12.7, “Virtual machine performance monitoring tools”.

- NUMA tuning is currently not possible to perform on IBM Z hosts. For further information, see How virtualization on IBM Z differs from AMD64 and Intel 64

### 12.5.4. Sample vCPU performance tuning scenario

To obtain the best vCPU performance possible, Red Hat recommends using manual `vcpupin`, `emulatorpin`, and `numatune` settings together, for example like in the following scenario.

**Starting scenario**

- Your host has the following hardware specifics:
  - 2 NUMA nodes
  - 3 CPU cores on each node
  - 2 threads on each core

  The output of `virsh nodeinfo` of such a machine would look similar to:

```
# 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:

```xml
<topology>
  <cells num="2">
    <cell id="0">
      <memory unit="KiB">15624346</memory>
      <pages unit="KiB" size="4">3906086</pages>
      <pages unit="KiB" size="2048">0</pages>
      <pages unit="KiB" size="1048576">0</pages>
      <distances>
        <sibling id="0" value="10" />
        <sibling id="1" value="21" />
      </distances>
      <cpus num="6">
        <cpu id="0" socket_id="0" core_id="0" siblings="0,3" />
        <cpu id="1" socket_id="0" core_id="1" siblings="1,4" />
        <cpu id="2" socket_id="0" core_id="2" siblings="2,5" />
        <cpu id="3" socket_id="0" core_id="0" siblings="0,3" />
        <cpu id="4" socket_id="0" core_id="1" siblings="1,4" />
        <cpu id="5" socket_id="0" core_id="2" siblings="2,5" />
      </cpus>
    </cell>
    <cell id="1">
      <memory unit="KiB">15624346</memory>
      <pages unit="KiB" size="4">3906086</pages>
      <pages unit="KiB" size="2048">0</pages>
      <pages unit="KiB" size="1048576">0</pages>
      <distances>
        <sibling id="0" value="21" />
        <sibling id="1" value="10" />
      </distances>
      <cpus num="6">
        <cpu id="6" socket_id="1" core_id="3" siblings="6,9" />
        <cpu id="7" socket_id="1" core_id="4" siblings="7,10" />
        <cpu id="8" socket_id="1" core_id="5" siblings="8,11" />
        <cpu id="9" socket_id="1" core_id="3" siblings="6,9" />
        <cpu id="10" socket_id="1" core_id="4" siblings="7,10" />
        <cpu id="11" socket_id="1" core_id="5" siblings="8,11" />
      </cpus>
    </cell>
  </cells>
</topology>
```

2. **Optional**: Test the performance of the VM using the applicable tools and utilities.

3. Set up and mount 1 GiB huge pages on the host:

   a. Add the following line to the host’s kernel command line:

   ```
   default_hugepagesz=1G hugepagesz=1G
   ```
b. Create the `/etc/systemd/system/hugetlb-gigantic-pages.service` file with the following content:

```
[Unit]
Description=HugeTLB Gigantic Pages Reservation
DefaultDependencies=no
Before=dev-hugepages.mount
ConditionPathExists=/sys/devices/system/node
ConditionKernelCommandLine=hugepagesz=1G

[Service]
Type=oneshot
RemainAfterExit=yes
ExecStart=/etc/systemd/hugetlb-reserve-pages.sh

[Install]
WantedBy=sysinit.target
```

c. Create the `/etc/systemd/hugetlb-reserve-pages.sh` file with the following content:

```
#!/bin/sh

nodes_path=/sys/devices/system/node/
if [ ! -d $nodes_path ]; then
    echo "ERROR: $nodes_path does not exist"
    exit 1
fi

reserve_pages() {
    echo $1 > $nodes_path/$2/hugepages/hugepages-1048576kB/nr_hugepages
}
reserve_pages 4 node1
reserve_pages 4 node2
```

This reserves four 1GiB huge pages from `node1` and four 1GiB huge pages from `node2`.

d. Make the script created in the previous step executable:

```
# chmod +x /etc/systemd/hugetlb-reserve-pages.sh
```

e. Enable huge page reservation on boot:

```
# systemctl enable hugetlb-gigantic-pages
```

4. Use the `virsh edit` command to edit the XML configuration of the VM you wish to optimize, in this example `super-VM`:

```
# virsh edit super-vm
```

5. Adjust the XML configuration of the VM in the following way:

a. Set the VM to use 8 static vCPUs. Use the `<vcpu/>` element to do this.
b. Pin each of the vCPU threads to the corresponding host CPU threads that it mirrors in the topology. To do so, use the `<vcpupin/>` elements in the `<cputune/>` section. Note that, as shown by the `virsh capabilities` utility above, host CPU threads are not ordered sequentially in their respective cores. In addition, the vCPU threads should be pinned to the highest available set of host cores on the same NUMA node. For a table illustration, see the Additional Resources section below.

The XML configuration for steps a. and b. can look similar to:

```xml
<cpu>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='4'/>
  <vcpupin vcpu='2' cpuset='2'/>
  <vcpupin vcpu='3' cpuset='5'/>
  <vcpupin vcpu='4' cpuset='7'/>
  <vcpupin vcpu='5' cpuset='10'/>
  <vcpupin vcpu='6' cpuset='8'/>
  <vcpupin vcpu='7' cpuset='11'/>
  <emulatorpin cpuset='6,9'/>
</cpu>
```

c. Set the VM to use 1 GiB huge pages:

```xml
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB'/>
  </hugepages>
</memoryBacking>
```

d. Configure the VM’s NUMA nodes to use memory from the corresponding NUMA nodes on the host. To do so, use the `<memnode/>` elements in the `<numatune/>` section:

```xml
<numatune>
  <memory mode="preferred" nodeset="1"/>
  <memnode cellid="0" mode="strict" nodeset="0"/>
  <memnode cellid="1" mode="strict" nodeset="1"/>
</numatune>
```

e. Ensure the CPU mode is set to `host-passthrough`, and that the CPU uses cache in `passthrough` mode:

```xml
<cpu mode="host-passthrough">
  <topology sockets="2" cores="2" threads="2"/>
  <cache mode="passthrough"/>
</cpu>
```

**Verification**

1. Confirm that the resulting XML configuration of the VM includes a section similar to the following:

```xml
[...]
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB'/>
```
Optional: 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>
<thead>
<tr>
<th>Table 12.1. Host topology</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU threads</strong></td>
</tr>
<tr>
<td><strong>Cores</strong></td>
</tr>
<tr>
<td><strong>Sockets</strong></td>
</tr>
</tbody>
</table>
In this scenario, there are 2 NUMA nodes and 8 vCPUs. Therefore, 4 vCPU threads should be pinned to each node.

In addition, Red Hat recommends leaving at least a single CPU thread available on each node for host system operations.

Because in this example, each NUMA node houses 3 cores, each with 2 host CPU threads, the set for node 0 translates as follows:

```xml
<vcpupin vcpu='0' cpuset='1'/>
<vcpupin vcpu='1' cpuset='4'/>
<vcpupin vcpu='2' cpuset='2'/>
<vcpupin vcpu='3' cpuset='5'/>
```

### 12.5.5. Deactivating kernel same-page merging

Although kernel same-page merging (KSM) improves memory density, it increases CPU utilization, and might adversely affect overall performance depending on the workload. In such cases, you can improve the virtual machine (VM) performance by deactivating KSM.

Depending on your requirements, you can either deactivate KSM for a single session or persistently.

**Procedure**
To deactivate KSM for a single session, use the `systemctl` utility to stop `ksm` and `ksmtuned` services.

```
# systemctl stop ksm
# systemctl stop ksmtuned
```

To deactivate KSM persistently, use the `systemctl` utility to disable `ksm` and `ksmtuned` services.

```
# systemctl disable ksm
Removed /etc/systemd/system/multi-user.target.wants/ksm.service.
# systemctl disable ksmtuned
Removed /etc/systemd/system/multi-user.target.wants/ksmtuned.service.
```

**NOTE**

Memory pages shared between VMs before deactivating KSM will remain shared. To stop sharing, delete all the PageKSM pages in the system using the following command:

```
# echo 2 > /sys/kernel/mm/ksm/run
```

After anonymous pages replace the KSM pages, the `khugepaged` kernel service will rebuild transparent hugepages on the VM’s physical memory.

### 12.6. OPTIMIZING VIRTUAL MACHINE NETWORK PERFORMANCE

Due to the virtual nature of a VM’s network interface card (NIC), the VM loses a portion of its allocated host network bandwidth, which can reduce the overall workload efficiency of the VM. The following tips can minimize the negative impact of virtualization on the virtual NIC (vNIC) throughput.

**Procedure**

Use any of the following methods and observe if it has a beneficial effect on your VM network performance:

**Enable the `vhost_net` module**

On the host, ensure the `vhost_net` kernel feature is enabled:

```
# 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:

```
# modprobe vhost_net
```

**Set up multi-queue virtio-net**

To set up the `multi-queue virtio-net` feature for a VM, use the `virsh edit` command to edit 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:
If the VM is running, restart it for the changes to take effect.

**Batching network packets**
In Linux VM configurations with a long transmission path, batching packets before submitting them to the kernel may improve cache utilization. To set up packet batching, use the following command on the host, and replace `tap0` with the name of the network interface that the VMs use:

```bash
# ethtool -C tap0 rx-frames 64
```

**SR-IOV**
If your host NIC supports SR-IOV, use SR-IOV device assignment for your vNICs. For more information, see Managing SR-IOV devices.

**Additional resources**
- For additional information on virtual network connection types and tips for usage, see Section 11.1, “Understanding virtual networking”.

### 12.7. VIRTUAL MACHINE PERFORMANCE MONITORING TOOLS
To identify what consumes the most VM resources and which aspect of VM performance needs optimization, performance diagnostic tools, both general and VM-specific, can be used.

**Default OS performance monitoring tools**
For standard performance evaluation, you can use the utilities provided by default by your host and guest operating systems:

- On your RHEL 9 host, as root, use the `top` utility or the `system monitor` application, and look for `qemu` and `virt` in the output. This shows how much host system resources your VMs are consuming.
  - If the monitoring tool displays that any of the `qemu` or `virt` processes consume a large portion of the host CPU or memory capacity, use the `perf` utility to investigate. For details, see below.
  - In addition, if a `vhost_net` thread process, named for example `vhost_net-1234`, is displayed as consuming an excessive amount of host CPU capacity, consider using virtual network optimization features, such as `multi-queue virtio-net`.

- On the guest operating system, use performance utilities and applications available on the system to evaluate which processes consume the most system resources.
  - On Linux systems, you can use the `top` utility.
  - On Windows systems, you can use the `Task Manager` application.

```bash
perf kvm
```
You can use the `perf` utility to collect and analyze virtualization-specific statistics about the performance of your RHEL 9 host. To do so:

1. On the host, install the `perf` package:

   ```
   # dnf install perf
   ```

2. Use one of the `perf kvm stat` commands to display perf statistics for your virtualization host:
   - For real-time monitoring of your hypervisor, use the `perf kvm stat live` command.
   - To log the perf data of your hypervisor over a period of time, activate the logging using the `perf kvm stat record` command. After the command is canceled or interrupted, the data is saved in the `perf.data.guest` file, which can be analyzed using the `perf kvm stat report` command.

3. Analyze the `perf` output for types of VM-EXIT events and their distribution. For example, the `PAUSE_INSTRUCTION` events should be infrequent, but in the following output, the high occurrence of this event suggests that the host CPUs are not handling the running vCPUs well. In such a scenario, consider shutting down some of your active VMs, removing vCPUs from these VMs, or tuning the performance of the vCPUs.

   ```
   # perf kvm stat report
   Analyze events for all VMs, all VCPUs:

<table>
<thead>
<tr>
<th>VM-EXIT</th>
<th>Samples</th>
<th>Samples%</th>
<th>Time%</th>
<th>Min Time</th>
<th>Max Time</th>
<th>Avg time</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXTERNAL_INTERRUPT</td>
<td>365634</td>
<td>31.59%</td>
<td>18.04%</td>
<td>0.42us</td>
<td>58780.59us</td>
<td>204.08us ( +- 0.99% )</td>
</tr>
<tr>
<td>MSR_WRITE</td>
<td>293428</td>
<td>25.35%</td>
<td>0.13%</td>
<td>0.59us</td>
<td>17873.02us</td>
<td>1.80us ( +- 4.63% )</td>
</tr>
<tr>
<td>PREEMPTION_TIMER</td>
<td>276162</td>
<td>23.86%</td>
<td>0.23%</td>
<td>0.51us</td>
<td>21396.03us</td>
<td>3.38us ( +- 5.19% )</td>
</tr>
<tr>
<td>PAUSE_INSTRUCTION</td>
<td>189375</td>
<td>16.36%</td>
<td>11.75%</td>
<td>0.72us</td>
<td>29655.25us</td>
<td>256.77us ( +- 0.70% )</td>
</tr>
<tr>
<td>HLT</td>
<td>20440</td>
<td>1.77%</td>
<td>69.83%</td>
<td>0.62us</td>
<td>79319.41us</td>
<td>14134.56us ( +- 0.79% )</td>
</tr>
<tr>
<td>VMCALL</td>
<td>12426</td>
<td>1.07%</td>
<td>0.03%</td>
<td>1.02us</td>
<td>5416.25us</td>
<td>8.77us ( +- 7.36% )</td>
</tr>
<tr>
<td>EXCEPTION_NMI</td>
<td>27</td>
<td>0.00%</td>
<td>0.00%</td>
<td>0.69us</td>
<td>1.34us</td>
<td>0.98us ( +- 3.50% )</td>
</tr>
<tr>
<td>EPT_MISCONFIG</td>
<td>5</td>
<td>0.00%</td>
<td>0.00%</td>
<td>5.15us</td>
<td>10.85us</td>
<td>7.88us ( +- 11.67% )</td>
</tr>
</tbody>
</table>

   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:

```
# numastat -c qemu-kvm

Per-node process memory usage (in MBs)
PID   Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------- ------ ------ ------ ------ ------ ------ ------ ------ -----
51722 (qemu-kvm)  68    16   357   6936     2     3   147   598   8128
51747 (qemu-kvm) 245   11     5    18   5172   2532     1   92   8076
53736 (qemu-kvm)  62  432  1661   506    136    22   445  8116
53773 (qemu-kvm) 1393   3     7   12     0     0 6702   8114
---------- ------ ------ ------ ------ ------ ------ ------ ------ -----
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.

```
# numastat -c qemu-kvm

Per-node process memory usage (in MBs)
PID   Node 0 Node 1 Node 2 Node 3 Node 4 Node 5 Node 6 Node 7 Total
---------- ------ ------ ------ ------ ------ ------ ------ ------ -----
51747 (qemu-kvm)   0    0    7    0  8072     0    1    0   8080
53736 (qemu-kvm)   0    0    7    0    0   8113    0   8120
53773 (qemu-kvm)   0    0    7    0    0    1 8110  8118
59065 (qemu-kvm)   0    0  8050    0    0    0    0   8051
---------- ------ ------ ------ ------ ------ ------ ------ ------ -----
Total       0    0  8072   0  8072    0   8114 8110 32368
```

### 12.8. ADDITIONAL RESOURCES

- When using Windows as the guest operating system of your VM, Red Hat recommends applying additional optimization measures. For details, see Optimizing Windows virtual machines.
CHAPTER 13. SHARING FILES BETWEEN THE HOST AND ITS VIRTUAL MACHINES

You may frequently require to share data between your host system and the virtual machines (VMs) it runs. To do so quickly and efficiently, you can set up NFS or Samba file shares on your system. As a newly supported feature in RHEL 9, you can also use the `virtiofs` file system to share data with your Linux VMs.

13.1. SHARING FILES BETWEEN THE HOST AND ITS VIRTUAL MACHINES USING VIRTIOFS

When using RHEL 9 as your hypervisor, you can efficiently share files between your host system and its virtual machines (VM) using the `virtiofs` feature.

Prerequisites

- Virtualization is installed and enabled on your RHEL 9 host.
- A directory that you want to share with your VMs. If you do not want to share any of your existing directories, create a new one, for example named `shared-files`.

```
# mkdir shared-files
```

- The VM you want to share data with is using a Linux distribution as its guest OS.

Procedure

1. For each directory on the host that you want to share with your VM, set it as a virtiofs file system in the VM’s XML configuration.
   a. Open the XML configuration of the intended VM.

   ```
   # virsh edit vm-name
   ```

   b. Add an entry similar to the following to the `<devices>` section of the VM’s XML configuration.

   ```xml
   <filesystem type='mount' accessmode='passthrough'>
     <driver type='virtiofs'/>
     <binary path='/usr/libexec/virtiofsd' xattr='on'/>
     <source dir='/root/shared-files'/>
     <target dir='host-file-share'/>
   </filesystem>
   ```

   This example sets the `/root/shared-files` directory on the host to be visible as `host-file-share` to the VM.

2. Add a NUMA topology for shared memory to the XML configuration. The following example adds a basic topology for all CPUs and all RAM.

   ```xml
   <cpu mode='host-passthrough' check='none'>
   <numa>
     <cell id='0' cpus='0-{number-vcpus - 1}' memory='{ram-amount-KiB}' unit='KiB'>
   ```
3. Add shared memory backing to the `<domain>` section of the XML configuration:

```xml
<domain>
    [...]  
    <memoryBacking>
        <access mode="shared"/>
    </memoryBacking>
    [...]  
</domain>
```

4. Boot up the VM.

```
# virsh start vm-name
```

5. Mount the file system in the guest operating system (OS). The following example mounts the previously configured `host-file-share` directory with a Linux guest OS.

```
# mount -t virtiofs host-file-share /mnt
```

**Verification**

- Ensure that the shared directory became accessible on the VM and that you can now open files stored in the directory.

**Known issues and limitations**

- File-system mount options related to access time, such as `noatime` and `strictatime`, are not likely to work with virtiofs, and Red Hat discourages their use.

**Additional resources**

- If `virtiofs` is not optimal for your usecase or supported for your system, you can use **NFS** or **Samba** instead.

### 13.2. 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 is installed on the host.
- A directory that you want to share with your VMs. If you do not want to share any of your existing directories, create a new one, for example named `shared-files`.

```
# mkdir shared-files
```
The host is visible and reachable over a network for the VM. This is generally the case if the VM is connected using the NAT and bridge type of virtual networks. However, for the macvtap connection, you must first set up the macvlan feature on the host. To do so:

1. Create a network device file in the host’s `/etc/systemd/network/` directory, for example called `vm-macvlan.netdev`.
   
   ```bash
   # vim /etc/systemd/network/vm-macvlan.netdev
   ```

2. Edit the network device file to have the following content. You can replace `vm-macvlan` with the name you chose for your network device.

   ```
   [NetDev]
   Name=vm-macvlan
   Kind=macvlan
   
   [MACVLAN]
   Mode=bridge
   ```

3. Create a network configuration file for your macvlan network device, for example `vm-macvlan.network`.
   
   ```bash
   # vim /etc/systemd/network/vm-macvlan.network
   ```

4. Edit the network configuration file to have the following content. You can replace `vm-macvlan` with the name you chose for your network device.

   ```
   [Match]
   Name=_vm-macvlan_
   
   [Network]
   IPForward=yes
   Address=192.168.250.33/24
   Gateway=192.168.250.1
   DNS=192.168.250.1
   ```

5. Create a network configuration file for your physical network interface. For example, if your interface is `enp4s0`:
   
   ```bash
   # vim /etc/systemd/network/enp4s0.network
   ```

   If you are unsure what interface name to use, you can use the `ifconfig` command on your host to obtain the list of active network interfaces.

6. Edit the physical network configuration file to make the physical network a part of the macvlan interface, in this case `vm-macvlan`:

   ```
   [Match]
   Name=enp4s0
   
   [Network]
   MACVLAN=vm-macvlan
   ```

7. Reboot your host.
• Optional: For improved security, ensure your VMs are compatible with NFS version 4 or later.

Procedure

1. On the host, export a directory with the files you want to share as a network file system (NFS).
   a. Obtain the IP address of each virtual machine you want to share files with. The following example obtains the IPs of testguest1 and testguest2.

   ```bash
   # virsh domifaddr testguest1
   Name       MAC address          Protocol     Address
   ---------------------------------------------------------------
   vnet0      52:53:00:84:57:90    ipv4          192.168.124.220/24
   # virsh domifaddr testguest2
   Name       MAC address          Protocol     Address
   ---------------------------------------------------------------
   vnet1      52:53:00:65:29:21    ipv4          192.168.124.17/24
   ```

   b. Edit the `/etc/exports` file on the host and add a line that includes the directory you want to share, IPs of VMs you want to share with, and sharing options.

   ```bash
   Shared directory  VM1-IP(options)  VM2-IP(options)  [...] 
   ```

   c. Export the updated file system.

   ```bash
   # exportfs -a
   ```

   d. Ensure the NFS process is started:

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

e. Obtain the IP address of the host system. This will be used for mounting the shared directory on the VMs later.

   ```bash
   # ip addr
   [...] 5: virbr0: [BROADCAST,MULTICAST,UP,LOWER_UP] mtu 1500 qdisc noqueue state
   UP group default qlen 1000
   link/ether 52:54:00:32:ff:a5 brd ff:ff:ff:ff:ff:ff
   inet 192.168.124.1/24 brd 192.168.124.255 scope global virbr0
   valid_lft forever preferred_lft forever
   [...]
   ```

   Note that the relevant network is the one being used for connection to the host by the VMs you want to share files with. Usually, this is `virbr0`.

2. On the guest OS of a VM specified in the `/etc/exports` file, mount the exported file system.
a. Create a directory you want to use as a mount point for the shared file system, for example /mnt/host-share:

```bash
# 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:

```bash
# mount 192.168.124.1:/usr/local/shared-files /mnt/host-share
```

**Verification**

- To verify the mount has succeeded, access and explore the shared directory on the mount point:

```bash
# cd /mnt/host-share
# ls
shared-file1  shared-file2  shared-file3
```

### 13.3. 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:

  ```bash
  # dnf install samba
  ```

- The host is visible and reachable over a network for the VM. This is generally the case if the VM is connected using the **NAT** and **bridge** type of virtual networks. However, for the **macvtap** connection, you must first set up the **macvlan** feature on the host. To do so:

  1. Create a network device file, for example called `vm-macvlan.netdev` in the host’s /etc/systemd/network/ directory.

     ```bash
     # vim /etc/systemd/network/vm-macvlan.netdev
     ```

  2. Edit the network device file to have the following content. You can replace `vm-macvlan` with the name you chose for your network device.

     ```bash
     [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`.

   ```
   # vim /etc/systemd/network/vm-macvlan.network
   ```

4. Edit the network configuration file to have the following content. You can replace `vm-macvlan` with the name you chose for your network device.

   ```
   [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`:

   ```
   # vim /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 to make the physical network 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.

1. Create the `/samba/VM-share` directory.
   ```
   # mkdir -p /samba/VM-share
   ```
2. 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.
   ```
3. Restart the Samba service.
   ```
   # systemctl restart smb.service
   ```
4. 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/
   ```
5. Add the SELinux Samba sharing label to `/etc/samba/VM-share/`
   ```
   # chcon -t samba_share_t /samba/VM-share/
   ```

2. On the Windows guest operating system, attach the Samba share as a network location.
   a. Open the File Explorer and right-click "This PC".
   b. In the context menu, click **Add a network location**.
c. In the Add Network Location wizard that opens, select "Choose a custom network location" and click **Next**.

d. In the "Internet or network address" field, type `host-IP/VM-share`, where `host-IP` is the IP address of the host. Usually, the host IP is the default gateway of the VM. Afterwards, click **Next**.

e. When the wizard asks if you want to rename the shared directory, keep the default name. This ensures the consistency of file sharing configuration across the VM and the guest. Click **Next**.

f. If accessing the network location was successful, you can now click **Finish** and open the shared directory.
CHAPTER 14. FEATURE SUPPORT AND LIMITATIONS IN RHEL 9 VIRTUALIZATION

This document provides information on feature support and restrictions in Red Hat Enterprise Linux 9 (RHEL 9) virtualization.

This is a document for a Beta version of RHEL 9. Because Beta versions are unsupported by default, the following information serves only as a preview, and is subject to substantial change.

14.1. HOW RHEL VIRTUALIZATION SUPPORT WORKS

A set of support limitations applies to virtualization in Red Hat Enterprise Linux 9 (RHEL 9). This means that when you use certain features or exceed a certain amount of allocated resources when using virtual machines in RHEL 9, Red Hat will not support these guests unless you have a specific subscription plan.

Features listed in Section 14.2, “Recommended features in RHEL 9 virtualization” have been tested and certified by Red Hat to work with the KVM hypervisor on a RHEL 9 system. Therefore, they are fully supported and recommended for use in virtualization in RHEL 9.

Features listed in Section 14.3, “Unsupported features in RHEL 9 virtualization” may work, but are not supported and not intended for use in RHEL 9. Therefore, Red Hat strongly recommends not using these features in RHEL 9 with KVM.

Section 14.4, “Resource allocation limits in RHEL 9 virtualization” lists the maximum amount of specific resources supported on a KVM guest in RHEL 9. Guests that exceed these limits are not supported by Red Hat.

In addition, unless stated otherwise, all features and solutions used by the documentation for RHEL 9 virtualization are supported. However, some of these have not been completely tested and therefore may not be fully optimized.

IMPORTANT

Many of these limitations do not apply to other virtualization solutions provided by Red Hat, such as Red Hat Virtualization (RHV), OpenShift Virtualization, or Red Hat OpenStack Platform (RHOSP).

14.2. RECOMMENDED FEATURES IN RHEL 9 VIRTUALIZATION

The following features are recommended for use with the KVM hypervisor included with Red Hat Enterprise Linux 9 (RHEL 9):

Host system architectures

RHEL 9 with KVM is only supported on the following host architectures:

- AMD64 and Intel 64
- IBM Z - IBM z13 systems and later

Any other hardware architectures are not supported for using RHEL 9 as a KVM virtualization host, and Red Hat highly discourages doing so. Notably, this includes the 64-bit ARM architecture (ARM 64), which is only provided as Technology Preview.

Guest operating systems
Red Hat supports KVM virtual machines that use the following operating systems (OSs):

- Red Hat Enterprise Linux 7 and later
- Microsoft Windows 10 and later
- Microsoft Windows Server 2016 and later

Note, however, that by default, your guest OS does not use the same subscription as your host. Therefore, you must activate a separate licence or subscription for the guest OS to work properly.

**Machine types**

To ensure that your VM is compatible with your host architecture and that the guest OS runs optimally, the VM must use an appropriate machine type.

When creating a VM using the command line, the `virt-install` utility provides multiple methods of setting the machine type.

- When you use the `--os-variant` option, `virt-install` automatically selects the machine type recommended for your host CPU and supported by the guest OS.
- If you do not use `--os-variant` or require a different machine type, use the `--machine` option to specify the machine type explicitly.
- If you specify a `--machine` value that is unsupported or not compatible with your host, `virt-install` fails and displays an error message.

The recommended machine types for KVM virtual machines on supported architectures, and the corresponding values for the `--machine` option, are as follows. Y stands for the latest minor version of RHEL 9.

- On Intel 64 and AMD64 (x86_64): `pc-q35-rhel9.Y.0 → --machine=q35`
- On IBM Z (s390x): `s390-ccw-virtio-rhel9.Y.0 → --machine=s390-ccw-virtio`

To obtain the machine type of an existing VM:

```
# virsh dumpxml VM-name | grep machine=
```

To view the full list of machine types supported on your host:

```
# /usr/libexec/qemu-kvm -M help
```

**Additional resources**

- For information about unsupported guest OS types and features in RHEL 9 virtualization, see Section 14.3, “Unsupported features in RHEL 9 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 9 virtualization”.

### 14.3. UNSUPPORTED FEATURES IN RHEL 9 VIRTUALIZATION

The following features are not supported by the KVM hypervisor included with Red Hat Enterprise Linux 9 (RHEL 9):
IMPORTANT

Many of these limitations may not apply to other virtualization solutions provided by Red Hat, such as Red Hat Virtualization (RHV), OpenShift Virtualization, or Red Hat OpenStack Platform (RHOSP).

Features supported by RHV 4.2 and later, or RHOSP 13 and later, are described as such in the following paragraphs.

Host system architectures

RHEL 9 with KVM is not supported on any host architectures that are not listed in Section 14.2, “Recommended features in RHEL 9 virtualization”.

Notably, the 64-bit ARM architecture (ARM 64) is provided only as a Technology Preview for KVM virtualization on RHEL 9, and Red Hat therefore discourages its use in production environments.

Guest operating systems

KVM virtual machines (VMs) using the following guest operating systems (OSs) on a RHEL 9 host are not supported:

- Microsoft Windows 8.1 and earlier
- Microsoft Windows Server 2012 and earlier
- macOS
- Solaris for x86 systems
- Any OS released prior to 2009

For a list of guest OSs supported on RHEL hosts, see Certified guest operating systems for Red Hat Enterprise Linux with KVM.

Other solutions:

- For a list of guest OSs supported by other virtualization solutions provided by Red Hat, see Certified Guest Operating Systems in Red Hat OpenStack Platform, Red Hat Virtualization and OpenShift Virtualization.

- For a list of guest OSs supported specifically by RHV, see Supported guest operating systems in RHV.

Creating VMs in containers

Red Hat does not support creating KVM virtual machines in any type of container that includes the elements of the RHEL 9 hypervisor (such as the QEMU emulator or the libvirt package).

Other solutions:

- To create VMs in containers, Red Hat recommends using the OpenShift Virtualization offering.

Undocumented virsh commands and options

Any virsh commands and options that are not explicitly recommended by Red Hat documentation may not work correctly, and Red Hat recommends not using them in your production environment.
The QEMU command line

QEMU is an essential component of the virtualization architecture in RHEL 9, but it is difficult to manage manually, and improper QEMU configurations may cause security vulnerabilities. Therefore, using `qemu-*` command-line utilities, such as `qemu-kvm` is not supported by Red Hat.

Instead, 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.

vCPU hot unplug

Removing a virtual CPU (vCPU) from a running VM, also referred to as a vCPU hot unplug, is not supported in RHEL 9.

Other solutions:

- vCPU hot unplugs are supported in RHV. For details, see Hot plugging VCPUs.

Memory hot unplug

Removing a memory device attached to a running VM, also referred to as a memory hot unplug, is unsupported in RHEL 9.

Other solutions:

- Memory hot unplugs are supported in RHV, but only on guest VMs running RHEL with specific guest configurations. For details, see Hot Unplugging Virtual Memory.

QEMU-side I/O throttling

Using the `virsh blkdeviotune` utility to configure maximum input and output levels for operations on virtual disk, also known as QEMU-side I/O throttling, is not supported in RHEL 9.

To set up I/O throttling in RHEL 9, use `virsh blkio`. This is also known as libvirt-side I/O throttling. For instructions, see Section 12.4.2, "Disk I/O throttling in virtual machines".

Other solutions:

- QEMU-side I/O throttling is supported in RHV. For details, see Storage quality of service.
- QEMU-side I/O throttling is also supported in RHOSP. For details, see Setting Resource Limitation on Disk and the Use Quality-of-Service Specifications section in the RHOSP Storage Guide.
- In addition, OpenShift Virtualization supports QEMU-side I/O throttling as well.

Storage live migration

Migrating a disk image of a running VM between hosts is not supported in RHEL 9.

Other solutions:

- Storage live migration is supported in RHV. For details, see Overview of Live Storage Migration.
- Storage live migration is also supported in RHOSP, but with some limitations. For details, see Migrate a Volume.
- It is also possible live-migrate VM storage when using OpenShift Virtualization. For more information, see Virtual machine live migration.
Live snapshots

Creating or loading a snapshot of a running VM, also referred to as a live snapshot, is not supported in RHEL 9.

In addition, note that non-live VM snapshots are deprecated in RHEL 9. Therefore, creating or loading a snapshot of a shut-down VM is supported, but Red Hat recommends not using it.

Other solutions:

- Live snapshots are supported in RHV. For details, see Live snapshots in Red Hat Virtualization.
- RHOSP also supports live snapshots. For details, see Importing virtual machines into the overcloud.

vHost Data Path Acceleration

On RHEL 9 hosts, it is possible to configure vHost Data Path Acceleration (vDPA) for virtio devices, but Red Hat currently does not support this feature, and strongly discourages its use in production environments.

vhost-user

RHEL 9 does not support the implementation of a user-space vHost interface.

Other solutions:

- vhost-user is supported in RHOSP, but only for virtio-net interfaces. For details, see virtio-net implementation and vhost user ports.
- OpenShift Virtualization supports vhost-user as well.

S3 and S4 system power states

Suspending a VM to the Suspend to RAM (S3) or Suspend to disk (S4) system power states is not supported. Note that these features are disabled by default, and enabling them will make your VM not supportable by Red Hat.

Note that the S3 and S4 states are also currently not supported in any other virtualization solution provided by Red Hat.

S3-PR on a multipathed vDisk

SCSI3 persistent reservation (S3-PR) on a multipathed vDisk is not supported in RHEL 9. As a consequence, Windows Cluster is not supported in RHEL 9.

Other solutions:

- S3-PR on a multipathed vDisk is supported in RHV. Therefore, if you require Windows Cluster support, Red Hat recommends using RHV as your virtualization solution. For details, see Cluster support on RHV guests.

virtio-crypto

The drivers for the virtio-crypto device are available in the RHEL 9.0 kernel, and the device can thus be enabled on a KVM hypervisor under certain circumstances. However, using the virtio-crypto device in RHEL 9 is not supported and its use is therefore highly discouraged.
Note that virtio-crypto devices are also not supported in any other virtualization solution provided by Red Hat.

**Incremental live backup**

Configuring a VM backup that only saves VM changes since the last backup, also known as incremental live backup, is not supported in RHEL 9, and Red Hat highly discourages its use.

Other solutions:

- Incremental live backup is provided as a Technology Preview in RHV 4.4 and later.

**net_failover**

Using the net_failover driver to set up an automated network device failover mechanism is not supported in RHEL 9.

Note that net_failover is also currently not supported in any other virtualization solution provided by Red Hat.

**vTPM**

Attaching virtual Trusted Platform Module (vTPM) devices to VMs hosted on a RHEL 9 system is unsupported.

Note that vTPM is also currently not supported in any other virtualization solution provided by Red Hat.

TBD - this should actually be supported, also as per https://docs.google.com/spreadsheets/d/1Gho3X-enEtMPCnadh-dOJqMcRXohYmtHEa7BkYHdKuY/edit#gid=1786802242 → update docs accordingly!

**Multi-FD migration**

Migrating VMs using multiple file descriptors (FDs), also known as multi-FD migration, is not supported in RHEL 9.

Note that multi-FD migrations are also currently not supported in any other virtualization solution provided by Red Hat.

**NVMe devices**

Attaching Non-volatile Memory express (NVMe) devices to VMs hosted in RHEL 9 is not supported.

Note that attaching NVMe devices to VMs is also currently not supported in any other virtualization solution provided by Red Hat.

**TCG**

QEMU and libvirt include a dynamic translation mode using the QEMU Tiny Code Generator (TCG). This mode does not require hardware virtualization support. However, TCG is not supported by Red Hat.

TCG-based guests can be recognized by examining its XML configuration, for example using the virsh dumpxml command.

- The configuration file of a TCG guest contains the following line:

  ```xml
  <domain type='qemu'>
  ```

- The configuration file of a KVM guest contains the following line:
Additional resources

- For information about supported guest OS types and recommended features in RHEL 9 virtualization, see Section 14.2, “Recommended features in RHEL 9 virtualization”.
- For information about the maximum supported amounts of resources that can be allocated to a VM, see Section 14.4, “Resource allocation limits in RHEL 9 virtualization”.

14.4. RESOURCE ALLOCATION LIMITS IN RHEL 9 VIRTUALIZATION

The following limits apply to virtualized resources that can be allocated to a single KVM virtual machine (VM) on a Red Hat Enterprise Linux 9 (RHEL 9) host.

**IMPORTANT**

Many of these limitations do not apply to other virtualization solutions provided by Red Hat, such as Red Hat Virtualization (RHV), OpenShift Virtualization, or Red Hat OpenStack Platform (RHOSP).

**Maximum vCPUs per VM**

RHEL 9 supports up to 384 vCPUs allocated to a single VM.

**PCI devices per VM**

RHEL 9 supports 32 PCI device slots per VM bus, and 8 PCI functions per device slot. This gives a theoretical maximum of 256 PCI functions per bus when multi-function capabilities are enabled in the VM, and no PCI bridges are used.

Each PCI bridge adds a new bus, potentially enabling another 256 device addresses. However, some buses do not make all 256 device addresses available for the user; for example, the root bus has several built-in devices occupying slots.

**Virtualized IDE devices**

KVM is limited to a maximum of 4 virtualized IDE devices per VM.

14.5. AN OVERVIEW OF VIRTUALIZATION FEATURES SUPPORT IN RHEL 9

The following tables provide comparative information about the support state of selected virtualization features in RHEL 9 across the supported system architectures.

**Table 14.1. Device hot plug and hot unplug**

<table>
<thead>
<tr>
<th></th>
<th>Intel 64 and AMD64</th>
<th>IBM Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU hot plug</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>CPU hot unplug</td>
<td>UNSUPPORTED</td>
<td>UNSUPPORTED</td>
</tr>
<tr>
<td>Feature</td>
<td>Intel 64 and AMD64</td>
<td>IBM Z</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Memory hot plug</td>
<td>Supported</td>
<td>UNSUPPORTED</td>
</tr>
<tr>
<td>Memory hot unplug</td>
<td>UNSUPPORTED</td>
<td>UNSUPPORTED</td>
</tr>
<tr>
<td>PCI hot plug</td>
<td>Supported</td>
<td>Supported[a]</td>
</tr>
<tr>
<td>PCI hot unplug</td>
<td>Supported</td>
<td>Supported[b]</td>
</tr>
</tbody>
</table>

[a] Requires using `virtio-*ccw` devices instead of `virtio-*pci`

[b] Requires using `virtio-*ccw` devices instead of `virtio-*pci`

Table 14.2. Other selected features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Intel 64 and AMD64</th>
<th>IBM Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMA tuning</td>
<td>Supported</td>
<td>UNSUPPORTED</td>
</tr>
<tr>
<td>SR-IOV devices</td>
<td>Supported</td>
<td>UNSUPPORTED</td>
</tr>
<tr>
<td>virt-v2v and p2v</td>
<td>Supported</td>
<td>UNSUPPORTED</td>
</tr>
</tbody>
</table>

Note that some of the unsupported features are supported on other Red Hat products, such as Red Hat Virtualization and Red Hat OpenStack platform. For more information, see Section 14.3, “Unsupported features in RHEL 9 virtualization”.

Additional sources

- For a complete list of unsupported features of virtual machines in RHEL 9, see Section 14.3, "Unsupported features in RHEL 9 virtualization".