Red Hat Enterprise Linux 9

配置和管理虚拟化

设置主机、创建和管理虚拟机并了解虚拟化功能

Last Updated: 2024-06-29

Red Hat Enterprise Linux 9 配置和管理虚拟化

设置主机、创建和管理虚拟机并了解虚拟化功能

法律通告

摘要

要将 Red Hat Enterprise Linux (RHEL)系统用作虚拟化主机，请按照本文档中的说明操作。 提供的信息包括： 虚拟化的功能和使用案例是什么 如何使用命令行工具以及 web 控制台管理您的主机和虚拟机 各种系统构架，如 Intel 64、AMD64 和 IBM Z 上的虚拟化的支持限制是什么

目录

对红帽文档提供反馈

我们感谢您对我们文档的反馈。让我们了解如何改进它。

通过 Jira 提交反馈（需要帐户）

1. 登录到 Jira 网站。
2. 点顶部导航栏中的 **Create**
3. 在 **Summary** 字段中输入描述性标题。
4. 在 **Description** 字段中输入您对改进的建议。包括文档相关部分的链接。
5. 点对话框底部的 **Create**。

第 1 章 介绍 RHEL 中的虚拟化

如果您对虚拟化概念或其在 Linux 中的实施不熟悉，以下小节提供了 RHEL 9 虚拟化的一般概述：红帽提供的基本知识、优势、组件和其他可能的虚拟化解决方案。

1.1. 什么是虚拟化？

RHEL 9 提供了*虚拟化*功能，它允许运行 RHEL 9 的机器*托管*多个虚拟机(VM)，也称为*客户机*（*guest*）。VM 使用主机的物理硬件和计算资源，在主机操作系统中作为用户空间进程运行一个独立的虚拟操作系统（*客户机操作系统*）。

换句话说，虚拟化功能使在一个操作系统中执行其他操作系统成为可能。

VM 可让您安全地测试软件配置和功能，运行旧的软件或优化硬件的工作负载效率。有关优点的更多信息，请参阅 虚拟化的优势。

有关虚拟化的更多信息，请参阅 虚拟化主题页面。

后续步骤

- 要在 Red Hat Enterprise Linux 9 中使用虚拟化，请参阅在 Red Hat Enterprise Linux 9 中启用虚拟化。
- 除了 Red Hat Enterprise Linux 9 虚拟化外，红帽还提供很多特殊的虚拟化解决方案，每个解决方案都有不同的用户重点和功能。如需更多信息，请参阅 Red Hat Virtualization 解决方案。

1.2. 虚拟化的优点

与使用物理机器相比，使用虚拟机（VM）有以下优点：

- **灵活精细的资源分配**
  一个虚拟机在主机机器（通常是物理机器）上运行，主机的硬件也可以分配给客户机操作系统使用。但是，物理资源分配是在软件级别上完成的，因此非常灵活。虚拟机使用的主机内存、CPU 或存储空间的比例是可以配置的，可以指定非常精细的资源请求。

  例如：客户机操作系统的磁盘可以在主机的文件系统中以一个文件代表，且该磁盘的大小限制比物理磁盘的限制要小。

- **软件控制的配置**
  虚拟机的整个配置都作为数据保存在主机上，并由软件控制。因此，虚拟机可轻松创建、删除、克隆、迁移、远程操作或连接到远程存储。

- **与主机分离**
  在虚拟内核中运行的客户机操作系统与主机操作系统分开。这意味着可在虚拟机中安装任何操作系统，即使虚拟机操作系统不稳定或受损，主机也不会受到任何影响。

- **空间及成本效率**
  单个物理机器可以托管大量虚拟机。因此，无需多个物理机器执行同样的任务，因此降低了与物理硬件关联的空间、电源和维护的要求。

- **软件兼容性**
  因为虚拟机可以使用不同于其主机的操作系统，所以通过虚拟化，可以运行最初没有为主机操作系统发布的应用程序。例如，通过使用 RHEL 7 客户机操作系统，您可以在 RHEL 9 主机系统上运行为 RHEL 7 发布的应用程序。

注意

不是所有操作系统都作为 RHEL 9 主机中的客户机操作系统被支持。详情请查看 RHEL 9 虚拟化中的推荐功能。

1.3. 虚拟机组件及其交互

RHEL 9 中的虚拟化由以下主要软件组件组成：

虚拟机监控程序

在 RHEL 9 中创建虚拟机(VM)的基础是 *hypervisor*，它是一个软件层，用于控制硬件并在主机中运行多个操作系统。

虚拟机监控程序包括 **Kernel-based Virtual Machine（KVM）**模块和虚拟化内核驱动。这些组件可确保主机中的 Linux 内核为用户空间软件提供虚拟化资源。

在用户空间级别，**QEMU** 模拟器会模拟一个客户机操作系统可以在上面运行的完整虚拟硬件平台，并管理如何在主机中分配资源并提供给客户机。

此外，**libvirt** 软件套件充当管理和通信层，使 与 QEMU 更容易交互、实施安全规则，并提供用于配置和运行 VM 的许多其他工具。

XML 配置

基于主机的 XML 配置文件（也称*域 XML* 文件）决定了特定虚拟机中的所有设置和设备。配置包括：

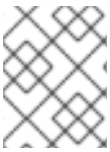
- 元数据，如虚拟机名称、时区和其他有关虚拟机的信息。
- 对虚拟机中的设备的描述，包括虚拟 CPU（vCPUS）、存储设备、输入/输出设备、网络接口卡及其他真实和虚拟硬件。
- 虚拟机设置，如它可以使用的最大内存量、重启设置和其他有关虚拟机行为的设置。

有关 XML 配置内容的更多信息，请参阅 虚拟机 XML 配置示例。

组件交互

当虚拟机启动时，虚拟机监控程序使用 XML 配置在主机上以用户空间进程的形式创建虚拟机实例。hypervisor 还使虚拟机进程能被基于主机的接口访问，如 **virsh**、**virt-install** 和 **guestfish** 工具，或者 Web 控制台 GUI。

当使用这些虚拟化工具时，libvirt 会将它们的输入转换成 QEMU 的指令。QEMU 将指令信息发送到 KVM，这样可确保内核正确分配执行该指令所需的资源。因此，QEMU 可以执行相应的用户空间更改，如创建或修改虚拟机或在虚拟机的客户机操作系统中执行操作。

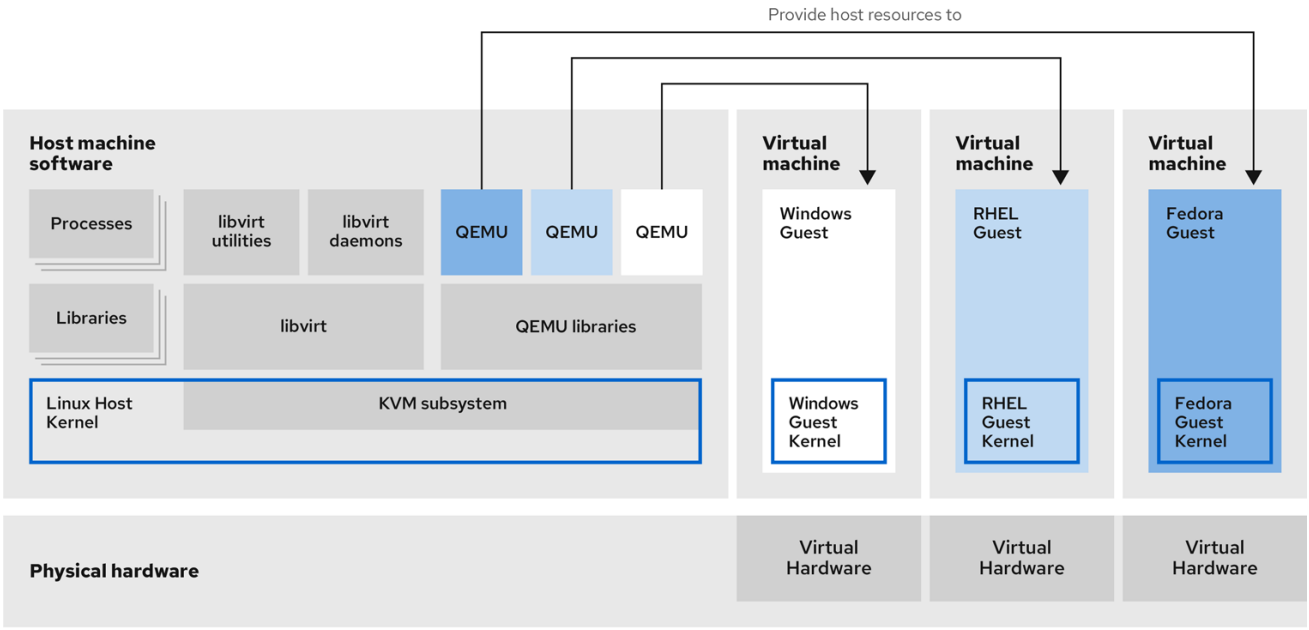


注意

尽管 QEMU 是架构的一个基本组件，但由于安全原因，它并适合于直接在 RHEL 9 系统中使用。因此，红帽不支持 **qemu-*** 命令，强烈建议您使用 libvirt 与 QEMU 进行交互。

有关基于主机的接口的更多信息，请参阅 虚拟化管理的工具和接口。

图 1.1. RHEL 9 虚拟化架构



1.4. 用于虚拟化管理的工具和界面

您可以在 RHEL 9 中使用命令行界面(CLI)或几个图形用户界面(GUI)管理虚拟化。

命令行界面

CLI 是 RHEL 9 中管理虚拟化的最强大的方法。虚拟机（VM）管理的 CLI 命令包括：

- **virsh** - 一个多用途的虚拟化命令行工具程序和 shell，根据提供的参数，可以实现不同功能。例如：
  - 启动和关闭虚拟机 - **virsh start** 和 **virsh shutdown**
  - 列出可用的虚拟机 - **virsh list**
  - 从配置文件创建虚拟机 - **virsh create**
  - 进入虚拟化 shell - **virsh**

  如需更多信息，请参阅 **virsh(1)** 手册页。
- **virt-install** - 用于创建新虚拟机的 CLI 工具。如需更多信息，请参阅 **virt-install(1)** 手册页。
- **virt-xml** - 用于编辑虚拟机配置的工具。
- **guestfish** - 用于检查和修改虚拟机磁盘镜像的工具。如需更多信息，请参阅 **guestfish(1)** 手册页。

图形界面

您可以使用以下 GUI 在 RHEL 9 中管理虚拟化：

- **RHEL 9 web 控制台**（也称为 *Cockpit*）提供了一个可以远程访问的、易于使用的图形用户界面，用于管理虚拟机和虚拟化主机。
  有关使用 web 控制台进行基本虚拟化管理的步骤，请参阅 web 控制台中管理虚拟机。

1.5. 红帽虚拟化解决方案

以下红帽产品基于 RHEL 9 虚拟化功能构建，并扩展了 RHEL 9 中的 KVM 虚拟化功能。另外，许多 RHEL 9 虚拟化的限制 不适用于这些产品：

OpenShift Virtualization

OpenShift Virtualization 基于 KubeVirt 技术，作为 Red Hat OpenShift Container Platform 的一部分，并可在容器中运行虚拟机。
有关 OpenShift Virtualization 的更多信息，请参阅红帽混合云页面。

Red Hat OpenStack Platform（RHOSP）

Red Hat OpenStack Platform 为创建、部署并扩展一个安全可靠的公共或私有 OpenStack 云环境提供了一个集成的基础。
如需有关 Red Hat OpenStack Platform 的更多信息，请参阅红帽客户门户网站或 Red Hat OpenStack Platform 文档套件。

注意

有关 RHEL 中不支持但在其他 Red Hat 虚拟化解决方案中支持的虚拟化功能的详情，请参考 RHEL 9 虚拟化中不支持的功能

第 2 章 启用虚拟化

要在 RHEL 9 中使用虚拟化，您必须安装虚拟化软件包并确保将您的系统配置为托管虚拟机(VM)。具体步骤根据您的 CPU 架构而有所不同。

2.1. 在 AMD64 和 INTEL 64 中启用虚拟化

要在运行 RHEL 9 的 AMD64 或者 Intel 64 系统中设置 KVM 管理程序并创建虚拟机(VM)，请按照以下步骤操作。

先决条件

- Red Hat Enterprise Linux 9 已在主机中安装并注册。
- 您的系统满足以下硬件要求以作为虚拟主机工作：
  - 主机的构架支持 KVM 虚拟化。
  - 有以下最小系统资源可用：
    - 主机有 6 GB 可用磁盘空间，以及每个预期的虚拟机需要额外 6 GB 空间。
    - 主机需要 2 GB RAM，以及每个预期的虚拟机需要额外 2 GB。

流程

1. 安装虚拟化 hypervisor 软件包。

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

2. 启动虚拟化服务：

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

验证

1. 确认您的系统已准备好成为虚拟化主机：

```
# 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. 如果所有 **virt-host-validate** 检查返回 **PASS** 值，则您的系统已准备好 创建虚拟机 。
   如果有任何检查返回 **FAIL** 值，请按照显示的说明来修复问题。

   如果有任何检查返回 **WARN** 值，请考虑按照显示的说明改进虚拟化功能。

故障排除

- 如果您的主机 CPU 不支持 KVM 虚拟化，**virt-host-validate** 会产生以下输出：

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

但是，在这样的主机系统上的虚拟机将无法引导，而不存在性能问题。

要临时解决这个问题，您可以将虚拟机的 XML 配置中的 **<domain type>** 的值改为 **qemu**。但请注意，红帽不支持使用 **qemu** 域类型的虚拟机，在生产环境中不建议这样做。

后续步骤

- 在 RHEL 9 主机上创建虚拟机

2.2. 在 IBM Z 中启用虚拟化

要在运行 RHEL 9 的 IBM Z 系统上设置 KVM 管理程序并创建虚拟机(VM)，请按照以下步骤操作。

先决条件

- 有以下最小系统资源可用：
  - 主机有 6 GB 可用磁盘空间，以及每个预期的虚拟机需要额外 6 GB 空间。
  - 主机需要 2 GB RAM，以及每个预期的虚拟机需要额外 2 GB。
  - 主机上有 4 个 CPU。虚拟机通常可以使用单个分配的 vCPU 运行，但红帽建议为每个虚拟机分配 2 个或更多 vCPU，以避免虚拟机在高负载期间变得无响应。
- 您的 IBM Z 主机系统使用 z13 CPU 或更高版本。
- RHEL 9 安装在逻辑分区(LPAR)上。另外，LPAR 支持 *启动阶段执行*（SIE）虚拟化功能。要进行验证，请在 **/proc/cpuinfo** 文件中搜索 **sie**。

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

流程

1. 安装虚拟化软件包：

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

2. 启动虚拟化服务：

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

验证

1. 验证您的系统已准备好成为虚拟化主机。

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

2. 如果所有 **virt-host-validate** 检查返回 **PASS** 值，则您的系统已准备好 创建虚拟机 。
如果有任何检查返回 **FAIL** 值，请按照显示的说明来修复问题。

   如果有任何检查返回 **WARN** 值，请考虑按照显示的说明改进虚拟化功能。

故障排除

- 如果您的主机 CPU 不支持 KVM 虚拟化，**virt-host-validate** 会产生以下输出：

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

  但是，在这样的主机系统上的虚拟机将无法引导，而不存在性能问题。

  要临时解决这个问题，您可以将虚拟机的 XML 配置中的 **<domain type>** 的值改为 **qemu**。但请注意，红帽不支持使用 **qemu** 域类型的虚拟机，在生产环境中不建议这样做。

其他资源

- IBM Z 上的虚拟化与 AMD64 和 Intel 64 有什么不同

2.3. 在 ARM 64 中启用虚拟化

要在运行 RHEL 9 的 ARM 64 系统（也称为 **AArch64**）上建立一个用于创建虚拟机(VM)的 KVM hypervisor，请按照以下指令操作。

先决条件

- 您的主机系统和客户机系统使用具有 64 KB 内存页大小的内核。要在 RHEL 系统上安装这样的内核，请参阅 在带有 内核-64k 的 ARM 上安装 RHEL 。
- 有以下最小系统资源可用：
  - 6 GB 的可用磁盘空间用于主机，以及每个预期的客户机都需要额外 6 GB 空间。
  - 主机需要 4 GB RAM，以及每个预期的客户机都需要 4 GB。

流程

1. 安装虚拟化软件包：

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

2. 启动虚拟化服务：

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

验证

1. 确认您的系统已准备好成为虚拟化主机：

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

2. 如果所有 **virt-host-validate** 检查都返回 **PASS** 值，则代表您的系统已准备好创建虚拟机。
   如果有任何检查返回 **FAIL** 值，请按照显示的说明来修复问题。

   如果有任何检查返回 **WARN** 值，请考虑按照显示的说明改进虚拟化功能。

后续步骤

- 创建虚拟机

其他资源

- ARM 64 上的虚拟化与 AMD64 和 Intel 64 上的虚拟化有何不同

2.4. 在虚拟机上启用 QEMU 客户机代理功能

要在 RHEL 9 系统上托管的虚拟机(VM)上使用某些功能，您必须首先配置虚拟机，以使用 QEMU 客户机代理(GA)。

有关这些功能的完整列表，请参阅 需要 QEMU 客户机代理的虚拟化功能。

在虚拟机上配置 QEMU GA 所需的具体步骤因虚拟机使用的客户机操作系统而异：

- 对于 Linux 虚拟机，请参阅 在 Linux 客户机上启用 QEMU 客户机代理 。
- 对于 Windows 虚拟机，请参阅 在 Windows 客户机上启用 QEMU 客户机代理 。

2.4.1. 在 Linux 客户机上启用 QEMU 客户机代理

要允许 RHEL 主机在 Linux 虚拟机(VM)上执行 某些操作的子集，您必须启用 QEMU 客户机代理(GA)。

您可以在运行和关闭的虚拟机上启用 QEMU GA。

流程

1. 为 QEMU GA 创建一个 XML 配置文件，例如 **qemuga.xml** ：

```
# touch qemuga.xml
```

2. 在文件中添加以下行：

```
<channel type='unix'>
  <source mode='bind' path='/var/lib/libvirt/qemu/f16x86_64.agent'/>
  <target type='virtio' name='org.qemu.guest_agent.0'/>
</channel>
```

3. 使用 XML 文件将 QEMU GA 添加导虚拟机的配置中。

   - 如果虚拟机正在运行，请使用以下命令：

   ```
   # virsh attach-device <vm-name> qemuga.xml --live --config
   ```

   - 如果虚拟机已关闭，请使用以下命令：

   ```
   # virsh attach-device <vm-name> qemuga.xml --config
   ```

4. 在 Linux 客户机操作系统中，安装 QEMU GA：

```
# dnf install qemu-guest-agent
```

5. 在客户机上启动 QEMU GA 服务：

```
# systemctl start qemu-guest-agent
```

验证

要确保 QEMU GA 已启用并在 Linux 虚拟机上运行，请执行以下操作之一：

- 在客户机操作系统中，请使用 **systemctl status qemu-guest-agent | grep Loaded** 命令。如果输出包括 **enabled**，则 QEMU GA 在虚拟机上处于活跃状态。
- 在主机上使用 **virsh domfsinfo *<vm-name>*** 命令。如果其显示任何输出，则 QEMU GA 在指定虚拟机上处于活跃状态。

其他资源

- 需要 QEMU 客户机代理的虚拟化功能

2.4.2. 在 Windows 客户机上启用 QEMU 客户机代理

要允许 RHEL 主机在 Windows 虚拟机(VM)上执行 某些操作的子集，您必须启用 QEMU 客户机代理(GA)。要做到这一点，将包含 QEMU 客户机代理安装程序的存储设备添加到现有虚拟机上或在创建新虚拟机时，在 Windows 客户机操作系统上安装驱动程序。

要使用图形界面安装客户机代理(GA)，请参阅以下流程 。要在命令行界面中安装 GA，请使用 Microsoft Windows 安装程序(MSI)。

先决条件

- 带有客户机代理的安装介质被附加到虚拟机。有关准备该介质的步骤，请参考在主机中准备 virtio 驱动程序安装介质。

流程

1. 在 Windows 客户机操作系统中，打开 **File Explorer** 应用程序。
2. 单击 **This PC**。
3. 在 **Devices and drives** 窗格中，打开 **virtio-win** 介质。
4. 打开 **guest-agent** 文件夹。
5. 根据虚拟机上安装的操作系统，运行以下安装程序之一：
   - 如果使用 32 位操作系统，请运行 **qemu-ga-i386.msi** 安装程序。
   - 如果使用 64 位操作系统，请运行 **qemu-ga-x86_64.msi** 安装程序。
6. **可选** ：如果您要使用半虚拟化串行驱动程序(**virtio-serial**)作为主机和 Windows 客户机之间的通信接口，请验证 **virtio-serial** 驱动程序是否已安装在 Windows 客户机上。有关安装 **virtio** 驱动程序的更多信息，请参阅 在 Windows 客户端上安装 virtio 驱动程序 。

验证

1. 在 Windows 虚拟机上，进入到 **Services** 窗口。
   computer Management > Services
2. 确保 **QEMU 客户机代理** 服务的状态为 **Running**。

其它资源

- 需要 QEMU 客户机代理的虚拟化功能

2.4.3. 需要 QEMU 客户机代理的虚拟化功能

如果您在虚拟机(VM)上启用了 QEMU 客户机代理(GA)，您可以在主机上使用以下命令来管理虚拟机：

virsh shutdown --mode=agent

这个关闭方法比 **virsh shutdown --mode=acpi** 更可靠，因为与 QEMU GA 一起使用的 **virsh shutdown** 可以保证在干净的状态下关闭合作客户机。

virsh domfsfreeze 和 virsh domfsthaw

冻结处于隔离状态的客户机文件系统。

virsh domfstrim

指示客户机修剪其文件系统，这有助于减少迁移期间需要传输的数据。

重要

如果要使用这个命令管理 Linux 虚拟机，您还必须在客户机操作系统中设置以下 SELinux 布尔值：

```
# setsebool virt_qemu_ga_read_nonsecurity_files on
```

virsh domtime

查询或设置客户机时钟。

virsh setvcpus --guest

指示客户机将 CPU 离线，这在无法热拔 CPU 时非常有用。

virsh domifaddr --source agent

使用 QEMU GA 查询客户机操作系统的 IP 地址。例如，当客户机接口直接连接到主机接口时，这非常有用。

virsh domfsinfo

显示正在运行的客户机中挂载的文件系统的列表。

virsh set-user-password

在客户机中设置给定用户帐户的密码。

virsh set-user-sshkeys

客户机中编辑给定用户的授权的 SSH 密钥文件。

重要

如果要使用这个命令管理 Linux 虚拟机，您还必须在客户机操作系统中设置以下 SELinux 布尔值：

```
# setsebool virt_qemu_ga_manage_ssh on
```

其它资源

- 在 Linux 客户机上启用 QEMU 客户机代理
- 在 Windows 客户机上启用 QEMU 客户机代理

第 3 章 创建虚拟机

要在 RHEL 9 中创建虚拟机(VM)，请使用命令行界面（CLI）或 RHEL 9 web 控制台。

3.1. 使用命令行界面创建虚拟机

要使用 **virt-install** 工具在 RHEL 9 主机上创建虚拟机(VM)，请按照以下说明操作。

先决条件

- 虚拟化已在您的主机系统中启用。
- 您有足够的系统资源来分配给虚拟机，如磁盘空间、RAM 或 CPU。根据虚拟机的预期任务和工作负载，推荐的值可能会有很大不同。
- 操作系统（OS）安装源可存在于本地或者网络中。可以是以下之一：
  - 安装介质的 ISO 镜像
  - 现有虚拟机安装的磁盘镜像

> **警告**
>
> 在 RHEL 9 中无法从主机 CD-ROM 或者 DVD-ROM 设备安装。当使用 RHEL 9 中的任何虚拟机安装方法时，如果选择了 CD-ROM 或者 DVD-ROM 作为安装源，则安装将失败。如需更多信息，请参阅红帽知识库。

另请注意，红帽只对 一组有限的客户机操作系统 提供支持。

- 可选：对于快速、简单的配置安装，可以使用 Kickstart 文件。

流程

要创建虚拟机并启动其操作系统安装，请使用 **virt-install** 命令以及以下强制参数：

- **--name**: 新机器的名称
- **--memory** ：分配的内存量
- **--vcpus** ：分配的虚拟 CPU 数
- **--disk** ：分配的存储的类型和大小
- **--cdrom** 或 **--location** ：操作系统安装源的类型和位置

根据所选安装方法，所需选项和值可能会有所不同。请参阅以下命令的示例：

- 以下命令创建一个名为 **demo-guest1** 的虚拟机，它从本地存储在 **/home/username/Downloads/Win10install.iso** 文件中的 ISO 镜像安装 Windows 10 OS。此虚拟机还可分配 2048 MiB RAM 和 2 个 vCPU，为虚拟机自动配置 80 GiB qcow2 虚拟磁盘。

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

- 以下命令创建一个名为 **demo-guest2** 的虚拟机，它使用 **/home/username/Downloads/rhel9.iso** 镜像从 live CD 运行 RHEL 9 OS。没有为这个虚拟机分配磁盘空间，因此在此会话中所做的更改不会被保留。另外，虚拟机被分配 4096 MiB RAM 和 4 个 vCPU。

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

- 以下命令创建一个名为 **demo-guest3** 的 RHEL 9 虚拟机，它连接到现有磁盘镜像 **/home/username/backup/disk.qcow2**。这和在不同的机器间物理地移动硬盘驱动器类似，因此 demo-guest3 可用的操作系统和数据由之前处理镜像的方式决定。另外，这个虚拟机还会分配 2048 MiB RAM 和 2 个 vCPU。

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

请注意，在导入磁盘镜像时，强烈建议使用 **--os-variant** 选项。如果没有提供，创建虚拟机的性能将会受到负面影响。

- 以下命令创建一个名为 **demo-guest4** 的虚拟机，它从 **http://example.com/OS-install** URL 安装。要使安装成功启动，URL 必须包含可正常工作的操作系统安装树。另外，操作系统使用 **/home/username/ks.cfg** kickstart 文件自动进行配置。此虚拟机还可分配 2048 MiB RAM、2 个 vCPU 和 160 GiB qcow2 虚拟磁盘。

```
# 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"
```

另外，如果您要在 ARM 64 主机上的 RHEL 9 上托管 demo-guest4，请包含以下行，以确保 kickstart 文件安装 **kernel-64k** 软件包：

```
%packages
-kernel
kernel-64k
%end
```

- 以下命令创建一个名为 **demo-guest5** 的虚拟机，它以纯文本模式从 **RHEL9.iso** 镜像文件安装，而无需图形。它将客户端控制台连接到串行控制台。虚拟机有 16384 MiB 内存、16 个 vCPU 和 280 GiB 磁盘。当通过慢速网络连接连接到主机时这种安装很有用。

```
# 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'
```

- 以下命令创建一个名为 **demo-guest6** 的虚拟机，其与 demo-guest5 有相同的配置，但位于 192.0.2.1 远程主机上。

```
# virt-install \
    --connect qemu+ssh://root@192.0.2.1/system --name demo-guest6 --memory 16384 \
    --vcpus 16 --disk size=280 --os-variant rhel9.0 --location RHEL9.iso \
    --graphics none --extra-args='console=ttyS0'
```

- 以下命令创建一个名为 **demo-guest-7** 的虚拟机，其与 demo-guest5 有相同的配置，但对于其存储，它使用 DASD 介质设备 **mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8**，并给其分配了设备号 **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'
```

请注意，可以使用 **virsh nodedev-list --cap mdev** 命令来检索可用于安装的介质设备的名称。

验证

- 如果成功创建虚拟机，则使用虚拟机的图形控制台打开 virt-viewer 窗口并启动客户端操作系统安装。

故障排除

- 如果 **virt-install** 失败，且出现 **cannot find default network** 错误：
  - 确定 **libvirt-daemon-config-network** 软件包已安装：

```
# {PackageManagerCommand} info libvirt-daemon-config-network
Installed Packages
Name         : libvirt-daemon-config-network
[...]
```

  - 验证 **libvirt** 默认网络是否处于活动状态，并且已配置为自动启动：

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

  - 如果没有，激活默认网络并将其设置为 auto-start：

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

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

- 如果激活默认网络失败并显示以下错误，则代表还没有正确安装 **libvirt-daemon-config-network** 软件包。

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

  要解决这个问题，请重新安装 **libvirt-daemon-config-network** ：

  ```
  # {PackageManagerCommand} reinstall libvirt-daemon-config-network
  ```

- 如果激活默认网络失败并显示类似如下的错误，则默认网络子网和主机上现有接口之间出现了冲突。

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

  要解决这个问题，请使用 **virsh net-edit default** 命令，并将配置中的 **192.0.2.*** 值改为主机上未被使用的子网。

其它资源

- **virt-install (1)** 手册页
- 使用 web 控制台创建虚拟机并安装客户机操作系统
- 克隆虚拟机

3.2. 使用 WEB 控制台创建虚拟机并安装客户机操作系统

要在 RHEL 9 主机上的 GUI 中管理虚拟机(VM)，请使用 web 控制台。以下章节介绍了如何使用 RHEL 9 web 控制台创建虚拟机并在其上安装客户机操作系统的信息。

3.2.1. 使用 web 控制台创建虚拟机

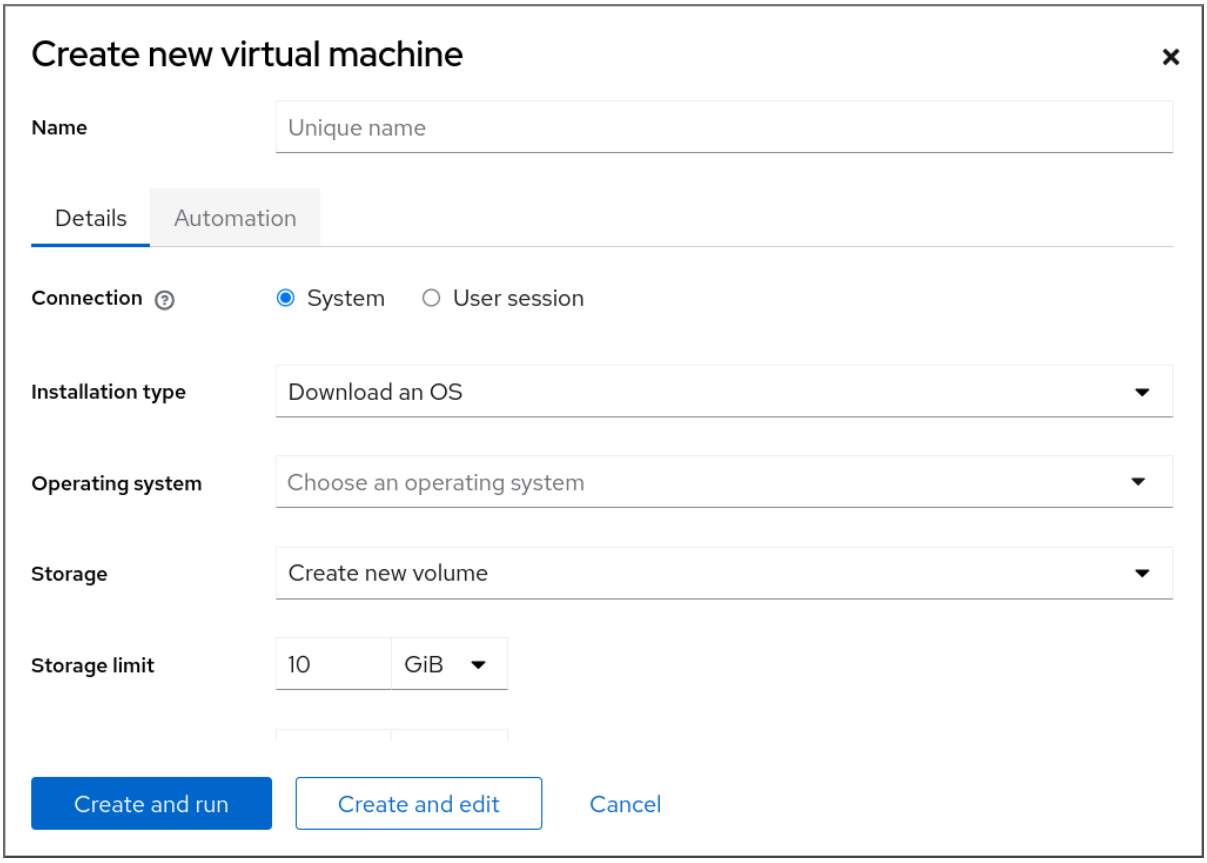
要在 RHEL 9 web 控制台连接的主机机器上创建虚拟机(VM)，请使用以下说明。

先决条件

- 虚拟化已在您的主机系统上启用。
- Web 控制台 VM 插件已安装在您的主机系统上。
- 您有足够的系统资源来分配给虚拟机，如磁盘空间、RAM 或 CPU。推荐的值可能会根据虚拟机的预期任务和工作负载而有很大的不同。

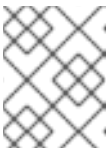
流程

1. 在 web 控制台的 **Virtual Machines** 界面中，点 **Create VM**。
   此时会出现 **Create new virtual machine** 对话框。



2. 输入您要创建的虚拟机的基本配置。

   - **Name** - 虚拟机的名称。
   - **Connection** - 授予会话的特权级别。如需了解更多详细信息，请展开 web 控制台中相关的对话框。
   - **Installation type** - 安装可以使用本地安装介质、URL、PXE 网络引导、云基础镜像，或者从有限的操作系统集合中下载操作系统。
   - **operating system** - 在虚拟机上运行的客户机操作系统。请注意，红帽只对一组有限的客户机操作系统提供支持。

     注意

     要从 web 控制台直接下载并安装 Red Hat Enterprise Linux，您必须在 **Offline token** 字段中添加一个离线令牌。

   - **Storage** - 存储的类型。
   - **Storage Limit** - 存储空间量。
   - **Memory** - 内存量。

3. 创建虚拟机：

   - 如果您希望虚拟机自动安装操作系统，点 **创建并运行**。
   - 如果要在安装操作系统前编辑虚拟机，点**创建并编辑**。

后续步骤

- 使用 Web 控制台安装客户机操作系统

其它资源

- 使用命令行界面创建虚拟机

3.2.2. 使用 web 控制台，通过导入磁盘镜像来创建虚拟机

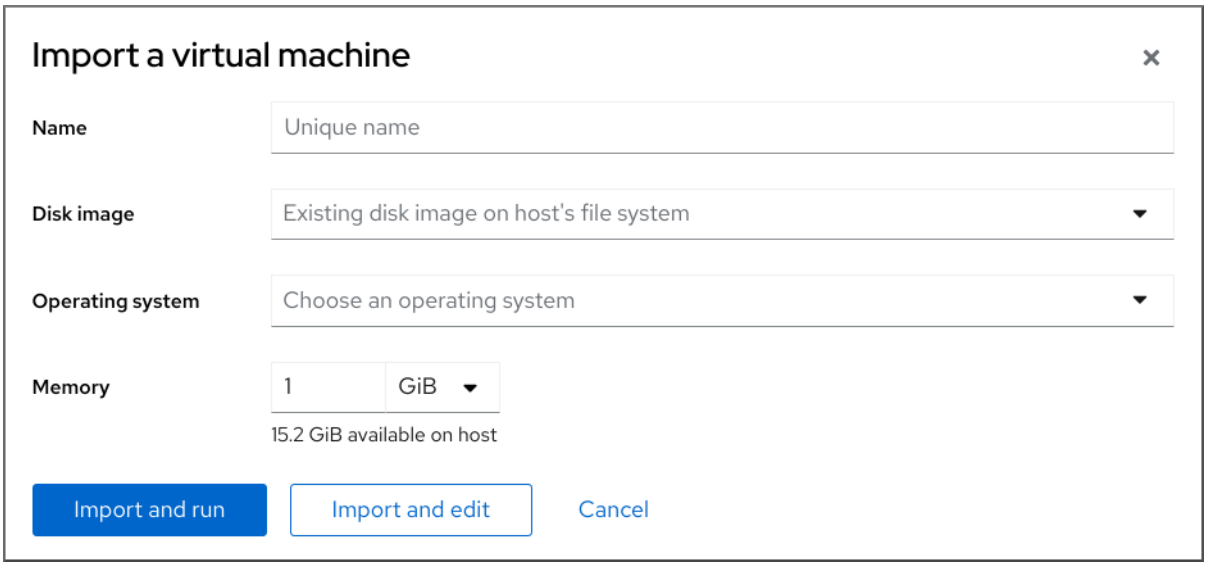
您可以通过在 RHEL 9 web 控制台中导入现有虚拟机安装的磁盘镜像来创建虚拟机(VM)。

先决条件

- Web 控制台 VM 插件已安装在您的系统上。
- 您有足够的系统资源来分配给虚拟机，如磁盘空间、RAM 或 CPU。根据虚拟机的预期任务和工作负载，推荐的值可能会有很大不同。
- 您已下载了现有虚拟机安装的磁盘镜像。

流程

1. 在 web 控制台的 **Virtual Machines** 界面中，单击 **Import VM**。
此时会出现 **Import a virtual machine** 对话框。

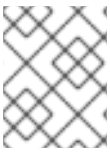


2. 输入您要创建的虚拟机的基本配置：
   - **Name** - 虚拟机的名称。
   - **Disk image** - 主机系统上虚拟机现有磁盘映像的路径。
   - **Operating system** - 运行在 VM 磁盘上的操作系统。请注意，红帽只对 一组有限的客户机操作系统提供支持。
   - **Memory** - 分配给虚拟机使用的内存量。
3. 导入虚拟机：
   - 要在虚拟机上安装操作系统，而无需对虚拟机设置进行额外的编辑，请点 **Import and run**。

- 要在安装操作系统前编辑虚拟机设置，请点 **Import and edit**。

3.2.3. 使用 Web 控制台安装客户机操作系统

当虚拟机(VM)第一次引导时，您必须在虚拟机上安装操作系统。



注意

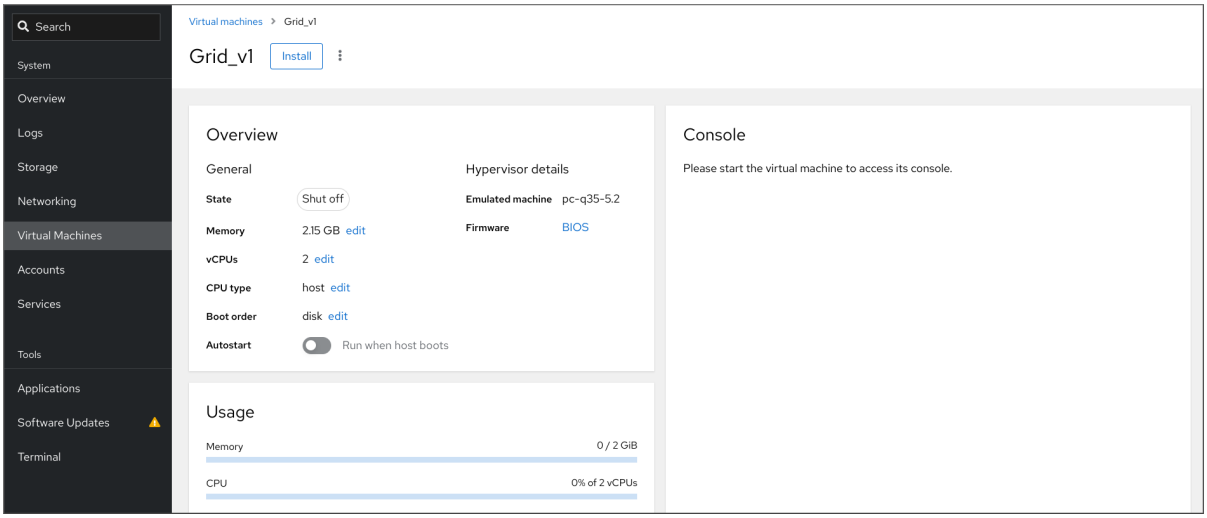
如果您在创建新虚拟机时点击了 **Create and run** 或 **Import and run**，则操作系统的安装例程会在虚拟机创建时自动启动。

先决条件

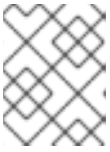
- Web 控制台 VM 插件已安装在您的主机系统上。

流程

1. 在 **Virtual Machines** 界面中，单击要在其上安装客户机操作系统的虚拟机。
此时将打开一个新页面，其中包含有关所选虚拟机的基本信息，以及管理虚拟机各方面的控制。



2. **可选**：修改固件。



注意

只有在创建新虚拟机时选择了 **Create and edit**或**Import and edit**，且操作系统还没有在虚拟机上安装，才能更改固件。

+ ..点击固件。

   a. 在 **Change Firmware** 窗口中，选择所需的固件。

   b. 点击 **Save**。

3. 点 **Install**。
在 VM 控制台中运行的操作系统的安装过程。

故障排除

- 如果安装例程失败，请在再次开始安装前删除并重新创建虚拟机。

3.2.4. 使用 web 控制台，使用云镜像身份验证创建虚拟机

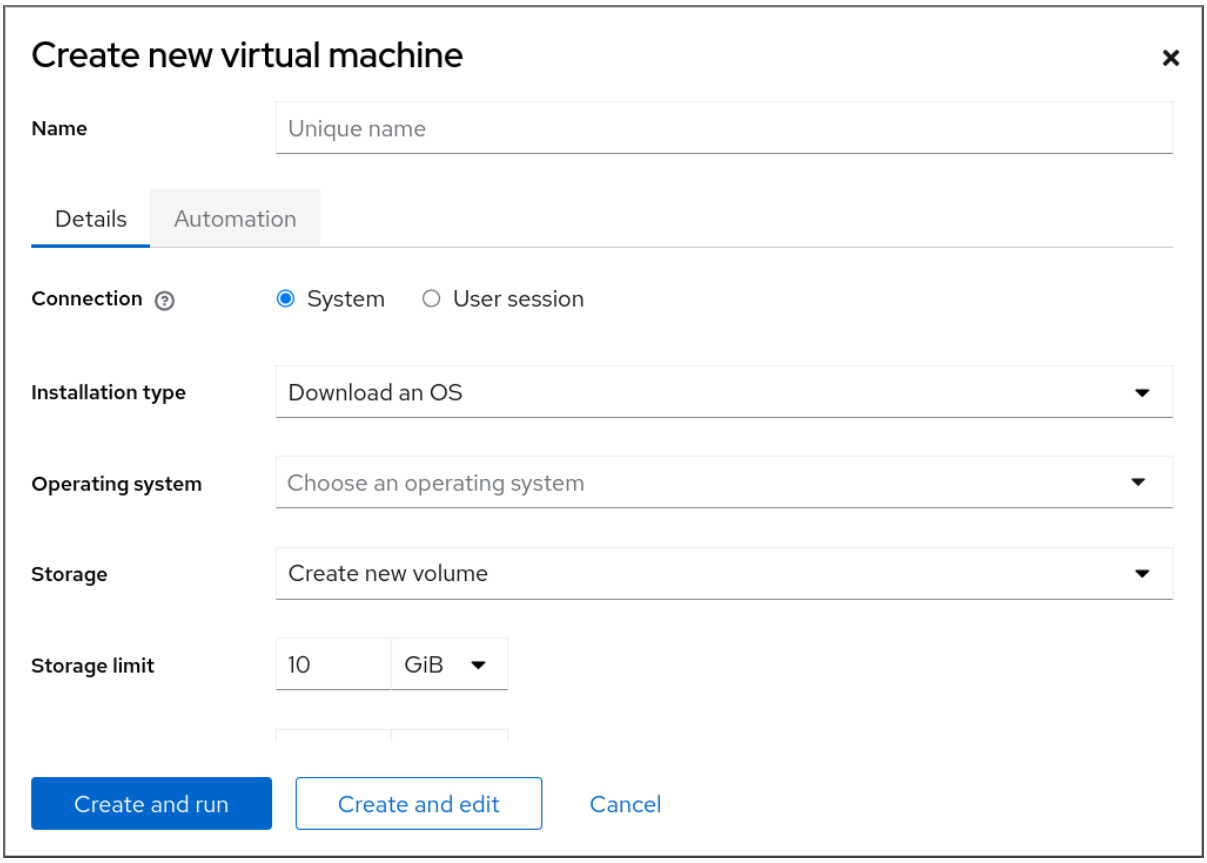
默认情况下，发行版云镜像没有登录帐户。但是，通过使用 RHEL web 控制台，您现在可以创建虚拟机 (VM)，并指定 root 和用户帐户登录凭证，然后传给 cloud-init。

先决条件

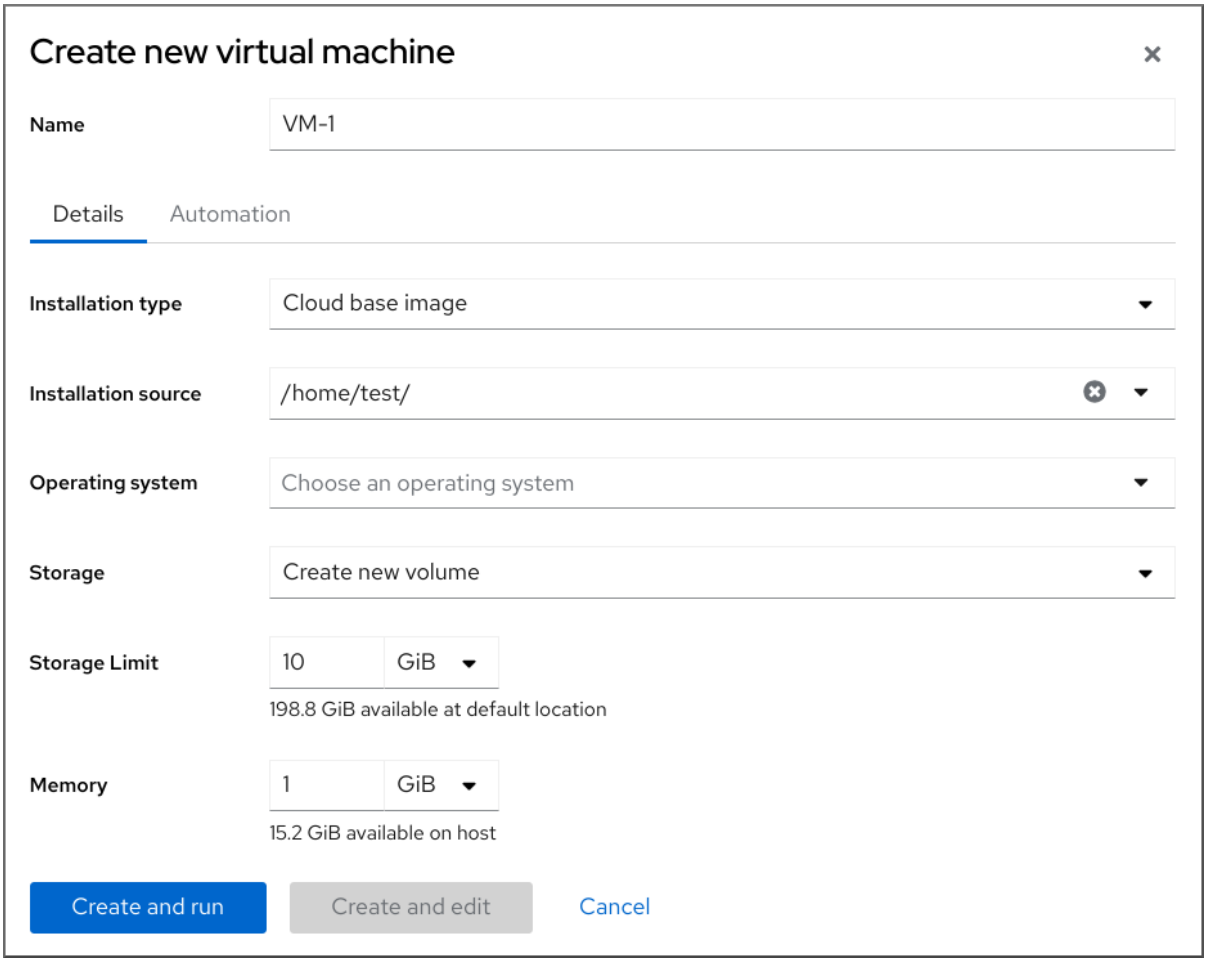
- Web 控制台 VM 插件 已安装在您的系统上。
- 虚拟化已在您的主机系统中启用。
- 您有足够的系统资源来分配给虚拟机，如磁盘空间、RAM 或 CPU。根据虚拟机的预期任务和工作负载，推荐的值可能会有很大不同。

流程

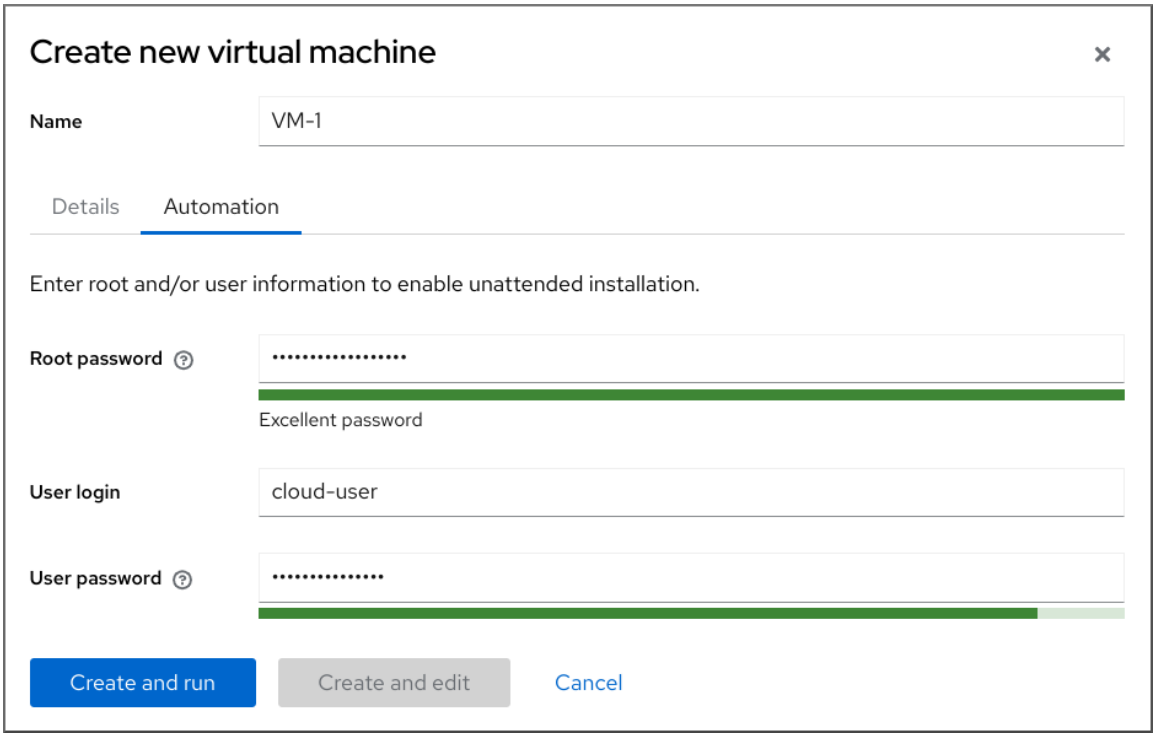
1. 在 web 控制台的 **Virtual Machines** 界面中，点 **Create VM**。
   此时会出现 Create new virtual machine 对话框。



2. 在 **Name** 字段中输入虚拟机的名称。
3. 在 **Details** 标签页的 **Installation type** 字段中选择 **Cloud base image**。



4. 在 **Installation source** 字段中，设置主机系统上镜像文件的路径。
5. 输入您要创建的虚拟机的配置。
   - **Operating system** - 虚拟机的操作系统.请注意，红帽只对一组有限的客户机操作系统提供支持。
   - **Storage** - 用于配置虚拟机的存储的类型。
   - **Storage Limit** - 配置虚拟机的存储空间量。
   - **Memory** - 用于配置虚拟机的内存量。
6. 点 **Automation** 标签页。
   设置云身份验证凭证。
   - **Root password** - 输入虚拟机的 root 密码。如果您不想设置 root 密码，请将该字段留空。
   - **User login** - 输入 cloud-init 用户登录名。如果您不想创建用户帐户，请将此字段留空。
   - **User password** - 输入密码.如果您不想创建用户帐户，请将此字段留空。



7. 点 **创建并运行**。
虚拟机已创建。

其它资源

- 在虚拟机上安装操作系统

第 4 章 启动虚拟机

要在 RHEL 9 中启动虚拟机（VM），您可以使用 命令行界面 或 web 控制台 GUI。

先决条件

- 在启动虚拟机前，它必须被创建，理想情况下，还要使用操作系统进行安装。有关操作的说明，请参阅创建虚拟机。

4.1. 使用命令行界面启动虚拟机

您可以使用命令行界面(CLI)启动关闭的虚拟机(VM)或恢复保存的虚拟机。通过使用 CLI，您可以启动本地和远程虚拟机。

先决条件

- 已定义的一个不活跃地虚拟机。
- 虚拟机的名称。
- 对于远程虚拟机：
  - 虚拟机所在主机的 IP 地址。
  - 对主机的 root 访问权限。

流程

- 对于本地虚拟机，请使用 **virsh start** 工具。
  例如，以下命令启动 *demo-guest1* 虚拟机。

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

- 对于位于远程主机上的虚拟机，请使用 **virsh start** 工具以及与主机的 QEMU+SSH 连接。
  例如，以下命令在 192.0.2.1 主机上启动 *demo-guest1* 虚拟机。

```
# virsh -c qemu+ssh://root@192.0.2.1/system start demo-guest1

root@192.0.2.1's password:

Domain 'demo-guest1' started
```

其它资源

- **virsh start --help** 命令
- 设置对远程虚拟化主机的简单访问
- 当主机启动时自动启动虚拟机

4.2. 使用 WEB 控制台启动虚拟机

如果虚拟机(VM)处于 *shut off* 状态，您可以使用 RHEL 9 web 控制台启动它。您还可以将虚拟机配置为在主机启动时自动启动。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。
- 已定义的一个不活跃地虚拟机。
- 虚拟机的名称。

流程

1. 在 **Virtual Machines** 界面中，点击您要启动的虚拟机。
   此时将打开一个新页面，其中包含有关所选虚拟机的详细信息，以及关闭和删除虚拟机的控制。
2. 点 **Run**。
   虚拟机启动，您可以连接到其控制台或图形输出。
3. **可选：** 要将虚拟机配置为在主机启动时自动启动，请切换 **Overview** 部分中的 **Autostart** 复选框。
   如果使用不由 libvirt 管理的网络接口，您还必须对 systemd 配置进行额外的更改。否则，受影响的虚拟机可能无法启动，请参阅 当主机启动时自动启动虚拟机。

其它资源

- 在 web 控制台中关闭虚拟机
- 使用 web 控制台重启虚拟机

4.3. 当主机启动时自动启动虚拟机

当一个带有运行的虚拟机(VM)的主机重启时，虚拟机将关闭，默认必须手动启动。要确保虚拟机在其主机运行时处于活跃状态，您可以将虚拟机配置为自动启动。

先决条件

- 已创建了虚拟机

流程

1. 使用 **virsh autostart** 工具将虚拟机配置为在主机启动时自动启动。
   例如，以下命令将 *demo-guest1* 虚拟机配置为自动启动：

```
# virsh autostart demo-guest1
Domain 'demo-guest1' marked as autostarted
```

2. 如果您使用不是由 **libvirt** 管理的网络接口，则也必须对 systemd 配置进行额外的更改。否则，受影响的虚拟机可能无法启动。

注意

例如，这些接口包括：

- **NetworkManager** 创建的网桥设备
- 网络配置为使用 **<forward mode='bridge'/>**

a. 在 systemd 配置目录树中，如果 **virtqemud.service.d** 目录尚不存在，则创建该目录。

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

b. 在之前创建的目录中创建一个 **10-network-online.conf** systemd 单元覆盖文件。此文件的内容覆盖 **virtqemud** 服务的默认 systemd 配置。

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

c. 将以下行添加到 **10-network-online.conf** 文件中：这个配置更改可确保 systemd 仅在主机上的网络就绪后启动 **virtqemud** 服务。

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

验证

1. 查看虚拟机配置，并检查是否启用了 *autostart* 选项。
例如，以下命令显示有关 *demo-guest1* 虚拟机的基本信息，包括 *autostart* 选项：

```
# 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. 如果您使用不由 libvirt 管理的网络接口，请检查 **10-network-online.conf** 文件的内容是否与以下输出匹配：

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

其它资源

- **virsh autostart --help** 命令
- 使用 web 控制台启动虚拟机。

第 5 章 连接至虚拟机

要在 RHEL 9 中与虚拟机(VM)交互，您需要通过以下方法之一连接它：

- 使用 Web 控制台界面时，请在 web 控制台界面中使用 Virtual Machines 窗格。如需更多信息，请参阅 使用 web 控制台与虚拟机进行交互。
- 如果您需要在不使用 Web 控制台的情况下与虚拟机图形显示交互，请使用 Virt Viewer 应用程序。详情请查看 使用 Virt Viewer 打开虚拟机图形控制台。
- 如果不需要图形显示，请使用 SSH 终端连接。
- 当使用网络无法从您的系统访问虚拟机时，请使用 virsh 控制台。

如果您要连接的虚拟机位于远程主机而不是本地主机，您可以选择配置您的系统以更方便地访问远程主机。

先决条件

- 已安装并启动您要与之交互的虚拟机。

5.1. 使用 WEB 控制台与虚拟机进行交互

要在 RHEL 9 web 控制台中与虚拟机(VM)交互，您需要连接到虚拟机的控制台。这包括图形和串行控制台。

- 要在 web 控制台中与虚拟机的图形界面交互，请使用图形控制台。
- 要在远程 viewer 中与虚拟机的图形界面交互，请使用remote viewers 中的图形控制台。
- 要在 web 控制台中与虚拟机的 CLI 交互，请使用 串行控制台。

5.1.1. 在 web 控制台中查看虚拟机图形控制台

通过使用虚拟机(VM)控制台界面，您可以在 RHEL 9 web 控制台中查看所选虚拟机的图形输出。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。
- 确保主机和虚拟机支持图形界面。

流程

1. 在 **Virtual Machines** 界面中，单击您要查看其图形控制台的虚拟机。
   此时将打开一个新页面，其中包含虚拟机的 **Overview** 和 **Console** 部分。
2. 在控制台下拉菜单中选择 **VNC 控制台**。
   VNC 控制台在 Web 界面中的菜单下方显示。

   图形控制台会出现在 web 界面中。
3. 单击 **Expand**
   现在，您可以使用鼠标和键盘以与实际机器进行交互的相同方式与虚拟机控制台进行交互。VM 控制台中的显示反映了虚拟机上正在执行的操作。

注意

运行 web 控制台的主机可能会截获特定的组合键，如 **Ctrl+Alt+Del**，阻止它们发送到虚拟机。

要发送此类组合键，请单击 **Send key** 菜单并选择要发送的键序列。

例如，要将 **Ctrl+Alt+Del** 组合发送给虚拟机，请单击 **Send key** 并选择 **Ctrl+Alt+Del** 菜单条目。

故障排除

- 如果在图形控制台中点击没有任何效果，请将控制台扩展为全屏。这是一个已知的鼠标光标偏移问题。

其它资源

- 使用 Web 控制台在远程查看器中查看图形控制台
- 在 web 控制台中查看虚拟机串口控制台

5.1.2. 使用 Web 控制台在远程查看器中查看图形控制台

通过使用 web 控制台界面，您可以在远程查看器（如 Virt Viewer）中显示所选虚拟机(VM)的图形控制台。

注意

您可以在 web 控制台中启动 Virt Viewer。其他 VNC 远程查看器可以被手动启动。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。
- 确保主机和虚拟机支持图形界面。
- 在 Virt Viewer 中查看图形控制台前，您必须在 web 控制台连接的机器上安装 Virt Viewer。

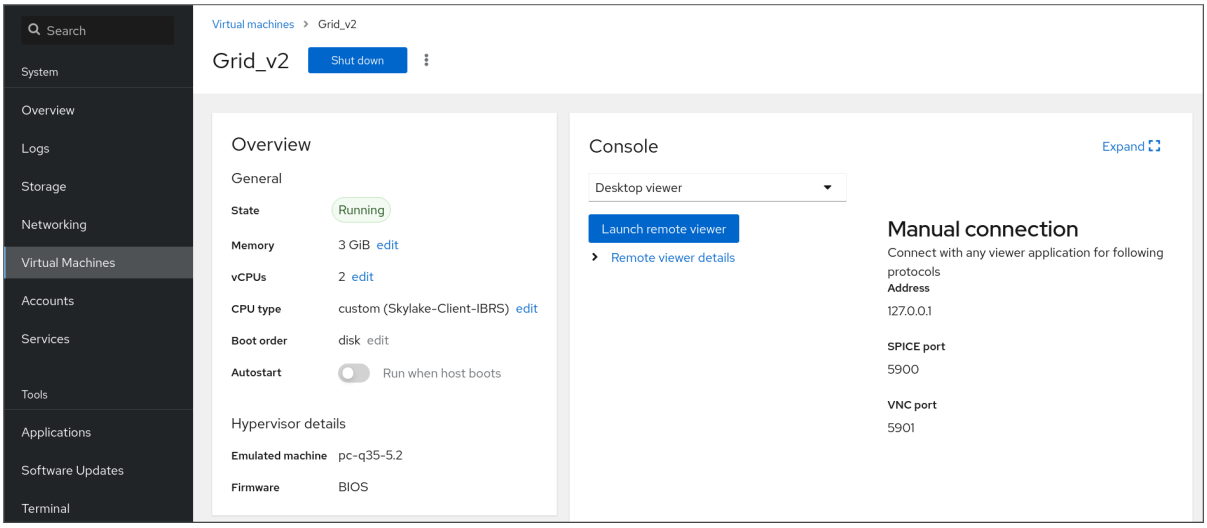
1. 单击 **Launch remote viewer**。
   virt viewer **.vv**，文件下载。
2. 打开文件以启动 Virt Viewer。

注意

远程查看器在大多数操作系统上提供。但是，一些浏览器扩展和插件不允许 Web 控制台打开 Virt Viewer。

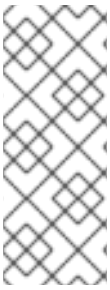
流程

1. 在 **Virtual Machines** 界面中，单击您要查看其图形控制台的虚拟机。
   此时将打开一个新页面，其中包含虚拟机的 **Overview** 和 **Console** 部分。
2. 在控制台下拉菜单中选择 **Desktop Viewer**。



3. 点击 **Launch Remote Viewer**。
图形控制台在 Virt Viewer 中打开。

您可以使用鼠标和键盘以与实际机器进行交互的相同方式与虚拟机控制台进行交互。VM 控制台中的显示反映了虚拟机上正在执行的操作。



注意

运行 web 控制台的服务器可以截获特定的组合键，如 **Ctrl+Alt+Del**，阻止它们发送到虚拟机。

要发送这样的组合键，点 **Send key** 菜单并选择要使用地键序列。

例如：要将 **Ctrl+Alt+Del** 组合发送到 VM，点 **Send key** 菜单并选择 **Ctrl+Alt+Del** 菜单。

故障排除

- 如果在图形控制台中点击没有任何效果，请将控制台扩展为全屏。这是一个已知的鼠标光标偏移问题。
- 如果在 web 控制台中启动远程查看器无法正常工作，或者不佳，您可以使用以下协议手动与任何查看器应用程序进行连接：
  - **Address** - 默认地址为 **127.0.0.1**。您可以修改 **/etc/libvirt/qemu.conf** 中的 **vnc_listen** 参数，将它更改为主机的 IP 地址。
  - **VNC port** - 5901

其它资源

- 在 web 控制台中查看虚拟机图形控制台
- 在 web 控制台中查看虚拟机串口控制台

5.1.3. 在 web 控制台中查看虚拟机串口控制台

您可以在 RHEL 9 web 控制台中查看所选虚拟机(VM)的串行控制台。这在主机机器或者虚拟机没有使用图形界面配置时很有用。

有关串行控制台的更多信息，请参阅打开虚拟机串口控制台。

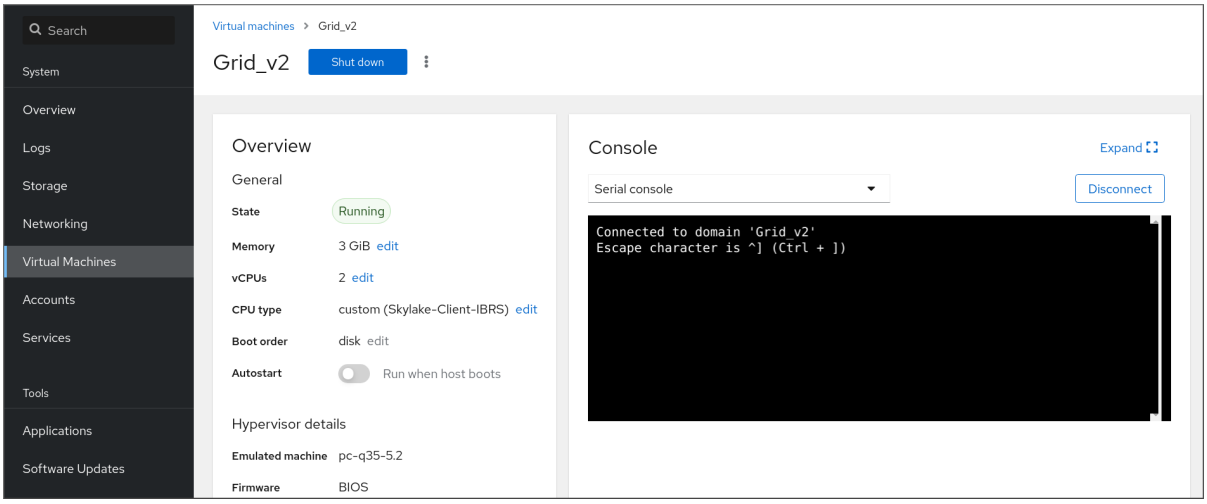
先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 在 **Virtual Machines** 窗格中，单击您要查看其串行控制台的虚拟机。
   此时将打开一个新页面，其中包含虚拟机的 **Overview** 和 **Console** 部分。

2. 在控制台下拉菜单中选择 **Serial console**。
   图形控制台会出现在 web 界面中。



您可以断开串行控制台与虚拟机的连接和重新连接。

- 要断开串行控制台与虚拟机的连接，请点 **Disconnect**。
- 要将串行控制台重新连接到虚拟机，请点 **Reconnect**。

其它资源

- 在 web 控制台中查看虚拟机图形控制台
- 使用 Web 控制台在远程查看器中查看图形控制台

5.1.4. 在 web 控制台中使用 VNC 替换 SPICE 远程显示协议

已在 RHEL 9 主机上删除了对 SPICE 远程显示协议的支持。如果您有一个配置为使用 SPICE 协议的虚拟机(VM)，您可以使用 web 控制台，将 SPICE 协议替换为 VNC 协议。否则，虚拟机无法启动。但是，某些 SPICE 设备（如音频和 USB passthrough）将从虚拟机中删除，因为它们在 VNC 协议中没有合适的替代品。

重要

默认情况下，RHEL 8 虚拟机被配置为使用 SPICE 协议。在 RHEL 9 主机上，这些虚拟机无法启动，除非从 SPICE 切换到 VNC。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

- 您有一个现有的配置为使用 SPICE 远程显示协议，且已关闭的虚拟机。

流程

1. 在 web 控制台的 Virtual Machines 界面中，点配置为使用 SPICE 协议的虚拟机的菜单按钮 ⋮。
此时会打开一个下拉菜单，其中包含对各种虚拟机操作的控制。
2. 单击 **Replace SPICE devices**。
此时会打开 Replace SPICE devices 对话框。

注意

如果您有多个使用 SPICE 协议的现有虚拟机，则它们都列在此对话框中。这里，您可以在一步中选择多个要从使用 SPICE 转换到使用 VNC 的虚拟机。

3. 点 **Replace**。
此时会出现一个操作成功的确认。

5.2. 使用 VIRT VIEWER 打开虚拟机图形控制台

要连接到 KVM 虚拟机(VM)的图形控制台并在 **Virt Viewer** 桌面应用程序中打开它，请按照以下流程操作。

先决条件

- 您的系统以及您要连接的虚拟机必须支持图形显示。
- 如果目标虚拟机位于远程主机上，则需要对主机有连接和 root 访问权限。
- **可选：** 如果目标虚拟机位于远程主机上，请设置 libvirt 和 SSH 以更方便地访问远程主机。

流程

- 要连接到本地虚拟机，请使用以下命令，并将 *guest-name* 替换为您要连接的虚拟机的名称：

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

- 要连接到远程虚拟机，请使用 **virt-viewer** 命令及 SSH 协议。例如，以下命令以 root 用户身份连接到位于远程系统 192.0.2.1 上的名为 *guest-name* 的虚拟机。连接还需要 192.0.2.1 的 root 身份验证。

```
# virt-viewer --direct --connect qemu+ssh://root@192.0.2.1/system guest-name
root@192.0.2.1's password:
```

验证

如果连接正常工作，则虚拟机将显示在 **Virt Viewer** 窗口中。

您可以使用鼠标和键盘以与实际机器进行交互的相同方式与虚拟机控制台进行交互。VM 控制台中的显示反映了虚拟机上正在执行的操作。

故障排除

- 如果在图形控制台中点击没有任何效果，请将控制台扩展为全屏。这是一个已知的鼠标光标偏移问题。

其它资源

- **virt-viewer** 手册页
- 设置对远程虚拟化主机的简单访问
- 使用 web 控制台与虚拟机进行交互

5.3. 使用 SSH 连接到虚拟机

要使用 SSH 连接协议与虚拟机(VM)终端进行交互，请按照以下流程操作。

先决条件

- 有对目标虚拟机的网络连接和 root 访问权限。
- 如果目标虚拟机位于远程主机上，您也可以拥有对该主机的连接和 root 访问权限。
- 您的虚拟机网络由 **libvirt** 生成的 **dnsmasq** 分配 IP 地址。这是 **libvirt** NAT 网络 中的示例。值得注意的是，如果您的虚拟机使用以下网络配置之一，则无法使用 SSH 连接到虚拟机：
  - **hostdev** 接口
  - 直接接口
  - 网桥接口
- 在虚拟机主机上已安装并启用了 **libvirt-nss** 组件。如果没有，请执行以下操作：

  a. 安装 **libvirt-nss** 软件包：

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

  b. 编辑 **/etc/nsswitch.conf** 文件，并将 **libvirt_guest** 添加到 **hosts** 行中：

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

流程

1. 当连接到远程虚拟机时，请首先 SSH 到其物理主机。以下示例演示了如何使用其 root 凭证连接到主机 **192.0.2.1** ：

   ```
   # ssh root@192.0.2.1
   root@192.0.2.1's password:
   Last login: Mon Sep 24 12:05:36 2021
   root~#
   ```

2. 使用虚拟机的名称和用户访问凭证来连接它。例如，以下命令使用其 root 凭证连接到 **testguest1** 虚拟机：

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

故障排除

- 如果您不知道虚拟机的名称，您可以使用 **virsh list --all** 命令列出主机上所有可用的虚拟机：

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

其它资源

- 上游 libvirt 文档

5.4. 打开虚拟机串口控制台

通过使用 **virsh console** 命令，可以连接到虚拟机(VM)的串行控制台。

但虚拟机有以下情况时很有用：

- 没有提供 VNC 协议，因此没有为 GUI 工具提供视频显示。
- 没有网络连接，因此无法使用 SSH 进行交互。

先决条件

- 主机上的 GRUB 引导装载程序必须配置为使用串口控制台。要进行验证，请检查主机上的 **/etc/default/grub** 文件是否包含 **GRUB_TERMINAL=serial** 参数。

```
$ sudo grep GRUB_TERMINAL /etc/default/grub
GRUB_TERMINAL=serial
```

- 虚拟机必须配有一个串口控制台设备，如 **console type='pty'**。要进行验证，请执行以下操作：

```
# virsh dumpxml vm-name | grep console

<console type='pty' tty='/dev/pts/2'>
</console>
```

- 虚拟机必须在内核命令行中配置串口控制台。要验证这一点，虚拟机上的 **cat /proc/cmdline** 命令输出应包含 *console=<console-name>*，其中 *<console-name>* 是特定的架构：
  - 对于 AMD64 和 Intel 64: **ttyS0**
  - 对于 ARM 64: **ttyAMA0**

注意

此流程中的以下命令使用 **ttyS0**。

```
# 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
```

如果没有在虚拟机中正确设置串口控制台，请使用 **virsh 控制台**连接到虚拟机，请将您连接到无响应的客户端控制台。但是，您仍然可以使用 **Ctrl+]** 快捷键退出无响应的控制台。

- 要在虚拟机上设置串行控制台，请执行以下操作：

  i. 在虚拟机上启用 **console=ttyS0** 内核选项：

  ```
  # grubby --update-kernel=ALL --args="console=ttyS0"
  ```

  ii. 清除可能会阻止您更改生效的内核选项。

  ```
  # grub2-editenv - unset kernelopts
  ```

  iii. 重启虚拟机。

- 必须启用 **serial-getty@*<console-name>*** 服务。例如，在 AMD64 和 Intel 64 上：

```
# systemctl status serial-getty@ttyS0.service

○ serial-getty@ttyS0.service - Serial Getty on ttyS0
   Loaded: loaded (/usr/lib/systemd/system/serial-getty@.service; enabled; preset: enabled)
```

流程

1. 在您的主机系统上，使用 **virsh console** 命令。如果 libvirt 驱动程序支持安全控制台处理，以下示例连接到 *guest1* 虚拟机：

```
# 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. 您还可以使用与标准命令行界面相同的方法与 virsh 控制台互动。

其它资源

- **virsh** 手册页

5.5. 设置对远程虚拟化主机的简单访问

当使用 libvirt 工具在远程主机系统上管理虚拟机时，建议使用 **-c qemu+ssh://root@hostname/system** 语法。例如，要在 **192.0.2.1** 主机上以 root 用户身份使用 **virsh list** 命令：

```
# virsh -c qemu+ssh://root@192.0.2.1/system list
root@192.0.2.1's password:

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

但是，您可以通过修改 SSH 和 libvirt 配置来删除完整指定连接详情的需要。例如：

```
# virsh -c remote-host list
root@192.0.2.1's password:

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

要进行改进，请按照以下步骤操作。

流程

1. 使用以下详细信息编辑 **~/.ssh/config** 文件，其中 *host-alias* 是与特定远程主机关联的短名称，以及 root@192.0.2.1 的别名，*hosturl* 是主机的 URL 地址：

```
# vi ~/.ssh/config
Host example-host-alias
  User                root
  Hostname            192.0.2.1
```

2. 使用以下详情编辑 **/etc/libvirt/libvirt.conf** 文件，*example-qemu-host-alias* 是一个主机别名，QEMU 和 libvirt 工具将用来为 **qemu+ssh://192.0.2.1/system** 与预期主机 *example-host-alias* 进行关联：

```
# vi /etc/libvirt/libvirt.conf
uri_aliases = [
  "example-qemu-host-alias=qemu+ssh://example-host-alias/system",
]
```

验证

1. 确认通过在本地系统上使用基于 libvirt 的工具，并添加 **-c *qemu-host-alias*** 参数，您可以来管理远程虚拟机。这会在远程主机中使用 SSH 自动执行命令。
例如，验证以下 192.0.2.1 远程主机上列出的虚拟机，在前面的步骤中到其的连接被设置为 *example-qemu-host-alias* ：

```
# virsh -c example-qemu-host-alias list

root@192.0.2.1's password:

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

–

注意

除了 **virsh** 之外，**-c**（或 **--connect**）选项以及上述远程主机访问配置也可以由以下工具使用：

- virt-install
- virt-viewer

后续步骤

如果要在单一远程主机中只使用 libvirt 工具，您也可以为基于 libvirt 的实用程序设置特定的连接作为默认目标。但是，如果您也要管理本地主机或不同远程主机上的虚拟机，则不建议这样做。

- 您可以编辑 **/etc/libvirt/libvirt.conf** 文件，并将 **uri_default** 参数的值设置为 *example-qemu-host-alias* 来作为默认 libvirt 目标。

```
# 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 = "example-qemu-host-alias"
```

因此，所有基于 libvirt 的命令都会在指定的远程主机中自动执行。

```
$ virsh list
root@192.0.2.1's password:

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

- 连接到远程主机时，您可以避免向远程系统提供 root 密码。要做到这一点，请使用以下一个或多个方法：
  - 设置对远程主机的基于密钥的 SSH 访问
  - 使用 SSH 连接多路复用连接到远程系统
  - 身份管理中的 Kerberos 身份验证
- **-c**（或 **--connect**）选项可用于在远程主机上运行 **virt-install**、**virt-viewer** 和 **virsh** 命令。

第 6 章 关闭虚拟机

要关闭 RHEL 9 上托管的正在运行的虚拟机，请使用命令行界面或 web 控制台 GUI。

6.1. 使用命令行界面关闭虚拟机

要关闭响应的虚拟机（VM），请执行以下操作之一：

- 在连接到客户端时，使用适合客户端操作系统的 shutdown 命令。
- 在主机上使用 **virsh shutdown** 命令：
  - 如果虚拟机位于本地主机上：

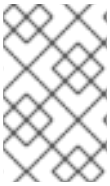
```
# virsh shutdown demo-guest1
Domain 'demo-guest1' is being shutdown
```

  - 如果虚拟机位于远程主机上，在本例中为 192.0.2.1 ：

```
# virsh -c qemu+ssh://root@192.0.2.1/system shutdown demo-guest1

root@192.0.2.1's password:
Domain 'demo-guest1' is being shutdown
```

要强制虚拟机关闭（例如，如果其机已变得无响应），请在主机上使用 **virsh destroy** 命令：

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



注意

virsh destroy 命令实际上不会删除虚拟机配置或磁盘镜像。它只会终止虚拟机的正在运行的虚拟机实例，类似于从物理机拔掉电源。因此，在个别情况下，**virsh destroy** 可能会导致虚拟机文件系统崩溃，因此仅在所有其他关闭方法都失败时才建议使用这个命令。

6.2. 使用 WEB 控制台关闭和重启虚拟机

使用 RHEL 9 web 控制台，您可以 关闭 或 重启 正在运行的虚拟机。您还可以向无响应的虚拟机发送不可屏蔽中断。

6.2.1. 在 web 控制台中关闭虚拟机

如果虚拟机(VM)处于 **running** 状态，您可以使用 RHEL 9 web 控制台关闭它。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 在 **Virtual Machines** 接口中，找到您要关闭的虚拟机行。

2. 在行的右侧，点击 **Shut Down**。
虚拟机关机。

故障排除

- 如果虚拟机没有关闭，请点 **Shut Down** 按钮旁边的 Menu ⋮ 按钮，然后选择 **Force Shut Down**。
- 要关闭无响应虚拟机，您还可以发送一个不可屏蔽的中断。

其它资源

- 使用 web 控制台启动虚拟机
- 使用 web 控制台重启虚拟机

6.2.2. 使用 web 控制台重启虚拟机

如果虚拟机(VM)处于 **running** 状态，您可以使用 RHEL 9 web 控制台重启它。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 在 **Virtual Machines** 接口中，找到您要重启的虚拟机行。
2. 在行的右侧，点击菜单按钮 ⋮。
此时会显示一个下拉操作菜单。
3. 在下拉菜单中，单击 **Reboot**。
虚拟机将关机并重启。

故障排除

- 如果虚拟机没有重启，点 **Reboot** 按钮旁边的 Menu ⋮ 按钮，然后选择 **Force Reboot**。
- 要关闭无响应虚拟机，您还可以发送一个不可屏蔽的中断。

其它资源

- 使用 web 控制台启动虚拟机
- 在 web 控制台中关闭虚拟机

6.2.3. 使用 web 控制台向虚拟机发送不可屏蔽中断

发送不可屏蔽中断（NMI）可能会导致无响应运行的虚拟机（VM）响应或关闭。例如，您可以将 **Ctrl**+**Alt**+**Del** NMI 发送到不响应标准输入的虚拟机。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 在 **Virtual Machines** 接口中，找到您要给其发送 NMI 的虚拟机行。
2. 在行的右侧，点击菜单按钮 ⋮。
   此时会显示一个下拉操作菜单。
3. 在下拉菜单中，点 **Send non-maskable interrupt**。
   一个 NMI 发送到虚拟机。

其它资源

- 使用 web 控制台启动虚拟机
- 使用 web 控制台重启虚拟机
- 在 web 控制台中关闭虚拟机

第 7 章 删除虚拟机

要删除 RHEL 9 中的虚拟机，请使用 命令行界面 或 Web 控制台 GUI。

7.1. 使用命令行界面删除虚拟机

要删除虚拟机(VM)，您可以使用命令行从主机中删除其 XML 配置和相关存储文件。按照以下步骤操作：

先决条件

- 备份虚拟机中的重要数据。
- 关闭虚拟机。
- 确保没有其他虚拟机使用相同的关联的存储。

流程

- 使用 **virsh undefine** 工具。
  例如：以下命令删除 *guest1* 虚拟机、与其关联的存储卷以及非电压 RAM（若有）。

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

其它资源

- **virsh undefine --help** 命令
- **virsh** 手册页

7.2. 使用 WEB 控制台删除虚拟机

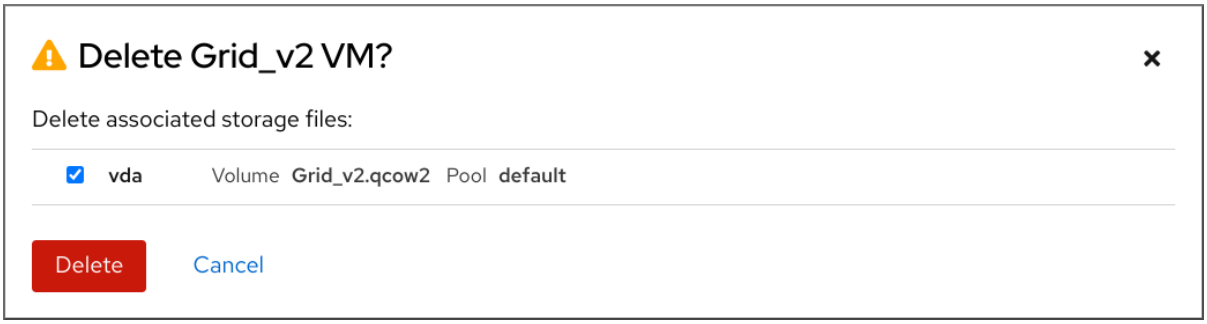
要从 RHEL 9 web 控制台连接的主机中删除虚拟机(VM)及其关联的存储文件，请按照以下步骤操作：

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。
- 备份虚拟机中的重要数据。
- 确保没有其他虚拟机使用相同的关联存储。
- **可选：** 关闭虚拟机。

流程

1. 在**虚拟机**界面中，点击您要删除 的虚拟机的 Menu 按钮 ⋮。
   此时会出现一个下拉菜单，控制各种虚拟机操作。
2. 点击 **Delete**。
   此时会出现确认对话框。

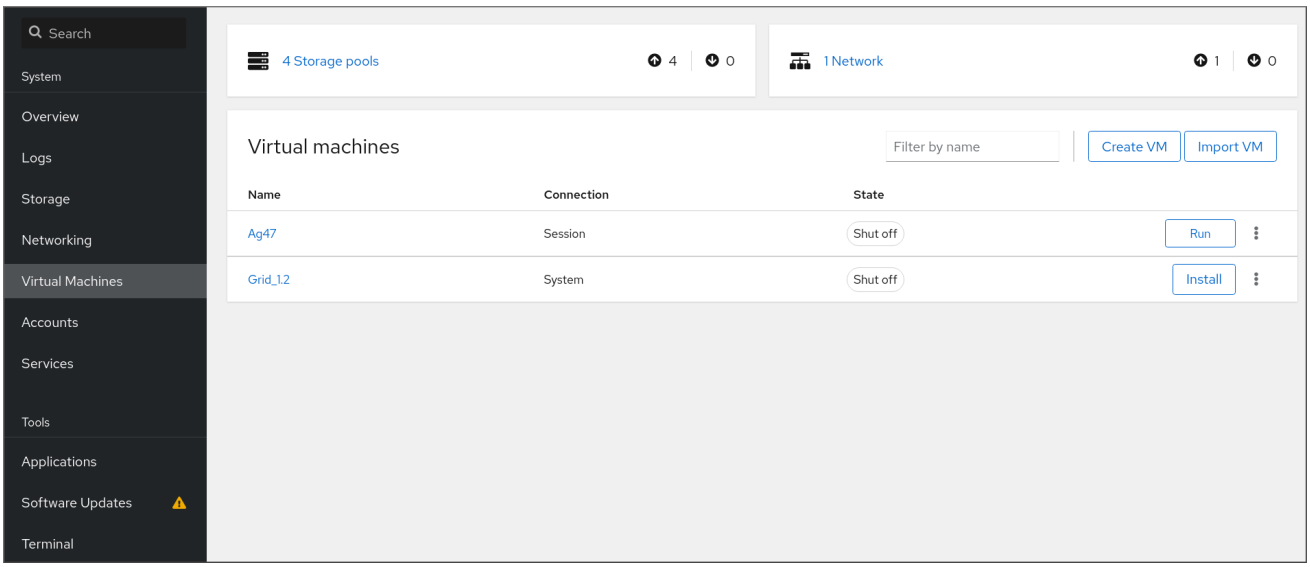


3. **可选：** 要删除与虚拟机关联的所有或部分存储文件，请选择您要删除的存储文件旁的复选框。

4. 点击 **Delete**。
虚拟机和任何选择的存储文件都将被删除。

第 8 章 在 WEB 控制台中管理虚拟机

要在 RHEL 9 主机上的图形界面管理虚拟机，您可以在 RHEL 9 web 控制台中使用 **Virtual Machines** 窗格。



8.1. 使用 WEB 控制台进行虚拟机管理的概述

RHEL 9 web 控制台是一个用于系统管理的基于 web 的界面。作为其功能之一，Web 控制台提供主机系统中虚拟机（VM）的图形视图，并可创建、访问和配置这些虚拟机。

请注意，要使用 Web 控制台在 RHEL 9 上管理虚拟机，您必须首先为虚拟化安装 web 控制台插件。

后续步骤

- 有关在 web 控制台中启用虚拟机管理的说明，请参阅 设置 Web 控制台来管理虚拟机。
- 有关 web 控制台提供的虚拟机管理操作的完整列表，请参阅 web 控制台中提供的虚拟机管理功能。

8.2. 设置 WEB 控制台以管理虚拟机

在使用 RHEL 9 web 控制台管理虚拟机 (VM) 之前，您必须在主机上安装 web 控制台虚拟机插件。

先决条件

- 确保机器上安装并启用了 Web 控制台。

```
# systemctl status cockpit.socket
cockpit.socket - Cockpit Web Service Socket
Loaded: loaded (/usr/lib/systemd/system/cockpit.socket
[...]
```

如果这个命令返回 **Unit cockpit.socket could not be found**，请按照安装 Web 控制台文档启用 Web 控制台。

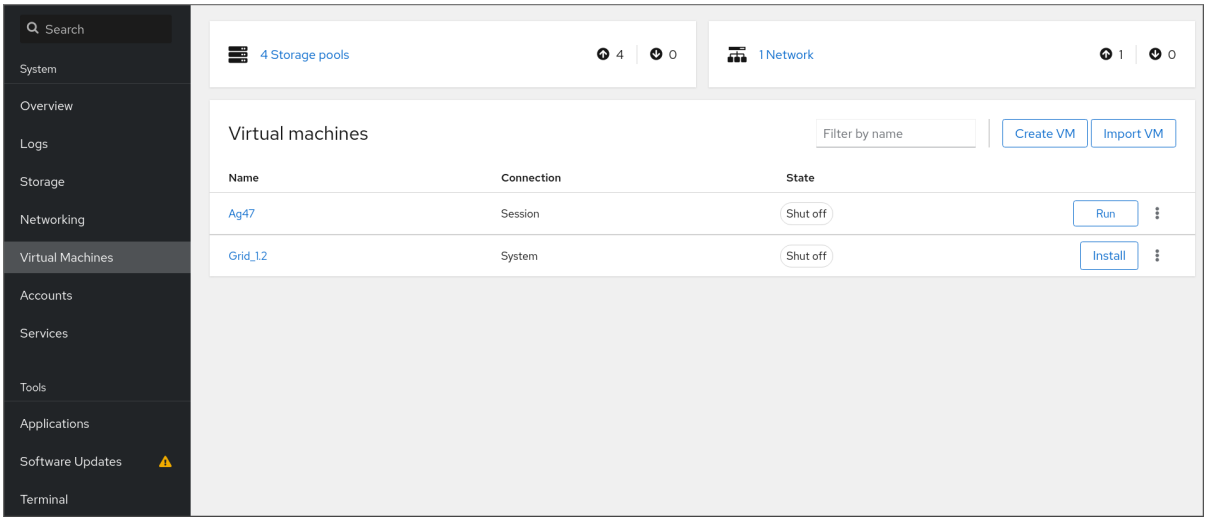
流程

- 安装 **cockpit-machines** 插件。

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

验证

1. 访问 Web 控制台，例如在浏览器中输入 **https://localhost:9090** 地址。
2. 登录。
3. 如果安装成功，则 **Virtual Machines** 会显示在 web 控制台侧菜单中。



其他资源

- 使用 RHEL 9 web 控制台管理系统

8.3. 使用 WEB 控制台重命名虚拟机

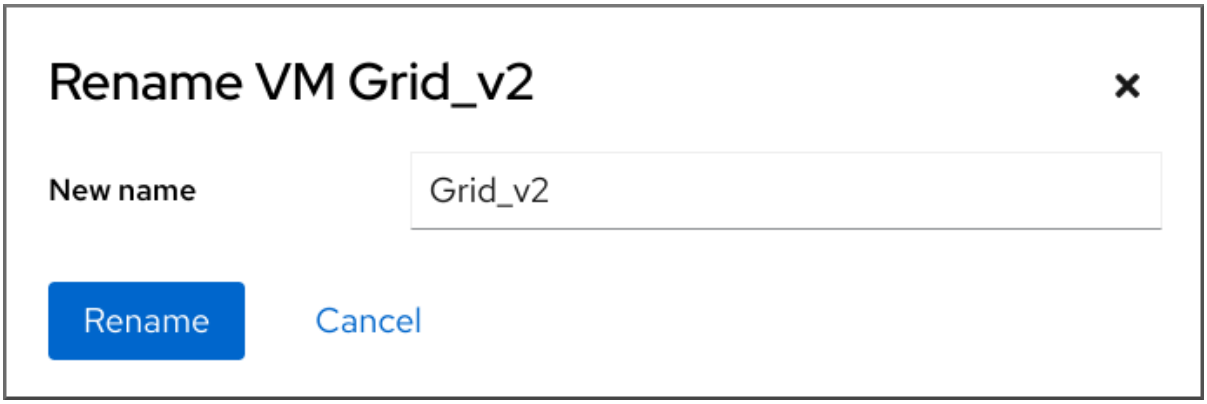
您可能需要重命名现有虚拟机(VM)以避免命名冲突，或者根据您的用例分配一个新的唯一名称。要重命名虚拟机，您可以使用 RHEL web 控制台。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。
- 虚拟机已关闭。

流程

1. 在 **Virtual Machines** 界面中，点击您要重命名的虚拟机的菜单按钮 ⋮。
   此时会出现一个控制各种虚拟机操作的下拉菜单。
2. 点击 **Rename**。
   出现 **Rename a VM**对话框。



3. 在 **New name** 字段中输入虚拟机的名称。
4. 点击 **Rename**。

验证

- 检查新虚拟机名称是否已出现在 **Virtual Machines** 界面中。

8.4. WEB 控制台中提供的虚拟机管理功能

通过使用 RHEL 9 web 控制台，您可以执行以下操作来管理系统上的虚拟机(VM)。

表 8.1. 您可以在 RHEL 9 web 控制台中执行的虚拟机管理任务

任务	详情请查看
创建虚拟机并将其安装到客户端操作系统	使用 web 控制台创建虚拟机并安装客户机操作系统
删除虚拟机	使用 web 控制台删除虚拟机
启动、关闭和重启虚拟机	使用 web 控制台启动虚拟机，并 使用 web 控制台关闭和重启虚拟机
使用各种控制台连接到虚拟机并与虚拟机交互	使用 web 控制台与虚拟机进行交互
查看有关虚拟机的各种信息	使用 web 控制台查看虚拟机信息
调整分配给虚拟机的主机内存	使用 web 控制台添加和删除虚拟机内存
管理虚拟机的网络连接	使用 web 控制台管理虚拟机网络接口
管理主机上可用的虚拟机存储，并将虚拟磁盘附加到虚拟机	使用 web 控制台管理虚拟机的存储
配置虚拟机的虚拟 CPU 设置	使用 Web 控制台管理虚拟 CPU
实时迁移虚拟机	使用 web 控制台实时迁移虚拟机
管理主机设备	使用 Web 控制台管理主机设备

任务	详情请查看
管理虚拟光驱	管理虚拟光驱
附加 watchdog 设备	使用 web 控制台将 watchdog 设备附加到虚拟机

第 9 章 查看有关虚拟机的信息

当您需要在 RHEL 9 上调整或排除虚拟化部署的任何方面时，您需要执行的第一个步骤通常是查看有关虚拟机当前状态和配置的信息。要做到这一点，您可以使用 命令行界面 或 Web 控制台。您还可以查看虚拟机 XML 配置中的信息。

9.1. 使用命令行界面查看虚拟机信息

要检索主机上虚拟机（VM）的信息，请使用以下一个或多个命令。

流程

- 获取主机上的虚拟机列表：

```
# virsh list --all
 Id   Name            State
--------------------------------
 1    testguest1       running
 -    testguest2       shut off
 -    testguest3       shut off
 -    testguest4       shut off
```

- 要获取有关特定虚拟机的基本信息：

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

- 要获得特定虚拟机的完整 XML 配置：

```
# virsh dumpxml testguest2

<domain type='kvm' id='1'>
  <name>testguest2</name>
  <uuid>a973434f-2f6e-4ěša-8949-76a7a98569e1</uuid>
  <metadata>
[...]
```

- 有关虚拟机磁盘和其它块设备的详情：

```
# virsh domblklist testguest3
```

```
 Target   Source
------------------------------------------------------------
vda      /var/lib/libvirt/images/testguest3.qcow2
sda      -
sdb      /home/username/Downloads/virt-p2v-1.36.10-1.el7.iso
```

- 要获取有关虚拟机文件系统及其挂载点的信息：

```
# virsh domfsinfo testguest3
Mountpoint   Name   Type   Target
-----------------------------------
/            dm-0   xfs
/boot        vda1   xfs
```

- 要获取有关特定虚拟机 vCPU 的详细信息：

```
# virsh vcpuinfo testguest4
VCPU:           0
CPU:            3
State:          running
CPU time:       103.1s
CPU Affinity:   yyyy

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

要在虚拟机中配置和优化 vCPU，请参阅 优化虚拟机 CPU 性能。

- 列出主机上的所有虚拟网络接口：

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

有关特定接口的详情：

```
# virsh net-info default
Name:           default
UUID:           c699f9f6-9202-4ca8-91d0-6b8cb9024116
Active:         yes
Persistent:     yes
Autostart:      yes
Bridge:         virbr0
```

有关网络接口、虚拟机网络和配置它们的说明，请参阅配置虚拟机网络连接。

9.2. 使用 WEB 控制台查看虚拟机信息

通过使用 RHEL 9 web 控制台，您可以查看 web 控制台会话可以访问的所有 虚拟机 和 存储池 的信息。

您可以查看 web 控制台会话连接到的所选虚拟机的信息。这包括其 磁盘、虚拟网络接口和资源使用情况的信息。

9.2.1. 在 web 控制台中查看虚拟化概述

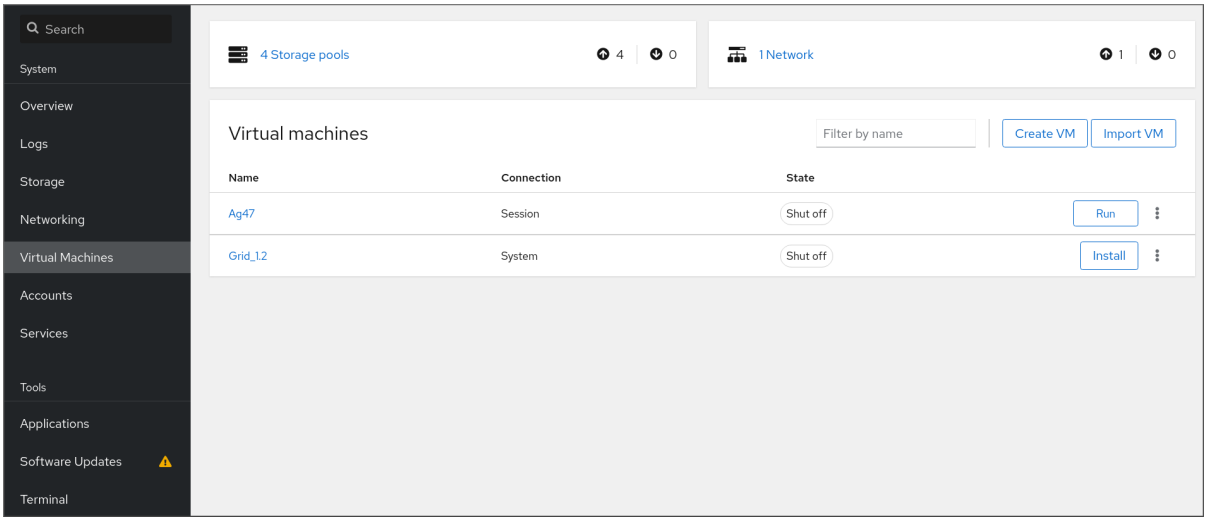
通过使用 web 控制台，您可以访问虚拟化概述，其包含有关可用虚拟机(VM)、存储池和网络的总结信息。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

- 在 web 控制台侧菜单中点击 **Virtual Machines**。
  此时将显示一个对话框，其中包含有关 Web 控制台所连接的可用存储池、可用网络和虚拟机的信息。



该信息包括：

- **存储池** - 可以通过 Web 控制台及其状态访问的活动的或非活动的存储池数量。
- **网络** - 可以通过 Web 控制台及其状态访问的活动的或非活动的网络数量。
- **Name** - 虚拟机的名称。
- **Connection** - libvirt 连接、系统或者会话的类型。
- **State** - 虚拟机的状态。

其它资源

- 使用 web 控制台查看虚拟机信息

9.2.2. 使用 Web 控制台查看存储池信息

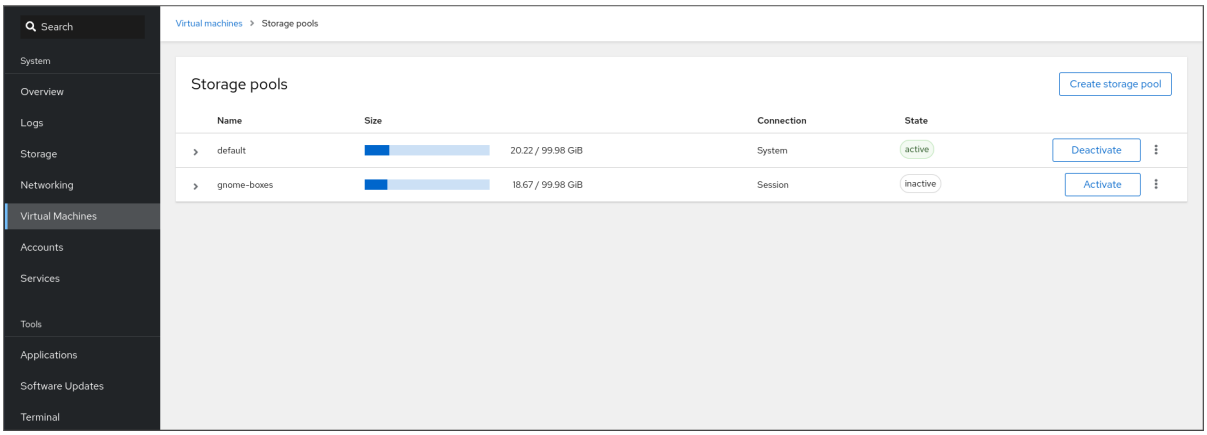
通过使用 Web 控制台，您可以查看系统上可用存储池的详细信息。存储池可用于为您的虚拟机创建磁盘映像。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

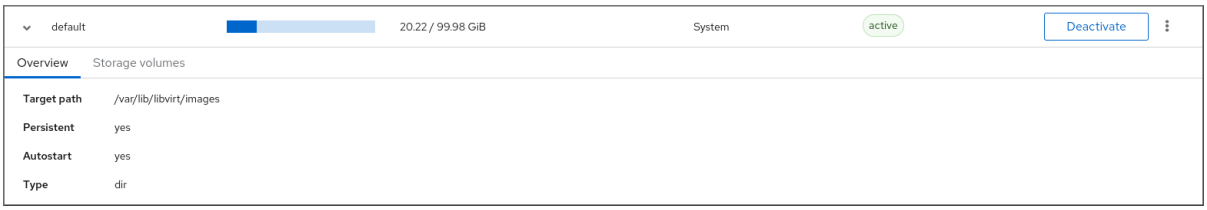
1. 点 **Virtual Machines** 接口顶部的 **Storage Pools**。
此时会出现**存储池**窗口，显示配置的存储池列表。



该信息包括：

- **Name** - 存储池的名称。
- **Size** - 当前分配和存储池的总容量。
- **connection** - 用于访问存储池的连接。
- **State** - 存储池的状态。

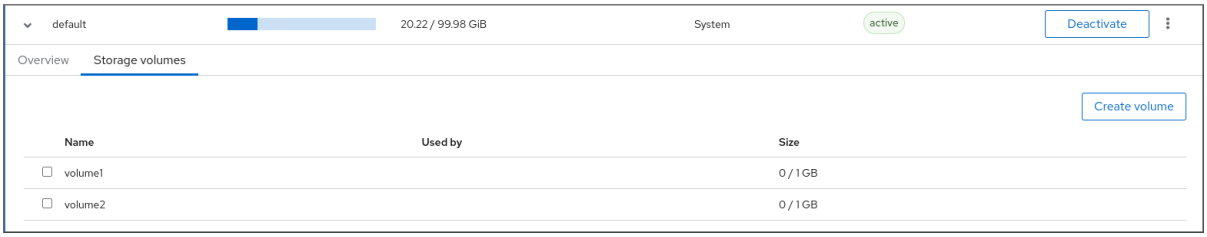
2. 点击您要查看其信息的存储池旁的箭头。
行会展开，以显示概述窗格，其中含有所选存储池的详细信息。



该信息包括：

- **Target path** - 存储池的位置。
- **Persistent** - 表示存储池是否具有持久性配置。
- **Autostart** - 表示在系统启动时存储池是否自动启动。
- **类型** - 存储池的类型。

3. 要查看与存储池相关联的存储卷的列表，请单击 **Storage Volumes**。
这时将出现存储卷窗格，显示已配置的存储卷的列表。



该信息包括：

- **Name** - 存储卷的名称。
- **Used by** - 当前使用存储卷的虚拟机。
- **Size** - 卷的大小。

其它资源

- 使用 web 控制台查看虚拟机信息

9.2.3. 在 web 控制台中查看基本虚拟机信息

通过使用 web 控制台，您可以查看有关所选虚拟机(VM)的基本信息，如分配的资源或 hypervisor 详情。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 在 web 控制台侧菜单中点击 **Virtual Machines**。
2. 点击您要查看其信息的虚拟机。
   此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟机的图形界面的控制台部分。

   概述部分包括以下常规虚拟机详情：

   - **State** - VM 的状态，运行或关闭。
   - **Memory** - 分配给虚拟机的内存量。
   - **CPU** - 为虚拟机配置的虚拟 CPU 的数量和类型。
   - **Boot Order** - 为虚拟机配置的引导顺序。
   - **Autostart** - 是否为虚拟机启用自动启动。

   该信息还包括以下管理程序详情：

   - **Emulated Machine** - 虚拟机模拟的机器类型。
   - **Firmware** - 虚拟机的固件。

其它资源

- 使用 web 控制台查看虚拟机信息
- 使用 Web 控制台管理虚拟 CPU

9.2.4. 在 web 控制台中查看虚拟机资源使用情况

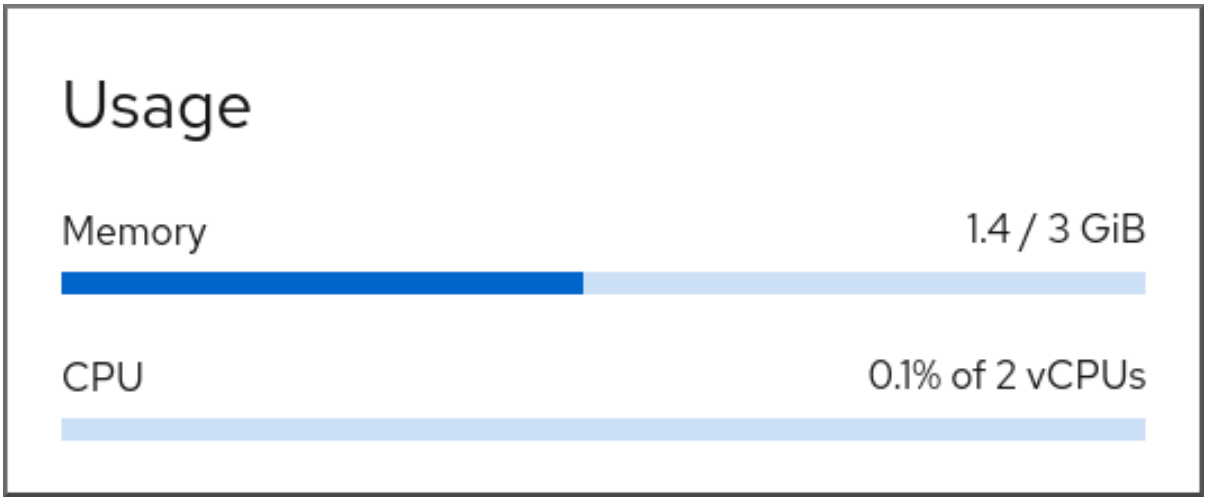
通过使用 web 控制台，您可以查看所选虚拟机(VM)的内存和虚拟 CPU 使用率。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 在 **Virtual Machines** 界面中，点击您要查看其信息的虚拟机。
   此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟机的图形界面的控制台部分。
2. 滚动到 **使用情况** 。
   使用情况部分显示有关虚拟机内存和虚拟 CPU 使用情况的信息。



其它资源

- 使用 web 控制台查看虚拟机信息

9.2.5. 在 web 控制台中查看虚拟机磁盘信息

通过使用 web 控制台，您可以查看分配给所选虚拟机(VM)的磁盘的详细信息。

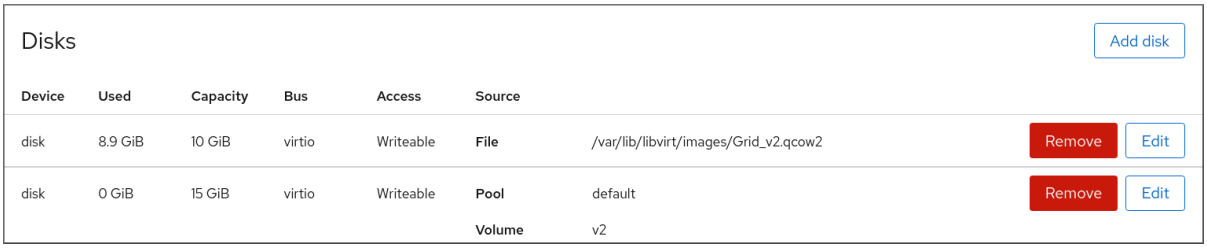
先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 点击您要查看其信息的虚拟机。
   此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟机的图形界面的控制台部分。

2. 滚动到 **磁盘** 。
Disks 部分显示有关分配给虚拟机的磁盘的信息，以及用于 **Add** 或 **Edit** 磁盘的选项。



该信息包括：

- **Device** - 磁盘的设备类型。
- **Used** - 当前分配的磁盘数量。
- **Capacity** - 存储卷的最大大小。
- **Bus** - 模拟的磁盘设备的类型。
- **Access** - 磁盘是否是 **可写** 或 **只读**。对于 **raw** 磁盘，您还可以将访问权限设为 **可写和共享**。
- **Source** - 磁盘设备或者文件。

其它资源

- 使用 web 控制台查看虚拟机信息

9.2.6. 在 web 控制台中查看和编辑虚拟网络接口信息

通过使用 RHEL 9 web 控制台，您可以在所选虚拟机(VM)上查看和修改虚拟网络接口：

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

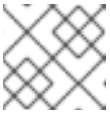
流程

1. 在 **Virtual Machines** 界面中，点击您要查看其信息的虚拟机。
此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟机的图形界面的控制台部分。

2. 滚动 **到网络接口** 。
网络接口部分显示为虚拟机配置的虚拟网络接口的信息，以及 **添加**、**删除**、**编辑** 或 **拔掉** 网络接口的选项。

该信息包括：

- **类型** - 虚拟机的网络接口类型。类型包括虚拟网络、网桥到 LAN 以及直接附加。

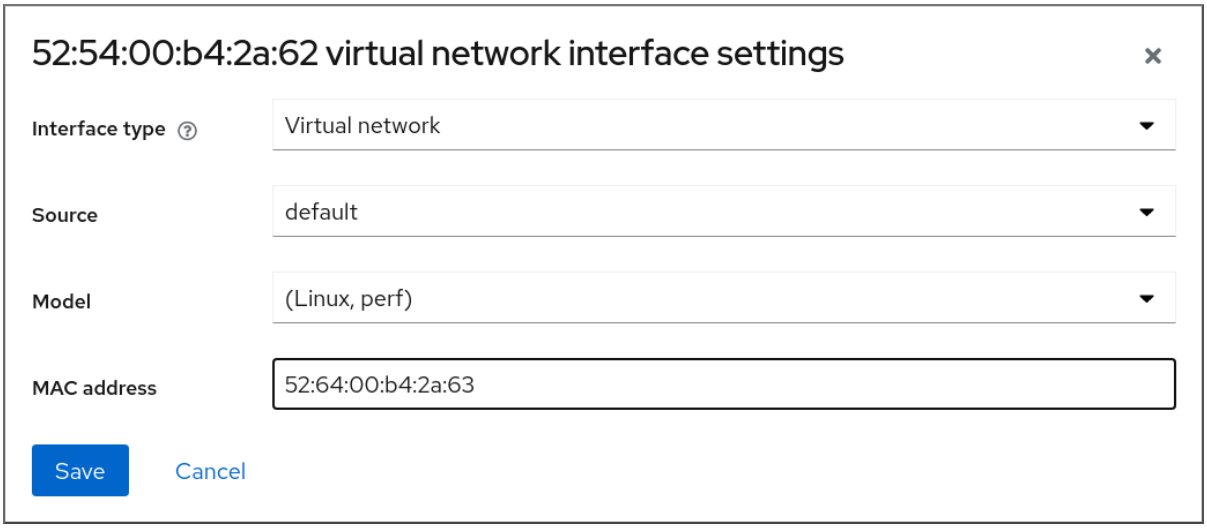


注意

RHEL 9 及更高版本不支持通用以太网连接。

- **型号类型** - 虚拟网络接口的型号。
- **MAC 地址** - 虚拟网络接口的 MAC 地址。
- **IP 地址** - 虚拟网络接口的 IP 地址。
- **Source** - 网络接口源。这取决于网络类型。
- **State** - 虚拟网络接口的状态。

3. 要编辑虚拟网络接口设置，请点 **Edit**。此时会打开「虚拟网络接口设置」对话框。



4. 更改接口类型、源、型号或 MAC 地址。
5. 点 **Save**。已修改网络接口。

注意

对虚拟网络接口设置的更改仅在重启虚拟机后生效。

此外，MAC 地址只能在虚拟机关闭时修改。

其它资源

- 使用 web 控制台查看虚拟机信息

9.3. 虚拟机 XML 配置示例

虚拟机的 XML 配置（也称为 *域 XML* ）决定虚拟机的设置和组件。下表显示了虚拟机（VM）的 XML 配置示例并解释了其内容。

要获取虚拟机的 XML 配置，您可以使用 **virsh dumpxml** 命令，后跟虚拟机的名称。

```
# virsh dumpxml testguest1
```

表 9.1. XML 配置示例

域 XML 部分	描述
<domain type='kvm'> <name>Testguest1</name> <uuid>ec6fbaa1-3eb4-49da-bf61-bb02fbec4967</uuid> <memory unit='KiB'>1048576</memory> <currentMemory unit='KiB'>1048576</currentMemory>	这是一个名为 *Testguest1* 的 KVM 虚拟机，内存为 1024 MiB。
<vcpu placement='static'>1</vcpu>	虚拟机被分配为单个虚拟 CPU（vCPU）。 有关配置 vCPU 的详情，请参考 优化虚拟机 CPU 性能。
<os> <type arch='x86_64' machine='pc-q35-rhel9.0.0'>hvm</type> <boot dev='hd'/> </os>	机器构架被设置为 AMD64 和 Intel 64 架构，并使用 Intel Q35 机器类型来决定功能兼容性。操作系统被设置为从硬盘引导。 有关使用安装的操作系统创建虚拟机的详情，请参考 使用 web 控制台创建虚拟机并安装客户机操作系统。
<features> <acpi/> <apic/> </features>	**acpi** 和 **apic** hypervisor 功能被禁用。
<cpu mode='host-model' check='partial'/>	功能 XML 中的主机 CPU 定义（可通过 **virsh domcapabilities** 可自动复制到虚拟机的 XML 配置中）。因此，当虚拟机引导时，**libvirt** 会选择类似主机 CPU 的 CPU 模型，然后尽可能向大约主机模型添加额外的功能。
<clock offset='utc'> <timer name='rtc' tickpolicy='catchup'/> <timer name='pit' tickpolicy='delay'/> <timer name='hpet' present='no'/> </clock>	VM 的虚拟硬件时钟使用 UTC 时区。另外，设置了三个不同的计时器以便与 QEMU 管理程序同步。
<on_poweroff>destroy</on_poweroff> <on_reboot>restart</on_reboot> <on_crash>destroy</on_crash>	当虚拟机关闭或其操作系统意外终止时，**libvirt** 会终止虚拟机并释放其所有分配的资源。虚拟机重启后，**libvirt** 会使用同样的配置重新启动它。

域 XML 部分	描述
<pm> <suspend-to-mem enabled='no'/> <suspend-to-disk enabled='no'/> </pm>	这个虚拟机禁用 S3 和 S4 ACPI 睡眠状态。
<devices> <emulator>/usr/libexec/qemu-kvm</emulator> <disk type='file' device='disk'> <driver name='qemu' type='qcow2'/> <source file='/var/lib/libvirt/images/Testguest.qcow2'/> <target dev='vda' bus='virtio'/> </disk> <disk type='file' device='cdrom'> <driver name='qemu' type='raw'/> <target dev='sdb' bus='sata'/> <readonly/> </disk>	虚拟机使用 **/usr/libexec/qemu-kvm** 二进制文件模拟，它连接了两个磁盘设备。 第一个磁盘是基于主机上存储的 **/var/lib/libvirt/images/Testguest.qcow2** 的虚拟硬盘，其逻辑设备名称设置为 **vda**。在 Windows 虚拟机中，建议您使用 **sata** 总线而不是 **virtio**。 第二个磁盘是虚拟化 CD-ROM，其逻辑设备名称被设置为 **sdb**。

域 XML 部分	描述
<controller type='usb' index='0' model='qemu-xhci' ports='15'/> <controller type='sata' index='0'/> <controller type='pci' index='0' model='pcie-root'/> <controller type='pci' index='1' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='1' port='0x10'/> </controller> <controller type='pci' index='2' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='2' port='0x11'/> </controller> <controller type='pci' index='3' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='3' port='0x12'/> </controller> <controller type='pci' index='4' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='4' port='0x13'/> </controller> <controller type='pci' index='5' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='5' port='0x14'/> </controller> <controller type='pci' index='6' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='6' port='0x15'/> </controller> <controller type='pci' index='7' model='pcie-root-port'> <model name='pcie-root-port'/> <target chassis='7' port='0x16'/> </controller> <controller type='virtio-serial' index='0'/>	VM 使用单个控制器来附加 USB 设备，而用于 PCI-Express（PCIe）设备的根控制器。此外，提供了 **virtio-serial** 控制器，它使虚拟机能够以各种方式与主机进行交互，如串行控制台。 有关虚拟设备的更多信息，请参阅虚拟设备的类型。
<interface type='network'> <mac address='52:54:00:65:29:21'/> <source network='default'/> <model type='virtio'/> </interface>	在虚拟机中设置了网络接口，它使用 **default** 虚拟网络和 **virtio** 网络设备模型。在 Windows 虚拟机中，建议您使用 **e1000e** 模型而不是 **virtio**。 有关配置网络接口的详情，请参考优化虚拟机网络性能。

域 XML 部分	描述
<serial type='pty'> <target type='isa-serial' port='0'> <model name='isa-serial'/> </target> </serial> <console type='pty'> <target type='serial' port='0'/> </console> <channel type='unix'> <target type='virtio' name='org.qemu.guest_agent.0'/> <address type='virtio-serial' controller='0' bus='0' port='1'/> </channel>	在虚拟机上设置了一个 **pty** 串行控制台，它可启用与主机的不便虚拟机通信。控制台使用端口 1 上的 **UNIX** 频道。这个设置是自动设置的，我们不推荐修改这些设置。 有关与虚拟机交互的更多信息，请参阅 使用 web 控制台与虚拟机进行交互。
<input type='tablet' bus='usb'> <address type='usb' bus='0' port='1'/> </input> <input type='mouse' bus='ps2'/> <input type='keyboard' bus='ps2'/>	虚拟机使用虚拟 **usb** 端口，该端口设定为接收表格输入，并设置了一个虚拟 **ps2** 端口以接收鼠标和键盘输入。这个设置是自动设置的，我们不推荐修改这些设置。
<graphics type='vnc' port='-1' autoport='yes' listen='127.0.0.1'> <listen type='address' address='127.0.0.1'/> </graphics>	虚拟机使用 **vnc** 协议渲染其图形输出。
<redirdev bus='usb' type='tcp'> <source mode='connect' host='localhost' service='4000'/> <protocol type='raw'/> </redirdev> <memballoon model='virtio'> <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/> </memballoon> </devices> </domain>	虚拟机使用 **tcp** re-director 远程附加 USB 设备，并启用内存气球功能。这个设置是自动设置的，我们不推荐修改这些设置。

第 10 章 保存和恢复虚拟机

要释放系统资源，您可以关闭该系统中运行的虚拟机（VM）。然而，当您再次需要虚拟机时，您必须引导客户端操作系统（OS）并重启应用程序，这可能需要大量时间。要减少这个停机时间并让虚拟机工作负载更早开始运行，您可以使用保存和恢复功能来完全避免操作系统关闭和引导序列。

本节提供有关保存虚拟机的信息，以及在没有完全引导虚拟机的情况下将虚拟机恢复到同一状态的信息。

10.1. 保存和恢复虚拟机的工作方式

保存虚拟机（VM）会将其内存和设备状态保存到主机的磁盘中，并立即停止虚拟机进程。您可以保存处于运行状态或暂停状态的虚拟机，在恢复后，虚拟机将返回到那个状态。

这个过程释放了主机系统中的 RAM 和 CPU 资源以交换磁盘空间，这样可提高主机系统的性能。当虚拟机被恢复时，因为不需要引导客户机操作系统，也避免使用较长的启动周期。

要保存虚拟机，您可以使用命令行界面（CLI）。具体说明请参阅 使用命令行界面保存虚拟机。

要恢复虚拟机，您可以使用 CLI 或 Web 控制台 GUI。

您还可以使用快照保存和恢复虚拟机的状态。如需更多信息，请参阅 使用快照保存和恢复虚拟机状态。

10.2. 使用命令行界面保存虚拟机

您可以将虚拟机(VM)及其当前状态保存到主机的磁盘上。这很有用，例如当您需要将主机的资源用于某些目的时。然后，保存的虚拟机可以快速恢复到其以前的运行状态。

要使用命令行保存虚拟机，请按照以下流程操作。

先决条件

- 确保您有足够的磁盘空间来保存虚拟机及其配置。请注意，虚拟机消耗的空间取决于分配给该虚拟机的 RAM 量。
- 确保虚拟机是持久的。
- **可选：** 根据需要备份虚拟机中的重要数据。

流程

- 使用 **virsh managedsave** 工具。
  例如，以下命令可停止 *demo-guest1* 虚拟机并保存其配置。

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

  保存的虚拟机文件默认位于 **/var/lib/libvirt/qemu/save** 目录中，即 **demo-guest1.save**。

  下次启动虚拟机时，它将自动从上述文件中恢复保存的状态。

验证

- 列出已启用了受管保存的虚拟机。在以下示例中，列为 *saved* 的虚拟机启用了受管保存。

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

列出具有受管保存镜像的虚拟机：

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

请注意，要列出处于关机状态的保存的虚拟机，您必须使用命令的 **--all** 或 **inactive** 选项。

故障排除

- 如果保存的虚拟机文件变得损坏或不可读，恢复虚拟机将启动标准虚拟机引导。

其它资源

- **virsh managedsave --help** 命令
- 使用命令行界面恢复保存的虚拟机
- 使用 Web 控制台恢复保存的虚拟机

10.3. 使用命令行界面启动虚拟机

您可以使用命令行界面(CLI)启动关闭的虚拟机(VM)或恢复保存的虚拟机。通过使用 CLI，您可以启动本地和远程虚拟机。

先决条件

- 已定义的一个不活跃地虚拟机。
- 虚拟机的名称。
- 对于远程虚拟机：
  - 虚拟机所在主机的 IP 地址。
  - 对主机的 root 访问权限。

流程

- 对于本地虚拟机，请使用 **virsh start** 工具。
  例如，以下命令启动 *demo-guest1* 虚拟机。

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

- 对于位于远程主机上的虚拟机，请使用 **virsh start** 工具以及与主机的 QEMU+SSH 连接。
  例如，以下命令在 192.0.2.1 主机上启动 *demo-guest1* 虚拟机。

```
# virsh -c qemu+ssh://root@192.0.2.1/system start demo-guest1

root@192.0.2.1's password:

Domain 'demo-guest1' started
```

其它资源

- **virsh start --help** 命令
- 设置对远程虚拟化主机的简单访问
- 当主机启动时自动启动虚拟机

10.4. 使用 WEB 控制台启动虚拟机

如果虚拟机(VM)处于 *shut off* 状态，您可以使用 RHEL 9 web 控制台启动它。您还可以将虚拟机配置为在主机启动时自动启动。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。
- 已定义的一个不活跃地虚拟机。
- 虚拟机的名称。

流程

1. 在 **Virtual Machines** 界面中，点击您要启动的虚拟机。
   此时将打开一个新页面，其中包含有关所选虚拟机的详细信息，以及关闭和删除虚拟机的控制。
2. 点 **Run**。
   虚拟机启动，您可以连接到其控制台或图形输出。
3. **可选：** 要将虚拟机配置为在主机启动时自动启动，请切换 **Overview** 部分中的 **Autostart** 复选框。
   如果使用不由 libvirt 管理的网络接口，您还必须对 systemd 配置进行额外的更改。否则，受影响的虚拟机可能无法启动，请参阅 当主机启动时自动启动虚拟机。

其它资源

- 在 web 控制台中关闭虚拟机
- 使用 web 控制台重启虚拟机

第 11 章 克隆虚拟机

要使用特定属性集合快速创建新虚拟机，您可以*克隆*现有虚拟机。

克隆会创建一个使用其自身磁盘镜像保存存储的新虚拟机，但大多数克隆的配置和源虚拟机的数据都是一样的。这样，可以针对某个特定任务准备优化多个虚拟机，而无需单独优化每个虚拟机。

11.1. 克隆虚拟机的方式

克隆虚拟机会复制源虚拟机及其磁盘镜像的 XML 配置，并对配置进行修改以确保新虚拟机的唯一性。这包括更改虚拟机的名称，并确保它使用磁盘镜像克隆。存储在克隆的虚拟磁盘上的数据与源虚拟机是一致的。

这个过程比创建新虚拟机要快，并使用客户端操作系统安装它，并可用于快速生成带有特定配置和内容的虚拟机。

如果您计划为虚拟机创建多个克隆，首先请创建一个不包含以下内容的虚拟机*模板*：

- 唯一设置，如持久性网络 MAC 配置，这可阻止克隆正常工作。
- 敏感数据，如 SSH 密钥和密码文件。

具体说明请参阅 创建虚拟机模板。

其它资源

- 使用命令行界面克隆虚拟机
- 使用 web 控制台克隆虚拟机

11.2. 创建虚拟机模板

要创建可正常工作的多个虚拟机(VM)克隆，您可以对删除源虚拟机的唯一信息和配置，如 SSH 密钥或持久性网络 MAC 配置。这将创建一个虚拟机 *模板*，您可以使用它来轻松且安全地创建虚拟机克隆。

您可以 使用 **virt-sysprep** 工具 创建虚拟机模板或根据您的要求 手动创建它们。

11.2.1. 使用 virt-sysprep 创建虚拟机模板

要从现有虚拟机(VM)创建克隆模板，您可以使用 **virt-sysprep** 工具。这会删除某些可能导致克隆工作不正常的配置，如特定的网络设置或系统注册元数据。因此，**virt-sysprep** 可以更高效地创建虚拟机的克隆，并确保克隆可以更加可靠地工作。

先决条件

- 在主机上安装了包含 **virt-sysprep** 工具的 **guestfs-tools** 软件包：

```
# dnf install guestfs-tools
```

- 充当模板的源虚拟机已关闭。
- 您知道源虚拟机的磁盘镜像所在的位置，并且您是虚拟机磁盘镜像文件的所有者。
  请注意，在 libvirt 的 系统连接 中创建的虚拟机的磁盘镜像位于 **/var/lib/libvirt/images** 目录中，默认由 root 用户拥有：

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

- **可选：** 源虚拟机磁盘上的所有重要数据都已备份。如果要保留源虚拟机完好无损，请首先 克隆它，并将克隆转换为模板。

流程

1. 确保您已作为虚拟机磁盘镜像的所有者登录：

```
# whoami
root
```

2. **可选：** 复制虚拟机的磁盘镜像。

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

这用于之后验证虚拟机是否已成功转换为模板。

3. 使用以下命令，将 */var/lib/libvirt/images/a-really-important-vm.qcow2* 替换为源虚拟机磁盘镜像的路径。

```
# virt-sysprep -a /var/lib/libvirt/images/a-really-important-vm.qcow2
[   0.0] Examining the guest ...
[   7.3] Performing "abrt-data" ...
[   7.3] Performing "backup-files" ...
[   9.6] Performing "bash-history" ...
[   9.6] Performing "blkid-tab" ...
[...]
```

验证

- 要确认进程成功，请将修改的磁盘镜像与原始镜像进行比较。以下示例显示了成功创建模板：

```
# virt-diff -a /var/lib/libvirt/images/a-really-important-vm-orig.qcow2 -A /var/lib/libvirt/images/a-really-important-vm.qcow2
- - 0644       1001 /etc/group-
- - 0000        797 /etc/gshadow-
= - 0444         33 /etc/machine-id
[...]
- - 0600        409 /home/username/.bash_history
- d 0700          6 /home/username/.ssh
- - 0600        868 /root/.bash_history
[...]
```

其它资源

- **virt-sysprep** 手册页中的 OPERATIONS 部分

- 使用命令行界面克隆虚拟机

11.2.2. 手动创建虚拟机模板

要从现有虚拟机(VM)创建模板，您可以手动重置或取消配置客户虚拟机来为克隆做好准备。

先决条件

- 确保您知道源虚拟机的磁盘镜像的位置，并且是虚拟机磁盘镜像文件的所有者。
  请注意，在 libvirt 的 系统连接 中创建的虚拟机的磁盘镜像默认位于 **/var/lib/libvirt/images** 目录中，并由 root 用户拥有：

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

- 确保虚拟机已关闭。
- **可选：** 虚拟机磁盘上的所有重要数据都已备份。如果要保留源虚拟机，请首先克隆它，并编辑克隆以创建模板。

流程

1. 配置虚拟机以进行克隆：
   a. 在克隆上安装任何所需的软件。
   b. 为操作系统配置任何非唯一的设置。
   c. 配置任何非唯一的应用设置。
2. 删除网络配置：
   a. 使用以下命令删除任何持久性 udev 规则：

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

注意

如果没有删除 udev 规则，第一个 NIC 的名称可能是 **eth1** 而不是 **eth0**。

   b. 从 **/etc/NetworkManager/system-connections/** 目录中的 **NMConnection** 文件中删除唯一信息。
      i. 删除 MAC 地址、IP 地址、DNS、网关以及任何其他 **唯一** 信息或非必要设置。

```
*ID=ExampleNetwork
BOOTPROTO="dhcp"
HWADDR="AA:BB:CC:DD:EE:FF"                <- REMOVE
NM_CONTROLLED="yes"
```

```
ONBOOT="yes"
TYPE="Ethernet"
UUID="954bd22c-f96c-4b59-9445-b39dd86ac8ab" <- REMOVE
```

ii. 从 **/etc/hosts** 和 **/etc/resolv.conf** 文件中删除类似的 **唯一** 信息和非必要设置。

3. 删除注册详情：

- 对于在 Red Hat Network (RHN) 上注册的虚拟机：

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

- 对于使用 Red Hat Subscription Manager (RHSM) 注册的虚拟机：
  - 如果您不打算使用原始虚拟机：

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

  - 如果您打算使用原始虚拟机：

```
# subscription-manager clean
```

注意

原始 RHSM 配置文件保留您 ID 代码的门户中。克隆后，使用以下命令重新激活虚拟机上的 RHSM 注册：

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

4. 删除其他唯一的详情：

a. 删除 SSH 公钥和私钥对：

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

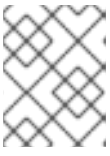
b. 删除 LVM 设备的配置：

```
# rm /etc/lvm/devices/system.devices
```

c. 如果在多台计算机上运行，删除任何其他特定于应用程序的标识符或配置可能会导致冲突。

5. 删除 **gnome-initial-setup-done** 文件，将虚拟机配置为在下次引导时运行配置向导：

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



注意

在下次引导时运行的向导依赖于已从虚拟机中删除的配置。另外，在克隆的第一次引导时，会建议您更改主机名。

11.3. 使用命令行界面克隆虚拟机

为了进行测试，要创建具有特定属性集的新虚拟机，您可以使用 CLI 克隆现有虚拟机。

先决条件

- 源虚拟机被关闭。
- 确保有足够的磁盘空间来存储克隆的磁盘镜像。
- **可选：** 在创建多个虚拟机克隆时，从源虚拟机中删除唯一数据和设置，以确保克隆的虚拟机正常工作。具体说明请参阅 创建虚拟机模板。

流程

- 使用 **virt-clone** 工具以及适合您环境和用例的选项。
  使用案例示例
  - 以下命令克隆一个名为 **example-VM-1** 的本地虚拟机，并创建 **example-VM-1-clone** 虚拟机。它还会在与原始虚拟机的磁盘镜像相同的位置创建并分配 **example-VM-1-clone.qcow2** 磁盘镜像，并使用相同的数据：

    ```
    # virt-clone --original example-VM-1 --auto-clone
    Allocating 'example-VM-1-clone.qcow2'                    | 50.0 GB  00:05:37

    Clone 'example-VM-1-clone' created successfully.
    ```

  - 以下命令克隆一个名为 **example-VM-2** 的虚拟机，并创建一个名为 **example-VM-3** 的本地虚拟机，它只使用 **example-VM-2** 的多个磁盘中的两个磁盘：

    ```
    # virt-clone --original example-VM-2 --name example-VM-3 --file
    /var/lib/libvirt/images/disk-1-example-VM-2.qcow2 --file /var/lib/libvirt/images/disk-2-
    example-VM-2.qcow2
    Allocating 'disk-1-example-VM-2-clone.qcow2'                    | 78.0 GB  00:05:37
    Allocating 'disk-2-example-VM-2-clone.qcow2'                    | 80.0 GB  00:05:37

    Clone 'example-VM-3' created successfully.
    ```

  - 要将虚拟机克隆到其他主机，请迁移虚拟机而无需在本地主机上取消它。例如，以下命令将之前创建的 **example-VM-3** 虚拟机克隆到 **192.0.2.1** 远程系统，包括其本地磁盘。请注意，您需要 root 权限来对 **192.0.2.1** 运行这些命令：

    ```
    # virsh migrate --offline --persistent example-VM-3 qemu+ssh://root@192.0.2.1/system
    root@192.0.2.1's password:

    # scp /var/lib/libvirt/images/<disk-1-example-VM-2-clone>.qcow2
    root@192.0.2.1/<user@remote_host.com>://var/lib/libvirt/images/

    # scp /var/lib/libvirt/images/<disk-2-example-VM-2-clone>.qcow2
    root@192.0.2.1/<user@remote_host.com>://var/lib/libvirt/images/
    ```

验证

1. 验证虚拟机是否已成功克隆，且正在正常工作：

a. 确认克隆已添加到主机上的虚拟机列表中：

```
# virsh list --all
Id   Name                State
----------------------------------------
-    example-VM-1        shut off
-    example-VM-1-clone  shut off
```

b. 启动克隆并观察它是否启动：

```
# virsh start example-VM-1-clone
Domain 'example-VM-1-clone' started
```

其它资源

- **virt-clone (1)** 手册页
- 迁移虚拟机

11.4. 使用 WEB 控制台克隆虚拟机

要创建具有特定属性集的新虚拟机，您可以克隆之前使用 web 控制台配置的虚拟机。

注意

克隆虚拟机也会克隆与该虚拟机关联的磁盘。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。
- 确保要克隆的虚拟机已关闭。

流程

1. 在 web 控制台的虚拟机界面中，单击您要克隆的虚拟机的菜单按钮 ⋮。
   此时会出现一个下拉菜单，控制各种虚拟机操作。
2. 单击 **Clone**。
   此时将显示一个创建克隆虚拟机对话框。

3. **可选：** 输入虚拟机克隆的新名称。

4. 单击 **Clone**。
基于源虚拟机的新虚拟机被创建。

验证

- 确认克隆的虚拟机是否出现在主机上可用的虚拟机列表中。

第 12 章 迁移虚拟机

如果虚拟机所在的当前主机变得不适合或者无法再使用，或者要重新分发托管工作负载，您可以将该虚拟机迁移到另一个 KVM 主机中。

12.1. 迁移虚拟机的工作方式

虚拟机迁移的基本部分是将虚拟机的 XML 配置复制到不同的主机机器中。如果没有关闭迁移的虚拟机，迁移还会将虚拟机内存和任何虚拟设备的状态传送到目标主机机器中。要使虚拟机在目标主机上正常工作，虚拟机的磁盘镜像必须仍可用。

默认情况下，迁移的虚拟机在目标主机上是临时的，虚拟机在源主机上仍然被定义。

您可以使用 *live* 或 *non-live* 迁移来迁移正在运行的虚拟机。要迁移关闭虚拟机，必须使用*离线*迁移。详情请查看下表。

表 12.1. VM 迁移类型

迁移类型	描述	使用案例	存储要求
实时迁移	VM 将继续在源主机中运行，而 KVM 会将虚拟机的内存页面传送到目标主机。当迁移接近完成后，KVM 会非常简单地挂起虚拟机，并在目标主机上恢复它。	对于需要一直保持运行的虚拟机，这个方法非常有用。但是，如果虚拟机修改内存页面的速度比 KVM 可以传输它们的速度更快，比如 I/O 负载较重的虚拟机，则不能进行实时迁移，这需要使用*非实时迁移*。	VM 的磁盘镜像必须位于共享网络中，同时可访问源主机和目标主机。
非实时迁移	挂起虚拟机，将其配置及其内存复制到目标主机，并恢复虚拟机。	这个迁移方式需要虚拟机停机，但通常比实时迁移更可靠。推荐用于内存负载过重的虚拟机。	VM 的磁盘镜像必须位于共享网络中，同时可访问源主机和目标主机。
离线迁移	将虚拟机的配置移到目标主机	建议用于关闭虚拟机，以及关闭虚拟机不会影响您的工作负载的情况。	VM 的磁盘镜像不必在共享网络中可用，并可手动复制或移动到目标主机。

您还可以将 *实时迁移* 与 *非实时迁移* 相组合。当实时迁移一个使用非常多的 vCPU 或大量内存的虚拟机（这会阻止迁移完成）时，建议您这样做。在这种情况下，您可以挂起源虚拟机。这可以防止生成额外的脏内存页，从而大大提高迁移完成的可能性。基于迁移期间客户机工作负载和静态页数，此类 *混合* 迁移可能会导致停机时间明显低于非实时迁移。

其它资源

- 迁移虚拟机的好处
- 将虚拟机磁盘镜像与其他主机共享

12.2. 迁移虚拟机的好处

迁移虚拟机对以下情况非常有用：

负载均衡

如果主机超载或者另一台主机使用不足，则可将虚拟机移动到使用率较低的主机中。

硬件独立

当您需要升级、添加或删除主机中的硬件设备时，您可以安全地将虚拟机重新定位到其他主机。这意味着，在改进硬件时虚拟机不需要停机。

节能

虚拟机可重新分发到其他主机，因此可关闭未载入的主机系统以便在低用量时节约能源并降低成本。

地理迁移

可将虚拟机移动到另一个物理位置，以减少延迟，或者因为其他原因需要。

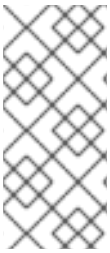
12.3. 迁移虚拟机的限制

在 RHEL 9 中迁移虚拟机前，请确定您了解迁移的限制。

- 将虚拟机从或迁移到 **libvirt** 的会话连接 是不可靠的，因此不建议这样做。
- 使用以下功能和配置的虚拟机在迁移时将无法正常工作，或者迁移失败。这些特性包括：
  - 设备透传
  - SR-IOV 设备分配
  - 介质设备，如 vGPU
- 使用非统一内存访问(NUMA)固定的主机之间的迁移只有在主机有类似的拓扑时才能正常工作。但是，迁移可能会对运行工作负载的性能造成负面影响。
- 源虚拟机和目标虚拟机上模拟的 CPU 必须相同，否则迁移可能会失败。虚拟机之间在以下 CPU 相关区域的任何区别都可能会引起迁移问题：
  - CPU 型号
    - 不支持在 Intel 64 主机和 AMD64 主机之间迁移，即使它们共享 x86-64 指令集。
    - 有关确保在迁移到具有不同 CPU 模型的主机后虚拟机可以正常工作的步骤，请参阅验证虚拟机迁移的主机 CPU 兼容性。
  - 固件设置
  - 微码版本
  - BIOS 版本
  - BIOS 设置
  - QEMU 版本
  - 内核版本
- 在某些情况下，实时迁移使用超过 1 TB 内存的虚拟机可能不可靠。有关如何防止或修复此问题的说明，请参阅 虚拟机的实时迁移时间很长，且没有完成。

12.4. 验证虚拟机迁移的主机 CPU 兼容性

要使迁移的虚拟机 (VM) 在目标主机上正常工作，源和目标主机上的 CPU 必须兼容。要确保情况如此，请在开始迁移前计算一个常见 CPU 基准。



注意

本节中的说明使用了具有以下主机 CPU 的迁移示例：

- 源主机：Intel Core i7-8650U
- 目标主机：Intel Xeon CPU E5-2620 v2

先决条件

- 已在您的系统中安装并启用虚拟化。
- 您具有对源主机和迁移的目标主机的管理员访问权限。

流程

1. 在源主机上，获取其 CPU 功能并将其粘贴到新的 XML 文件中，如 **domCaps-CPUs.xml**。

```
# virsh domcapabilities | xmllint --xpath "//cpu/mode[@name='host-model']" - > domCaps-CPUs.xml
```

2. 在 XML 文件中，将 **<mode> </mode>** 标签替换为 **<cpu> </cpu>**。
3. **可选：** 验证 **domCaps-CPUs.xml** 文件的内容类似如下：

```
# cat domCaps-CPUs.xml

  <cpu>
      <model fallback="forbid">Skylake-Client-IBRS</model>
      <vendor>Intel</vendor>
      <feature policy="require" name="ss"/>
      <feature policy="require" name="vmx"/>
      <feature policy="require" name="pdcm"/>
      <feature policy="require" name="hypervisor"/>
      <feature policy="require" name="tsc_adjust"/>
      <feature policy="require" name="clflushopt"/>
      <feature policy="require" name="umip"/>
      <feature policy="require" name="md-clear"/>
      <feature policy="require" name="stibp"/>
      <feature policy="require" name="arch-capabilities"/>
      <feature policy="require" name="ssbd"/>
      <feature policy="require" name="xsaves"/>
      <feature policy="require" name="pdpe1gb"/>
      <feature policy="require" name="invtsc"/>
      <feature policy="require" name="ibpb"/>
      <feature policy="require" name="ibrs"/>
      <feature policy="require" name="amd-stibp"/>
      <feature policy="require" name="amd-ssbd"/>
      <feature policy="require" name="rsba"/>
      <feature policy="require" name="skip-l1dfl-vmentry"/>
```

```
        <feature policy="require" name="pschange-mc-no"/>
        <feature policy="disable" name="hle"/>
        <feature policy="disable" name="rtm"/>
    </cpu>
```

4. 在目标主机上，使用以下命令获取其 CPU 功能：

```
# virsh domcapabilities | xmllint --xpath "//cpu/mode[@name='host-model']" -

    <mode name="host-model" supported="yes">
        <model fallback="forbid">IvyBridge-IBRS</model>
        <vendor>Intel</vendor>
        <feature policy="require" name="ss"/>
        <feature policy="require" name="vmx"/>
        <feature policy="require" name="pdcm"/>
        <feature policy="require" name="pcid"/>
        <feature policy="require" name="hypervisor"/>
        <feature policy="require" name="arat"/>
        <feature policy="require" name="tsc_adjust"/>
        <feature policy="require" name="umip"/>
        <feature policy="require" name="md-clear"/>
        <feature policy="require" name="stibp"/>
        <feature policy="require" name="arch-capabilities"/>
        <feature policy="require" name="ssbd"/>
        <feature policy="require" name="xsaveopt"/>
        <feature policy="require" name="pdpe1gb"/>
        <feature policy="require" name="invtsc"/>
        <feature policy="require" name="ibpb"/>
        <feature policy="require" name="amd-ssbd"/>
        <feature policy="require" name="skip-l1dfl-vmentry"/>
        <feature policy="require" name="pschange-mc-no"/>
    </mode>
```

5. 将从目标主机获取的 CPU 功能添加到源主机上的 **domCaps-CPUs.xml** 文件。同样，将 **<mode> </mode>** 标签替换为 **<cpu> </cpu>** 并保存文件。

6. **可选：** 验证 XML 文件现在包含两个主机的 CPU 功能。

```
# cat domCaps-CPUs.xml

    <cpu>
        <model fallback="forbid">Skylake-Client-IBRS</model>
        <vendor>Intel</vendor>
        <feature policy="require" name="ss"/>
        <feature policy="require" name="vmx"/>
        <feature policy="require" name="pdcm"/>
        <feature policy="require" name="hypervisor"/>
        <feature policy="require" name="tsc_adjust"/>
        <feature policy="require" name="clflushopt"/>
        <feature policy="require" name="umip"/>
        <feature policy="require" name="md-clear"/>
        <feature policy="require" name="stibp"/>
        <feature policy="require" name="arch-capabilities"/>
        <feature policy="require" name="ssbd"/>
        <feature policy="require" name="xsaves"/>
```

```
        <feature policy="require" name="pdpe1gb"/>
        <feature policy="require" name="invtsc"/>
        <feature policy="require" name="ibpb"/>
        <feature policy="require" name="ibrs"/>
        <feature policy="require" name="amd-stibp"/>
        <feature policy="require" name="amd-ssbd"/>
        <feature policy="require" name="rsba"/>
        <feature policy="require" name="skip-l1dfl-vmentry"/>
        <feature policy="require" name="pschange-mc-no"/>
        <feature policy="disable" name="hle"/>
        <feature policy="disable" name="rtm"/>
    </cpu>
    <cpu>
        <model fallback="forbid">IvyBridge-IBRS</model>
        <vendor>Intel</vendor>
        <feature policy="require" name="ss"/>
        <feature policy="require" name="vmx"/>
        <feature policy="require" name="pdcm"/>
        <feature policy="require" name="pcid"/>
        <feature policy="require" name="hypervisor"/>
        <feature policy="require" name="arat"/>
        <feature policy="require" name="tsc_adjust"/>
        <feature policy="require" name="umip"/>
        <feature policy="require" name="md-clear"/>
        <feature policy="require" name="stibp"/>
        <feature policy="require" name="arch-capabilities"/>
        <feature policy="require" name="ssbd"/>
        <feature policy="require" name="xsaveopt"/>
        <feature policy="require" name="pdpe1gb"/>
        <feature policy="require" name="invtsc"/>
        <feature policy="require" name="ibpb"/>
        <feature policy="require" name="amd-ssbd"/>
        <feature policy="require" name="skip-l1dfl-vmentry"/>
        <feature policy="require" name="pschange-mc-no"/>
    </cpu>
```

7. 使用 XML 文件计算您要迁移的虚拟机的 CPU 功能基准。

```
# virsh hypervisor-cpu-baseline domCaps-CPUs.xml

  <cpu mode='custom' match='exact'>
    <model fallback='forbid'>IvyBridge-IBRS</model>
    <vendor>Intel</vendor>
    <feature policy='require' name='ss'/>
    <feature policy='require' name='vmx'/>
    <feature policy='require' name='pdcm'/>
    <feature policy='require' name='pcid'/>
    <feature policy='require' name='hypervisor'/>
    <feature policy='require' name='arat'/>
    <feature policy='require' name='tsc_adjust'/>
    <feature policy='require' name='umip'/>
    <feature policy='require' name='md-clear'/>
    <feature policy='require' name='stibp'/>
    <feature policy='require' name='arch-capabilities'/>
    <feature policy='require' name='ssbd'/>
    <feature policy='require' name='xsaveopt'/>
```

```
    <feature policy='require' name='pdpe1gb'/>
    <feature policy='require' name='invtsc'/>
    <feature policy='require' name='ibpb'/>
    <feature policy='require' name='amd-ssbd'/>
    <feature policy='require' name='skip-l1dfl-vmentry'/>
    <feature policy='require' name='pschange-mc-no'/>
  </cpu>
```

8. 打开您要迁移的虚拟机 XML 配置，并将 **<cpu>** 部分的内容替换为上一步中获取的设置。

```
# virsh edit VM-name
```

9. 如果虚拟机正在运行，请重新启动它。

```
# virsh reboot VM-name
```

后续步骤

- 将虚拟机磁盘镜像与其他主机共享
- 使用命令行界面迁移虚拟机
- 使用 web 控制台实时迁移虚拟机

12.5. 将虚拟机磁盘镜像与其他主机共享

要在 支持的 KVM 主机 间执行虚拟机实时迁移，需要共享虚拟机存储。以下流程提供了使用 NFS 协议将本地存储的虚拟机镜像与源主机和目标主机进行共享的说明。

先决条件

- 旨在迁移的虚拟机被关闭。
- **可选：** 一个主机系统可用于托管不是源或目标主机的存储，但源和目标主机都可以通过网络访问它。这是共享存储的最佳解决方案，并被红帽推荐。
- 请确定 NFS 文件锁定没有被使用，因为在 KVM 中不支持它。
- 在源主机和目标主机上安装并启用 NFS。查看
- 部署 NFS 服务器.

流程

1. 连接到提供共享存储的主机。在本例中，它是 **example-shared-storage** 主机：

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

2. 在保存磁盘镜像的源主机上创建一个目录，并与迁移主机共享：

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

3. 将虚拟机的磁盘镜像从源主机复制到新创建的目录。以下示例将虚拟机的磁盘镜像 **example-disk-1** 复制到 **example-shared-storage** 主机的 **/var/lib/libvirt/shared-images/** 目录中：

```
# scp /var/lib/libvirt/images/example-disk-1.qcow2 root@example-shared-storage:/var/lib/libvirt/shared-images/example-disk-1.qcow2
```

4. 在您要用于共享存储的主机上，将共享目录添加到 **/etc/exports** 文件中。以下示例将 **/var/lib/libvirt/shared-images** 目录与 **example-source-machine** 和 **example-destination-machine** 主机共享：

```
# /var/lib/libvirt/shared-images example-source-machine(rw,no_root_squash) example-destination-machine(rw,no\_root_squash)
```

5. 在源和目标主机上，将共享目录挂载到 **/var/lib/libvirt/images** 目录中：

```
# mount example-shared-storage:/var/lib/libvirt/shared-images /var/lib/libvirt/images
```

验证

- 在源主机上启动虚拟机，并观察它是否已成功启动。

其它资源

- 部署 NFS 服务器

12.6. 使用命令行界面迁移虚拟机

如果虚拟机所在的当前主机变得不适合或者无法再使用，或者要重新分发托管工作负载，您可以将该虚拟机迁移到另一个 KVM 主机中。以下流程为此类迁移的不同场景提供了说明和示例。

先决条件

- 源主机和目标主机都使用 KVM 管理程序。
- 源主机和目标主机可以通过网络相互访问。使用 **ping** 工具进行验证。
- 确保目标主机上打开了以下端口：
  - 使用 SSH 连接到目标主机需要端口 22。
  - 使用 TLS 连接到目标主机需要端口 16509。
  - 使用 TCP 连接到目标主机需要端口 16514。
  - QEMU 需要端口 49152-49215 来传输内存和磁盘迁移数据。
- 对于红帽支持的迁移，源主机和目标主机必须使用特定的操作系统和机器类型。要确保情况是这种情况，请查看支持的虚拟机迁移主机。
- 虚拟机必须与目标主机的 CPU 功能兼容。要确保情况如此，请参阅验证虚拟机迁移的主机 CPU 兼容性。

- 要迁移的虚拟机的磁盘镜像位于源主机和目标主机都可访问的单独的网络位置。这在离线迁移中是可选的，但在迁移运行的虚拟机时是必需的。
  有关设置这样的共享虚拟机存储的步骤，请参阅与其他主机共享虚拟机磁盘镜像。

- 迁移正在运行的虚拟机时，您的网络带宽必须高于虚拟机生成脏内存页面的速度。
  要在开始实时迁移之前获取虚拟机的脏页面率，请执行以下操作：

  - 监控虚拟机在短时间内的脏页面生成率。

    ```
    # virsh domdirtyrate-calc example-VM 30
    ```

  - 在监控完成后，获取其结果：

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

    在本例中，VM 每秒会生成 2 MB 的脏内存页面。如果您不暂停虚拟机或降低其工作负载，那么在小于或等于 2 MB/秒带宽的网络中尝试实时迁移这样的虚拟机会导致实时迁移不会进行。

    为确保实时迁移成功完成，红帽建议您的网络带宽要明显大于虚拟机的脏页面生成率。

- 当迁移公共网桥中现有虚拟机时，源和目标主机必须位于同一网络中。否则，迁移后 VM 网络将不会操作。

注意

calc_period 选项的值可能因工作负载和脏页率而异。您可以使用多个 **calc_period** 值进行试验，以确定与您环境中脏页率一致的最合适的周期。

- 在执行虚拟机迁移时，源主机上的 **virsh** 客户端可以使用多种协议之一连接到目标主机上的 libvirt 守护进程。以下流程中的示例使用 SSH 连接，但您可以选择不同的连接。

  - 如果您希望 libvirt 使用 SSH 连接，请确保启用 **virtqemud** 套接字并在目标主机上运行。

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

  - 如果您希望 libvirt 使用 TLS 连接，请确保启用 **virtproxyd-tls** 套接字并在目标主机上运行。

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

  - 如果您希望 libvirt 使用 TCP 连接，请确保 **virtproxyd-tcp** 套接字已经启用并在目标主机上运行。

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

步骤

1. 使用 **virsh migrate** 命令，以及适合您迁移要求的选项。

a. 以下命令使用 SSH 隧道将 **example-VM-1** 虚拟机从本地主机迁移到 **example-destination** 主机的系统连接。虚拟机在迁移过程中继续运行。

```
# virsh migrate --persistent --live example-VM-1 qemu+ssh://example-destination/system
```

b. 以下命令可让您对主机上运行的 **example-VM-2** 虚拟机的配置进行手动调整，然后将虚拟机迁移到 **example-destination** 主机。迁移的虚拟机将自动使用更新的配置。

```
# virsh dumpxml --migratable example-VM-2 > example-VM-2.xml
# vi example-VM-2.xml
# virsh migrate --live --persistent --xml example-VM-2.xml example-VM-2
qemu+ssh://example-destination/system
```

当目标主机需要使用不同路径访问共享虚拟机存储或配置特定于目标主机的功能时，这个过程很有用。

c. 以下命令挂起 **example-source** 主机上的 **example-VM-3** 虚拟机，将其迁移到 **example-destination** 主机，并指示它使用 **example-VM-3-alt.xml** 文件提供的调整后的 XML 配置。迁移完成后，**libvirt** 会在目标主机上恢复虚拟机。

```
# virsh migrate example-VM-3 qemu+ssh://example-source/system qemu+ssh://example-destination/system --xml example-VM-3-alt.xml
```

迁移后，虚拟机在源主机上处于关闭状态，并在关闭后删除迁移的副本。

d. 以下命令关闭 **example-source** 主机上的 **example-VM-4** 虚拟机，并将其配置移到 **example-destination** 主机。

```
# virsh migrate --offline --persistent --undefinesource example-VM-4
qemu+ssh://example-source/system qemu+ssh://example-destination/system
```

请注意，这种类型的迁移不需要将虚拟机的磁盘镜像移到共享存储中。但是，为了使虚拟机在目标主机上可用，您还需要迁移虚拟机的磁盘镜像。例如：

```
# scp root@example-source:/var/lib/libvirt/images/example-VM-4.qcow2 root@example-destination:/var/lib/libvirt/images/example-VM-4.qcow2
```

e. 以下命令将 *example-VM-5* 虚拟机迁移到 *example-destination* 主机，并使用多个并行连接，也称为多个文件描述符(multi-FD)迁移。借助多 FD 迁移，您可以使用迁移过程的所有可用网络带宽来加快迁移速度。

```
# virsh migrate --parallel --parallel-connections 4 <example-VM-5>
qemu+ssh://<example-destination>/system
```

这个示例使用 4 个多 FD 通道来迁移 *example-VM-5* 虚拟机。建议为可用网络带宽的每 10 Gbps 使用一个通道。默认值为 2 通道。

2. 等待迁移完成。这个过程可能需要一些时间，具体要看网络带宽、系统负载和虚拟机的大小。如果 **virsh migrate** 没有使用 **--verbose** 选项，CLI 不会显示任何进度指示器，除了错误。迁移进行时，您可以使用 **virsh domjobinfo** 工具显示迁移统计信息。

验证

- 在目标主机上，列出可用虚拟机以验证虚拟机是否已迁移：

```
# virsh list
Id    Name           State
----------------------------------
10    example-VM-1   running
```

如果迁移仍在运行，这个命令会将虚拟机状态列为 **paused**。

故障排除

- 在某些情况下，目标主机与迁移虚拟机 XML 配置的某些值不兼容，比如网络名称或 CPU 类型。因此，虚拟机将无法在目标主机上引导。要解决这些问题，您可以使用 **virsh edit** 命令更新有问题的值。在更新值后，您必须重启虚拟机以使更改生效。
- 如果实时迁移需要很长时间才能完成，这可能是因为虚拟机负载很重，且有太多的内存页面改变使得实时迁移不可能实现。要解决这个问题，请挂起虚拟机，将迁移改为非实时迁移。

```
# virsh suspend example-VM-1
```

其它资源

- **virsh migrate --help** 命令
- **virsh (1)** 手册页

12.7. 使用 WEB 控制台实时迁移虚拟机

如果要迁移正在执行要求其持续运行任务的虚拟机(VM)，您可以将该虚拟机迁移到另一个 KVM 主机上，而不需要关闭它。这也称为实时迁移。以下说明解释了如何使用 Web 控制台进行此操作。

警告

如果修改内存页面任务的速度比 KVM 可以传输页面的速度快（I/O 负载较重的任务），建议不要实时迁移虚拟机。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。
- 源和目标主机都在运行。
- 确保目标主机上打开了以下端口：
  - 使用 SSH 连接到目标主机需要端口 22。
  - 使用 TLS 连接到目标主机需要端口 16509。
  - 使用 TCP 连接到目标主机需要端口 16514。
  - QEMU 需要端口 49152-49215 来传输内存和磁盘迁移数据。

- 虚拟机必须与目标主机的 CPU 功能兼容。要确保情况如此，请参阅验证虚拟机迁移的主机 CPU 兼容性。
- 虚拟机磁盘镜像位于 共享存储 上，该存储可被源主机和目标主机访问。
- 迁移正在运行的虚拟机时，您的网络带宽必须高于虚拟机生成脏内存页面的速度。要在开始实时迁移前获得虚拟机的脏页面率，请在命令行界面中执行以下操作：

  a. 监控虚拟机在短时间内的脏页面生成率。

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

  b. 在监控完成后，获取其结果：

  ```
  # 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
  ```

  在本例中，VM 每秒会生成 2 MB 的脏内存页面。如果您不暂停虚拟机或降低其工作负载，那么在小于或等于 2 MB/秒带宽的网络中尝试实时迁移这样的虚拟机会导致实时迁移不会进行。

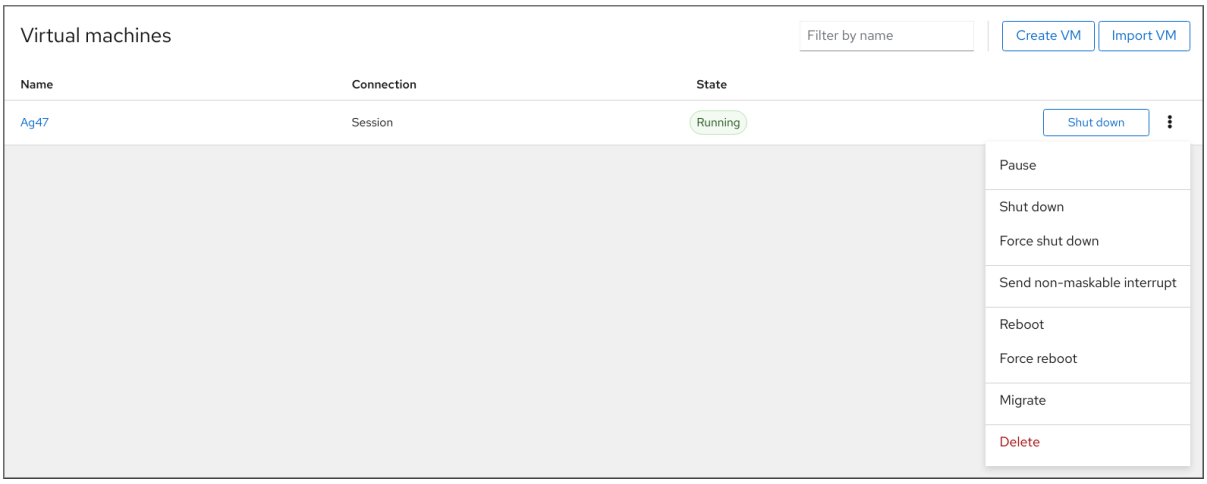
  为确保实时迁移成功完成，红帽建议您的网络带宽要明显大于虚拟机的脏页面生成率。

注意

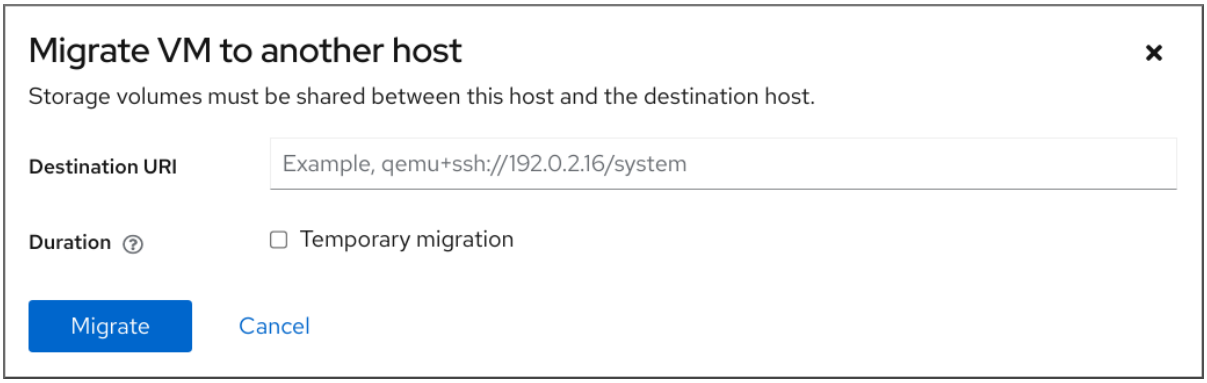
calc_period 选项的值可能因工作负载和脏页率而异。您可以使用多个 **calc_period** 值进行试验，以确定与您环境中脏页率一致的最合适的周期。

流程

1. 在 web 控制台的虚拟机界面中，单击您要迁移的虚拟机的菜单按钮 ⋮。
   此时会出现一个下拉菜单，控制各种虚拟机操作。



2. 点击 **Migrate**
   此时将显示将虚拟机迁移到另一主机的对话框。



3. 输入目标主机的 URI。
4. 配置迁移的持续时间：
   - **Permanent** - 如果您要永久迁移虚拟机，请勿选中该框。永久迁移会从源主机上完全删除虚拟机配置。
   - **Temporary** - 临时迁移会将虚拟机的副本迁移到目标主机。虚拟机关闭时，此副本将从目标主机中删除。原始虚拟机会保留在源主机上。
5. 点击 **Migrate**
   您的虚拟机被迁移到目标主机上。

验证

要验证虚拟机是否已成功迁移并正常工作：

- 确认虚拟机是否在目标主机上可用虚拟机的列表中。
- 启动迁移的虚拟机，并观察其是否启动。

12.8. 实时迁移带有附加 MELLANOX 虚拟功能的虚拟机

作为技术预览，您可以使用 Mellanox 网络设备的附加虚拟功能(VF)实时迁移虚拟机(VM)。目前，这只能在使用 Mellanox CX-7 网络设备时实现。Mellanox CX-7 网络设备上的 VF 使用一个新的 **mlx5_vfio_pci** 驱动程序，它添加了实时迁移所需的功能，并且 **libvirt** 自动将新驱动程序绑定到 VF。

限制

目前，当实时迁移带有附加 Mellanox 虚拟功能的虚拟机时，无法使用一些虚拟化功能：

- 计算虚拟机的脏内存页面速率生成。
- 使用复制后实时迁移。
- 在虚拟机中使用虚拟 I/O 内存管理单元(vIOMMU)设备。

重要

这个功能只 作为技术预览 包含在 RHEL 9 中，这意味着不支持它。

先决条件

- 您有一个 Mellanox CX-7 网络设备，其固件版本等于或大于 *28.36.1010*。
  有关固件版本的详情，请参阅 Mellanox 文档。

- **mstflint** 软件包安装在源和目标主机上：

```
# dnf install mstflint
```

- Mellanox CX-7 网络设备将 **VF_MIGRATION_MODE** 设置为 **MIGRATION_ENABLED** ：

```
# mstconfig -d <device_pci_address> query | grep -i VF_migration

VF_MIGRATION_MODE                   MIGRATION_ENABLED(2)
```

  - 您可以使用以下命令将 **VF_MIGRATION_MODE** 设置为 **MIGRATION_ENABLED** ：

```
# mstconfig -d <device_pci_address> set VF_MIGRATION_MODE=2
```

- **openvswitch** 软件包在源和目标主机上安装：

```
# dnf install openvswitch
```

- 您的主机的 CPU 和固件支持 I/O 内存管理单元(IOMMU)。
  - 如果使用 Intel CPU，它必须支持 Intel 的直接 I/O 虚拟化技术(VT-d)。
  - 如果使用 AMD CPU，则必须支持 AMD-Vi 功能。
- 主机系统使用访问控制服务(ACS)来为 PCIe 拓扑提供直接内存访问(DMA)隔离。与系统供应商一起验证这一点。
  如需更多信息，请参阅实施 SR-IOV 的硬件注意事项。
- 用于创建 VF 的主机网络接口正在运行。例如，要激活 *eth1* 接口并验证它正在运行，请使用以下命令：

```
# ip link set eth1 up
# ip link show eth1
8: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT qlen 1000
  link/ether a0:36:9f:8f:3f:b8 brd ff:ff:ff:ff:ff:ff
  vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
  vf 1 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
  vf 2 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
  vf 3 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
```

- 要使 SR-IOV 设备分配正常工作，必须在主机 BIOS 和内核中启用 IOMMU 功能。要做到这一点：
  - 在 Intel 主机上，为直接 I/O (VT-d)启用 Intel 虚拟化技术：
    i. 使用 **intel_iommu=on** 和 **iommu=pt** 参数重新生成 GRUB 配置：

```
# grubby --args="intel_iommu=on iommu=pt" --update-kernel=ALL
```

    ii. 重启主机。
  - 在 AMD 主机上启用 AMD-Vi：
    i. 使用 **iommu=pt** 参数重新生成 GRUB 配置：

```
# grubby --args="iommu=pt" --update-kernel=ALL
```

ii. 重启主机。

- 源主机和目标主机都使用 KVM 管理程序。
- 源主机和目标主机可以通过网络相互访问。使用 **ping** 工具进行验证。
- 目标主机上打开了以下端口：
  - 使用 SSH 连接到目标主机需要端口 22。
  - 使用 TLS 连接到目标主机需要端口 16509。
  - 使用 TCP 连接到目标主机需要端口 16514。
  - QEMU 需要端口 49152-49215 来传输内存和磁盘迁移数据。
- 源主机和目标主机使用允许迁移的操作系统和机器类型。要确保情况是这种情况，请查看支持的虚拟机迁移主机。
- 虚拟机必须与目标主机的 CPU 功能兼容。要确保情况如此，请参阅验证虚拟机迁移的主机 CPU 兼容性。
- 要迁移的虚拟机的磁盘镜像位于源主机和目标主机都可访问的单独的网络位置。这在离线迁移中是可选的，但在迁移运行的虚拟机时是必需的。
  有关设置这样的共享虚拟机存储的步骤，请参阅与其他主机共享虚拟机磁盘镜像。
- 迁移正在运行的虚拟机时，您的网络带宽必须高于虚拟机生成脏内存页面的速度。
- 启用虚拟网络套接字，对应于连接协议。
  在执行虚拟机迁移时，源主机上的 **virsh** 客户端可以使用多种协议之一连接到目标主机上的 libvirt 守护进程。以下流程中的示例使用 SSH 连接，但您可以选择不同的连接。** 如果您希望 libvirt 使用 SSH 连接，请确保启用了 **virtqemud** 套接字并在目标主机上运行。

  +

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

  - 如果您希望 libvirt 使用 TLS 连接，请确保启用 **virtproxyd-tls** 套接字并在目标主机上运行。

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

  - 如果您希望 libvirt 使用 TCP 连接，请确保 **virtproxyd-tcp** 套接字已经启用并在目标主机上运行。

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

流程

1. 在源主机上，将 Mellanox 网络设备设置为 **switchdev** 模式。

   ```
   # devlink dev eswitch set pci/<device_pci_address> mode switchdev
   ```

2. 在源主机上，在 Mellanox 设备中创建虚拟功能。

```
# echo 1 > /sys/bus/pci/devices/0000\:e1\:00.0/sriov_numvfs
```

文件路径的 **/0000\:e1\:00.0/** 部分基于设备的 PCI 地址。在示例中，为： **0000:e1:00.0**

3. 在源主机上，从其驱动程序中取消绑定 VF。

```
# virsh nodedev-detach <vf_pci_address> --driver pci-stub
```

您可以使用以下命令查看 VF 的 PCI 地址：

```
# lshw -c network -businfo

Bus info                  Device           Class          Description
=========================================================================

pci@0000:e1:00.0  enp225s0np0    network        MT2910 Family [ConnectX-7]
pci@0000:e1:00.1  enp225s0v0     network        ConnectX Family mlx5Gen Virtual Function
```

4. 在源主机上，启用 VF 的迁移功能。

```
# devlink port function set pci/0000:e1:00.0/1 migratable enable
```

在本例中，**pci/0000:e1:00.0/1** 代表带有给定 PCI 地址的 Mellanox 设备上的第一个 VF。

5. 在源主机上，配置 Open vSwitch (OVS)以迁移 VF。如果 Mellanox 设备处于 **switchdev** 模式，则无法通过网络传输数据。

   a. 确保 **openvswitch** 服务正在运行。

   ```
   # systemctl start openvswitch
   ```

   b. 启用硬件卸载以提高网络性能。

   ```
   # ovs-vsctl set Open_vSwitch . other_config:hw-offload=true
   ```

   c. 增加最大闲置时间，以确保网络连接在迁移过程中保持打开。

   ```
   # ovs-vsctl set Open_vSwitch . other_config:max-idle=300000
   ```

   d. 在 OVS 实例中创建一个新网桥。

   ```
   # ovs-vsctl add-br <bridge_name>
   ```

   e. 重新启动 **openvswitch** 服务。

   ```
   # systemctl restart openvswitch
   ```

   f. 将物理 Mellanox 设备添加到 OVS 网桥。

   ```
   # ovs-vsctl add-port <bridge_name> enp225s0np0
   ```

在本例中，& ***lt;bridge_name*** > 是您在步骤 *d* 中创建的网桥名称，**enp225s0np0** 是 Mellanox 设备的网络接口名称。

g. 将 Mellanox 设备的 VF 添加到 OVS 网桥。

```
# ovs-vsctl add-port <bridge_name> enp225s0npf0vf0
```

在本例中，& ***lt;bridge_name*** > 是您在步骤 *d* 中创建的网桥名称，**enp225s0npf0vf0** 是 VF 的网络接口名称。

6. **在目标主机上** 重复步骤 1-5。

7. 在源主机上，打开新文件，如 **mlx_vf.xml**，并添加 VF 的以下 XML 配置：

```
<interface type='hostdev' managed='yes'>
   <mac address='52:54:00:56:8c:f7'/>
   <source>
     <address type='pci' domain='0x0000' bus='0xe1' slot='0x00' function='0x1'/>
   </source>
</interface>
```

这个示例将 VF 的直通配置为虚拟机的网络接口。确保 MAC 地址是唯一的，并使用源主机上 VF 的 PCI 地址。

8. 在源主机上，将 VF XML 文件附加到虚拟机。

```
# virsh attach-device <vm_name> mlx_vf.xml --live --config
```

在本例中，**mlx_vf.xml** 是带有 VF 配置的 XML 文件的名称。使用 **--live** 选项将设备附加到正在运行的虚拟机中。

9. 在源主机上，使用附加 VF 启动正在运行的虚拟机的实时迁移。

```
# virsh migrate --live --domain <vm_name> --desturi
qemu+ssh://<destination_host_ip_address>/system
```

验证

1. 在迁移的虚拟机中，查看 Mellanox VF 的网络接口名称。

```
# ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.10  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::a00:27ff:fe4e:66a1  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:4e:66:a1  txqueuelen 1000  (Ethernet)
        RX packets 100000  bytes 6543210 (6.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 100000  bytes 6543210 (6.5 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

enp4s0f0v0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.3.10  netmask 255.255.255.0  broadcast 192.168.3.255
        inet6 fe80::a00:27ff:fe4e:66c3  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:4e:66:c3  txqueuelen 1000  (Ethernet)
```

```
        RX packets 200000  bytes 12345678 (12.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 200000  bytes 12345678 (12.3 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

2. 在迁移的虚拟机中，检查 Mellanox VF 是否正常工作，例如：

```
# ping -I <VF_interface_name> 8.8.8.8

PING 8.8.8.8 (8.8.8.8) from 192.168.3.10 <VF_interface_name>: 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=57 time=27.4 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=57 time=26.9 ms

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 26.944/27.046/27.148/0.102 ms
```

其它资源

- 使用命令行界面迁移虚拟机
- 虚拟机迁移故障排除

12.9. 虚拟机迁移故障排除

如果您在迁移虚拟机(VM)时面临以下问题之一，请参阅提供的说明来修复或避免问题。

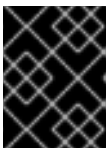
12.9.1. 虚拟机的实时迁移需要很长时间，且没有完成

原因

在某些情况下，迁移正在运行的虚拟机可能会比正常迁移更快地产生 *脏内存页*。当发生这种情况时，迁移无法成功完成。

以下场景经常导致这个问题：

- 在负载过重时实时迁移虚拟机
- 实时迁移一个使用大量内存的虚拟机，如 1 TB 或更多



重要

红帽已成功测试了具有最多 6TB 内存的 VM 的实时迁移。但是，对于涉及超过 1 TB 内存的虚拟机的实时迁移场景，客户应联系 红帽技术支持。

诊断

如果您的虚拟机实时迁移所需的时间比预期要长，请使用 **virsh domjobinfo** 命令获取虚拟机的内存页数据：

```
# virsh domjobinfo vm-name

Job type:         Unbounded
Operation:        Outgoing migration
```

```
Time elapsed:     168286974    ms
Data processed:   26.106 TiB
Data remaining:   34.383 MiB
Data total:       10.586 TiB
Memory processed: 26.106 TiB
Memory remaining: 34.383 MiB
Memory total:     10.586 TiB
Memory bandwidth: 29.056 MiB/s
Dirty rate: 17225 pages/s
Page size: 4096 bytes
```

在这个输出中，**Dirty rate** 和 **Page size** 的乘积大于 **Memory bandwidth**。这意味着，虚拟机比可以迁移它们的网络更快地产生脏内存页。因此，目标主机上虚拟机的状态无法与源主机上的虚拟机状态进行聚合，这阻止了迁移完成。

修复

要提高停滞的实时迁移成功完成的机会，您可以执行以下操作之一：

- 减少虚拟机的工作负载，特别是内存更新。
  - 为此，在源虚拟机的客户机操作系统中停止或取消不重要的进程。
- 增加实时迁移允许的停机时间：
  a. 在正在迁移的虚拟机实时迁移结束时显示当前的最长停机时间：

  ```
  # virsh migrate-getmaxdowntime vm-name
  ```

  b. 设置一个更高的最长停机时间：

  ```
  # virsh migrate-setmaxdowntime vm-name downtime-in-miliseconds
  ```

  设置的最长停机时间越高，迁移越有可能完成。
- 将实时迁移切换到 *post-copy* 模式。

  ```
  # virsh migrate-start-postcopy vm-name
  ```

  - 这样可确保虚拟机的内存页可以在目标主机上聚合，迁移可以完成。
    但是，当 post-copy 模式处于活跃状态时，虚拟机可能会由于从目标主机到源主机的远程页请求而显著减慢。另外，如果源主机和目标主机之间的网络连接在 post-copy 迁移过程中停止工作，一些虚拟机进程可能会因为缺少内存页而停止。

    因此，如果虚拟机可用性至关重要，或者迁移网络不稳定，请不要使用 post-copy 迁移。
- 如果您的工作负载允许，请挂起虚拟机，并使迁移作为 *非实时* 迁移完成。这会增加虚拟机的停机时间，但多数情况可确保迁移成功完成。

防止

成功完成虚拟机的实时迁移的可能性取决于以下方面：

- 在迁移过程中虚拟机的工作负载
  - 在开始迁移前，停止或取消虚拟机的客户机操作系统中不重要的进程。

- 主机可用于迁移的网络带宽
  - 要获得实时迁移的最佳结果，用于迁移的网络带宽必须显著高于虚拟机的脏页生成。有关获取 VM 脏页生成率的说明，请参阅 使用命令行界面迁移虚拟机 的先决条件。
  - 源主机和目标主机必须有用于迁移的专用的网络接口控制器(NIC)。对于实时迁移内存超过 1 TB 的虚拟机，红帽建议使用速度为 25 Gb/s 或更高的 NIC 。
  - 您还可以在启动迁移时使用 **--bandwidth** 选项来指定分配给实时迁移的网络带宽。对于迁移非常大的虚拟机，请为部署分配尽可能多的带宽。
- 实时迁移的模式
  - 默认的 *pre-copy* 迁移模式会在内存页变为脏状态时重复复制它们。
  - *Post-copy* 迁移仅复制内存页一次。
    要在迁移停滞时将实时迁移切换到 post-copy 模式，请在开始迁移时使用 **virsh migrate** 和 **--postcopy** 选项。
- 为部署指定的停机时间
  - 您可以使用 **virsh migrate-setmaxdowntime** 在迁移过程中调整它，如前面所述。

12.10. 虚拟机迁移支持的主机

要使虚拟机迁移正常工作且受红帽支持，源和目标主机必须是特定的 RHEL 版本和机器类型。下表显示了支持的虚拟机迁移路径。

表 12.2. 实时迁移兼容性

迁移方法	发行类型	将来的版本示例	支持状态
向前	次发行版本	9.0.1 → 9.1	在支持的 RHEL 9 系统上： 机器类型 **q35**。
向后	次发行版本	9.1 → 9.0.1	在支持的 RHEL 9 系统上： 机器类型 **q35**。

注意

对红帽提供的其他虚拟化解决方案（包括 RHOSP 和 OpenShift Virtualization）的支持等级有所不同。

第 13 章 使用快照保存和恢复虚拟机状态

要保存虚拟机的当前状态，您可以创建一个虚拟机的 *快照*。之后，您可以恢复到快照，来将虚拟机返回到保存的状态。

VM 快照包含虚拟机的磁盘镜像。如果您从正在运行的虚拟机创建一个快照（也称为 *实时快照*），则快照还包含虚拟机的内存状态，其中包括正在运行的进程和应用程序。

例如，对于以下任务，创建快照可能很有用：

- 保存客户机操作系统的干净状态
- 确保您在对虚拟机执行具有潜在的破坏性操作之前有一个恢复点

13.1. 对虚拟机快照的支持限制

红帽仅在使用 **external** 快照时支持对 RHEL 上虚拟机(VM)的快照功能。目前，只有在满足以下要求时，才能在 RHEL 上创建外部快照：

- 您的主机使用 RHEL 9.4 或更高版本。
- 虚拟机使用基于文件的存储。
- 您只能在以下场景之一创建虚拟机快照：
  - 虚拟机已关闭。
  - 如果虚拟机正在运行，您可以使用 **--disk-only --quiesce** 选项或 **--live --memspec** 选项。

大多数其他配置会创建 **内部** 快照，它们在 RHEL 9 中已弃用。内部快照可能适用于您的用例，但红帽不提供对它们的全面测试和支持。

警告

不要在生产环境中使用内部快照。

要确保快照被支持，请显示快照的 XML 配置，并检查快照类型和存储：

```
# virsh snapshot-dumpxml <vm-name> <snapshot-name>
```

- 支持的快照的输出示例：

```
<domainsnapshot>
  <name>sample-snapshot-name-1<name>
  <state>shutoff</state>
  <creationTime>1706658764</creationTime>
  <memory snapshot='no'/>
  <disks>
    <disk name='vda' snapshot='external' type='file'>
      <driver type='qcow2'/>
```

```
    <source file='/var/lib/libvirt/images/vm-name.sample-snapshot-name-1'/>
  </disk>
 </disks>
 <domain type='kvm'>
 [...]
```

- 不支持的快照的输出示例：

```
<domainsnapshot>
 <name>sample-snapshot-name-2</name>
 <state>running</state>
 <creationTime>1653396424</creationTime>
 <memory snapshot='internal'/>
 <disks>
  <disk name='vda' snapshot='internal'/>
  <disk name='sda' snapshot='no'/>
 </disks>
 <domain type='kvm'>
 [...]
```

13.2. 使用命令行界面创建虚拟机快照

要将虚拟机(VM)的状态保存到快照中，您可以使用 **virsh snapshot-create-as** 命令。

先决条件

- 您的主机使用 RHEL 9.4 或更高版本。
- 虚拟机使用基于文件的存储。要检查是否是这种情况，请使用以下命令，并确保对于 **disk** 设备，其 **disk type** 显示为 **file** ：

```
# virsh dumpxml <vm-name> | grep "disk type"
    <disk type='file' device='disk'>
    <disk type='file' device='cdrom'>
```

- 如果要创建一个包含正在运行的虚拟机内存的虚拟机快照，您必须有足够的磁盘空间来存储虚拟机的内存。
  - 保存虚拟机的内存的最少推荐空间等于虚拟机分配的 RAM。例如，保存具有 32 GB RAM 的虚拟机内存需要最多 32 GB 的磁盘空间。
  - 如果虚拟机 I/O 负载过重，则可能需要大量额外的磁盘空间。
  - 如果虚拟机已分配了 VFIO passthrough 设备，则可能需要额外的磁盘空间。
  - 如果快照是在不暂停虚拟机的情况下创建的，则可能需要额外的磁盘空间。

警告

红帽建议不保存在非常高的工作负载下运行的或使用 VFIO passthrough 设备的虚拟机的内存。保存此类虚拟机的内存可能会填满主机磁盘，并降低系统性能。相反，请考虑为此类虚拟机创建没有内存的快照。

另外，请注意，并非所有 VFIO 设备都能够创建具有内存的快照。目前，只有在附加的 VFIO 设备是启用了迁移功能的 Mellanox VF 时，创建带有内存的快照才能正常工作。

流程

- 要使用所需参数创建虚拟机快照，请使用 **virsh snapshot-create-as** 命令。

```
# virsh snapshot-create-as <vm-name> <snapshot-name> <optional-description> <additional-parameters>
```

  - 要创建关闭的虚拟机的快照，请使用 **--disk-only** 参数。例如，以下命令从关闭的 **Testguest1** 虚拟机的当前磁盘状态创建 **Snapshot1** ：

```
# virsh snapshot-create-as Testguest1 Snapshot1 --disk-only
Domain snapshot Snapshot1 created.
```

  - 要创建一个保存正在运行的虚拟机的磁盘状态而不是其内存的快照，请使用 **--disk-only --quiesce** 参数。例如，以下命令从正在运行的 **Testguest2** 虚拟机的当前磁盘状态创建 **Snapshot2**，描述为 **clean system install** ：

```
# virsh snapshot-create-as Testguest2 Snapshot2 "clean system install" --disk-only --quiesce
Domain snapshot Snapshot2 created.
```

  - 要创建一个暂停正在运行的虚拟机并保存其磁盘状态和内存的快照，请使用 **--memspec** 参数。例如，以下命令暂停 **Testguest3** 虚拟机，并从虚拟机的当前磁盘和内存状态创建 **Snapshot3**。VM 内存保存在 **/var/lib/libvirt/images/saved_memory.img** 文件中。快照完成后，虚拟机自动恢复操作。

```
# virsh snapshot-create-as Testguest3 Snapshot3 --memspec /var/lib/libvirt/images/saved_memory.img
Domain snapshot Snapshot3 created.
```

    在快照过程中暂停虚拟机会创建停机时间，但可能比创建正在运行的虚拟机的实时快照（使用 **--live** 选项）更可靠，特别是对于高负载下的虚拟机。

  - 要创建一个保存正在运行的虚拟机的磁盘状态及其实时内存的快照，请使用 **--live --memspec** 参数。例如，以下命令从正在运行的 **Testguest4** 虚拟机的当前磁盘和内存状态创建 **Snapshot4**，并将内存状态保存在 **/var/lib/libvirt/images/saved_memory2.img** 文件中。

```
# virsh snapshot-create-as Testguest4 Snapshot4 --live --memspec
/var/lib/libvirt/images/saved_memory2.img
Domain snapshot Snapshot4 created.
```

警告

将虚拟机的内存保存到快照中会将正在运行的进程的状态保存到虚拟机的客户机操作系统中。但是，当您恢复到此类快照时，进程可能会因为各种因素而失败，如丢失网络连接或未同步的系统时间。

验证

1. 列出与指定虚拟机关联的快照：

```
# virsh snapshot-list <Testguest1>

 Name                Creation Time              State
------------------------------------------------------------
Snapshot1            2024-01-30 18:34:58 +0100  shutoff
```

2. 验证快照是否已创建为 *external* ：

```
# virsh snapshot-dumpxml <Testguest1> <Snapshot1> | grep external

  <disk name='vda' snapshot='external' type='file'>
```

如果这个命令的输出包含 **snapshot='external'**，则快照是外部的，因此红帽完全支持。

后续步骤

- 使用 CLI 恢复到虚拟机快照
- 使用 Web 控制台恢复到虚拟机快照

其它资源

- 有关快照元数据的上游 libvirt 信息
- **virsh** 手册页

13.3. 使用 WEB 控制台创建虚拟机快照

要将虚拟机(VM)的状态保存到快照中，您可以使用 RHEL web 控制台。

先决条件

- 您的主机使用 RHEL 9.4 或更高版本。
- Web 控制台 VM 插件 已安装在您的系统上。

- 虚拟机使用基于文件的存储。要确保情况是这种情况，请执行以下步骤：

  a. 在 web 控制台的 **Virtual machines** 界面中，点您要创建快照的虚拟机。

  b. 在管理概述的 **Disks** 窗格中，检查列出的设备的 **Source** 列。在所有列出源的设备中，此源必须是 **File**。

流程

1. 在 web 控制台的 **Virtual machines** 界面中，点您要创建快照的虚拟机。
此时会打开虚拟机的管理概述。
2. 在管理概述的 **Snapshots** 窗格中，单击 **Create snapshot** 按钮。
3. 输入快照的名称，并可选择输入描述。
4. 点 **Create**。

验证

1. 要确保创建快照是否已成功，请检查快照现在是否列在虚拟机的 **Snapshots** 窗格中。
2. 验证快照是否已创建为 *external*。为此，请在主机的命令行界面中使用以下命令：

```
# virsh snapshot-dumpxml <Testguest1> <Snapshot1> | grep external

 <disk name='vda' snapshot='external' type='file'>
```

如果这个命令的输出包含 **snapshot='external'**，则快照是外部的，因此被红帽支持。

后续步骤

- 使用 Web 控制台恢复到虚拟机快照
- 使用命令行界面恢复到虚拟机快照

13.4. 使用命令行界面恢复到虚拟机快照

要将虚拟机(VM)返回到快照中保存的状态，您可以使用命令行界面(CLI)。

先决条件

- 您之前 在 web 控制台中 或通过 使用命令行界面 创建的虚拟机的快照可用。
- **可选：** 您已创建了虚拟机当前状态的快照。如果您恢复到以前没有保存当前状态的快照，则自上次快照以来在虚拟机上执行的更改将丢失。

流程

- 使用 **virsh snapshot-revert** 工具，并指定虚拟机的名称以及您要恢复到的快照的名称。例如：

```
# virsh snapshot-revert Testguest2 clean-install
Domain snapshot clean-install reverted
```

验证

验证

- 显示恢复的虚拟机的当前活跃的快照：

```
# virsh snapshot-current Testguest2 --name
clean-install
```

13.5. 使用 WEB 控制台恢复到虚拟机快照

要将虚拟机(VM)返回到快照中保存的状态，您可以使用 RHEL web 控制台。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。
- 您之前 在 web 控制台中 或通过 使用命令行界面 创建的虚拟机的快照可用。
- **可选：** 您已创建了虚拟机当前状态的快照。如果您恢复到以前没有保存当前状态的快照，则自上次快照以来在虚拟机上执行的更改将丢失。

流程

1. 在 web 控制台的 **Virtual machines** 界面中，点您要恢复其状态的虚拟机。
   此时会打开虚拟机的管理概述。
2. 在管理概述的 **Snapshots** 窗格中，单击您要恢复的快照旁边的 **Revert** 按钮。
3. 等待恢复操作完成。根据快照的大小以及其与当前状态如何不同，这可能需要最多几分钟。

验证

- 在 **Snapshots** 窗格中，如果现在在所选快照的左侧显示绿色复选符号，则您已成功恢复。

13.6. 使用命令行界面删除虚拟机快照

当虚拟机(VM)快照不再对您有用时，您可以在命令行界面中删除它，来释放其使用的磁盘空间。

先决条件

- **可选：** 您要删除的快照存在子快照。
  当您有一个活跃的快照并创建一个新快照时，会自动创建一个子快照。如果您删除了没有子快照的快照，则在从其父快照创建后，您将丢失快照中保存的任何更改。

  要查看虚拟机中快照的父子结构，请使用 **virsh snapshot-list --tree** 命令。以下示例显示 **Latest-snapshot** 为 **Redundant-snapshot** 的一个子快照。

```
# virsh snapshot-list --tree <vm-name>

Clean-install-snapshot
  |
  +- Redundant-snapshot
     |
     +- Latest-snapshot
```

流程

- 使用 **virsh snapshot-delete** 命令删除快照。例如，以下命令从 **Testguest1** 虚拟机中删除 **Redundant-snapshot** ：

```
# virsh snapshot-delete Testguest1 Redundant-snapshot
Domain snapshot Redundant-snapshot deleted
```

验证

- 要确保您删除的快照不再存在，请显示受影响的虚拟机的现有快照及其父-子结构：

```
# virsh snapshot-list --tree <Testguest1>

Clean-install-snapshot
  |
  +- Latest-snapshot
```

在本例中，**Redundant-snapshot** 已被删除，**Latest-snapshot** 已变为 **Clean-install-snapshot** 的子快照。

13.7. 使用 WEB 控制台删除虚拟机快照

当虚拟机(VM)快照不再对您有用时，您可以在 web 控制台中删除它，来释放其使用的磁盘空间。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。
- **可选：** 您要删除的快照存在子快照。
  当您有一个活跃的快照并创建一个新快照时，会自动创建一个子快照。如果您删除了没有子快照的快照，则在从其父快照创建后，您将丢失快照中保存的任何更改。

  要检查快照是否有子快照，请在虚拟机的 Web 控制台概述中 **Snapshots** 的 **Parent snapshot** 列中列出的快照中进行确认。

流程

1. 在 web 控制台的 **Virtual machines** 界面中，点击您要删除其快照的虚拟机。
   此时会打开虚拟机的管理概述。
2. 在管理概述的 **Snapshots** 窗格中，单击您要删除的快照旁边的 **Delete** 按钮。
3. 等待删除操作完成。根据快照的大小，这可能需要最多几分钟时间。

验证

- 如果快照不再出现在 **Snapshots** 窗格中，则它已被成功删除。

第 14 章 管理虚拟设备

管理虚拟机功能、特性和性能的最有效的方法之一是调整其*虚拟设备*。

以下章节提供了虚拟设备的 概述，以及如何使用 CLI 或 Web 控制台 来管理它们的说明。

14.1. 虚拟设备的工作原理

与物理机一样，虚拟机(VM)也要求专门的设备来为系统提供功能，如处理能力、内存、存储、网络或图形。物理系统通常将硬件设备用于这些目的。但是，由于虚拟机作为软件实现来工作，因此它们需要使用此类设备的软件抽象，称为 *虚拟设备*。

基础知识

附加到虚拟机的虚拟设备可在创建虚拟机时配置，也可以在现有虚拟机上管理。通常，只有在虚拟机处于关闭状态时才能从虚拟机挂载或卸载虚拟设备，但某些设备可在虚拟机运行时进行添加或删除。这个功能被称为设备*热插（hot plug）*和*热拔（hot unplug）*。

创建新虚拟机时，**libvirt** 会自动创建和配置一组默认的基本虚拟设备，除非用户另有指定。它们基于主机系统架构和机器类型，通常包括：

- CPU
- 内存
- 键盘
- 网络接口控制器（NIC）
- 各种设备控制器
- 一个视频卡
- 一个声卡

要在创建虚拟机后管理虚拟设备，请使用命令行界面(CLI)。但是，要管理虚拟存储设备和 NIC，您还可以使用 RHEL 9 web 控制台。

性能或灵活性

对于某些类型的设备，RHEL 9 支持多种实施，通常具有性能和灵活性之间的利弊。

例如，用于虚拟磁盘的物理存储可以以各种格式的文件（如 **qcow2** 或 **raw**）形式表示，并使用各种控制器呈现给虚拟机：

- 模拟控制器
- **virtio-scsi**
- **virtio-blk**

模拟控制器比 **virtio** 控制器慢，因为 **virtio** 设备是专为虚拟化设计的。另一方面，仿真控制器可以运行没有 **virtio** 设备驱动程序的操作系统。同样，**virtio-scsi** 为 SCSI 命令提供更完整的支持，并可以将更多磁盘附加到虚拟机。最后，**virtio-blk** 比 **virtio-scsi** 和仿真控制器提供更好的性能，但用例范围比较有限。例如，在使用 **virtio-blk** 时，无法将物理磁盘作为 LUN 设备附加到虚拟机。

有关虚拟设备类型的更多信息，请参阅 虚拟设备的类型。

14.2. 虚拟设备类型

RHEL 9 中的虚拟化可显示可附加到虚拟机的几种不同类型的虚拟设备：

模拟设备

模拟设备是广泛使用的物理设备的软件实现。为物理设备设计的驱动程序还与模拟设备兼容。因此可非常灵活地使用模拟设备。
但是，由于需要忠实地模拟特定类型的硬件，与相应的物理设备或更为优化的虚拟设备相比，模拟设备可能会会遭受显著的性能损失。

支持以下模拟设备类型：

- 虚拟 CPU(vCPU)，有大量 CPU 型号可供选择。模拟的性能影响显著取决于主机 CPU 和模拟 vCPU 之间的差异。
- 模拟系统组件，如 PCI 总线控制器。
- 模拟存储控制器，如 SATA、SCSI 甚至 IDE。
- 模拟声音设备，如 ICH9、ICH6 或 AC97。
- 模拟图形卡，如 VGA 卡。
- 模拟网络设备，如 rtl8139。

半虚拟设备

半虚拟（Paravirtualization）提供了向虚拟机公开虚拟设备的速度。半虚拟设备公开专用于虚拟机使用的接口，因此可显著提高设备的性能。RHEL 9 使用 *virtio* API 作为 hypervisor 和虚拟机之间的层，来为虚拟机提供半虚拟化设备。这个方法的缺陷在于它需要在客户端操作系统中使用特定的设备驱动程序。
建议您尽可能为虚拟机使用半虚拟设备而不是模拟设备，特别是当它们运行大量 I/O 的应用程序时。半虚拟设备减少 I/O 延迟并增加 I/O 吞吐量，在某些情况下可使其非常接近裸机性能。其它半虚拟设备还会在不能使用的虚拟机中添加功能。

支持以下半虚拟设备类型：

- 半虚拟化网络设备(**virtio-net**)。
- 半虚拟化存储控制器：
  - **virtio-blk** - 提供块设备模拟。
  - **virtio-scsi** - 提供更完整的 SCSI 模拟。
- 半虚拟时钟。
- 半虚拟化串行设备(**virtio-serial**)。
- 气球（balloon）设备(**virtio-balloon**)用于在虚拟机及其主机之间动态分配内存。
- 半虚拟随机数字生成器(**virtio-rng**)。

物理共享设备

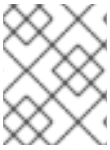
某些硬件平台可让虚拟机直接访问各种硬件设备和组件。这个过程被称为 *设备分配*或者 *透传（passthrough）*。

以这种方式连接时，物理设备的某些方面可以直接用于虚拟机，就像它们可用于物理机一样。这为虚拟机中使用的设备提供了出众的性能。但是，物理连接到虚拟机的设备对主机不可用，也无法迁移。

然而，某些设备可以在多个虚拟机之间 *共享*。例如，一个物理设备在某些情况下可以提供多个 *介质设备*，然后将其分配给不同的虚拟机。

支持以下 passthrough 设备类型：

- USB、PCI 和 SCSI 直通 - 直接向虚拟机公开通用的行业标准总线，以便其特定功能对虚拟客户机软件可用。
- 单根 I/O 虚拟化(SR-IOV)- 一种支持 PCI Express 资源的硬件强制隔离的规范。这使得将单个物理 PCI 资源划分成虚拟 PCI 功能变得安全而高效。它通常用于网络接口卡(NIC)。
- N_Port ID 虚拟化(NPIV)- 一种光纤通道技术，来与多个虚拟端口共享单个物理主机总线适配器(HBA)。
- GPU 和 vGPU - 用于特定图形或计算工作负载的加速器。一些 GPU 可以直接连接到虚拟机，而某些类型也提供能够创建共享底层物理硬件的虚拟 GPU(vGPU)的能力。



注意

这些类型的某些设备可能不被支持，或者与 RHEL 不兼容。如果您需要帮助设置虚拟设备，请咨询红帽支持。

14.3. 使用 CLI 管理附加到虚拟机的设备

要修改虚拟机的功能，您可以使用命令行界面(CLI)管理附加到虚拟机的设备。

您可以使用 CLI 来：

- 附加设备
- 修改设备
- 删除设备

14.3.1. 将设备附加到虚拟机

您可以通过附加新的虚拟设备来向虚拟机(VM)添加特定的功能。

以下流程使用命令行界面(CLI)创建虚拟设备，并将其附加到虚拟机(VM)。一些设备也可以 使用 RHEL web 控制台 附加到虚拟机。

例如，您可以通过将新虚拟磁盘设备附加到虚拟机来增加虚拟机的存储容量。这也被称为**内存热插拔**。

警告

RHEL 9 不支持从虚拟机中删除内存设备（也称为**内存热拔**）。红帽不建议使用它。

先决条件

- 获取您要附加到虚拟机的设备所需的选项。要查看特定设备的可用选项，请使用 **virt-xml --*device*=?** 命令。例如：

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

流程

1. 要将设备附加到虚拟机，请使用 **virt-xml --add-device** 命令，包括该设备的定义和所需的选项：

   - 例如，以下命令会在 **/var/lib/libvirt/images/** 目录中创建一个 20GB 的 *newdisk* qcow2 磁盘镜像，并在虚拟机启动时将其作为虚拟磁盘附加到运行的 *testguest* 虚拟机上：

     ```
     # 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.
     ```

   - 在虚拟机运行时，下面的命令会将一个 USB 盘（在主机的 002 总线中作为设备 004）附加到 *testguest2* 虚拟机：

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

     可以使用 **lsusb** 命令获得用于定义 USB 的总线设备组合。

验证

要验证设备是否已被添加，请执行以下操作之一：

- 使用 **virsh dumpxml** 命令，并查看设备的 XML 定义是否已添加到虚拟机 XML 配置中的 **<devices>** 部分。
  例如，以下输出显示了 *testguest* 虚拟机的配置，并确认已添加了 002.004 USB 闪存磁盘设备。

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

- 运行虚拟机并测试该设备是否存在并正常工作。

其它资源

- **man virt-xml** 命令

14.3.2. 修改附加到虚拟机的设备

您可以通过编辑附加的虚拟设备的配置来更改虚拟机(VM)的功能。例如，如果想要优化虚拟机的性能，您可以更改其虚拟 CPU 型号来更好地匹配主机的 CPU 。

以下流程提供了使用命令行界面(CLI)修改虚拟设备的通用说明。还可以 使用 RHEL 9 web 控制台 来修改附加到虚拟机的某些设备，如磁盘和 NIC。

先决条件

- 获取您要附加到虚拟机的设备所需的选项。要查看特定设备的可用选项，请使用 **virt-xml --*device*=?** 命令。例如：

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

- **可选：** 使用 **virsh dumpxml *vm-name*** 备份虚拟机的 XML 配置，并将结果发送到文件中。例如，以下命令将 *testguest1* 虚拟机的配置备份为 **testguest1.xml** 文件：

```
# virsh dumpxml testguest1 > testguest1.xml
# cat testguest1.xml
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  <name>testguest1</name>
  <uuid>ede29304-fe0c-4ca4-abcd-d246481acd18</uuid>
  [...]
</domain>
```

流程

1. 使用 **virt-xml --edit** 命令，包括设备的定义以及所需的选项：
例如，以下可清除关闭的 *testguest* 虚拟机的 *<cpu>* 配置，并将其设置为 *host-model*：

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

验证

要校验设备已被修改，请执行以下任一操作：

- 运行虚拟机并测试该设备是否存在并反映了所做的修改。
- 使用 **virsh dumpxml** 命令，并查看在虚拟机的 XML 配置中是否已修改了设备的 XML 定义。

例如,以下输出显示了 *testguest* 虚拟机的 配置，并确认 CPU 模式已被配置为 *host-model*。

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

故障排除

- 如果修改设备会导致虚拟机无法启动，请使用 **virsh 定义** 工具通过重新加载之前备份的 XML 配置文件来恢复 XML 配置。

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

注意

对于对虚拟机 XML 配置的小更改，您可以使用 **virsh edit** 命令 - 例如 **virsh edit testguest**。但是，对于更广泛的更改，请勿使用此方法，因为它很可能会破坏配置，从而阻止虚拟机启动。

其它资源

- **man virt-xml** 命令

14.3.3. 从虚拟机中删除设备

您可以通过移除虚拟设备来更改虚拟机(VM)的功能。例如，如果不再需要，您可以从其中一个虚拟机中删除虚拟磁盘设备。

以下流程演示了如何使用命令行界面(CLI)从虚拟机(VM)中删除虚拟设备。还可 使用 RHEL 9 web 控制台从虚拟机中删除一些设备，如磁盘或 NIC。

先决条件

- **可选：** 使用 **virsh dumpxml *vm-name*** 备份虚拟机的 XML 配置，并将结果发送到文件中。例如，以下命令将 *testguest1* 虚拟机的配置备份为 **testguest1.xml** 文件：

```
# virsh dumpxml testguest1 > testguest1.xml
# cat testguest1.xml
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  <name>testguest1</name>
  <uuid>ede29304-fe0c-4ca4-abcd-d246481acd18</uuid>
  [...]
</domain>
```

流程

1. 使用 **virt-xml --remove-device** 命令，包括设备的定义。例如：
   - 以下会在运行的 *testguest* 虚拟机关闭后，从其中删除标记为 *vdb* 的存储设备：

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

- 以下命令会立即从运行的 *testguest2* 虚拟机中删除 USB 闪存驱动器设备：

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

故障排除

- 如果移除设备会导致虚拟机无法启动，请使用 **virsh define** 工具，通过重新加载之前备份的 XML 配置文件来恢复 XML 配置。

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

其它资源

- **man virt-xml** 命令

14.4. 使用 WEB 控制台管理主机设备

要修改虚拟机的功能，您可以使用 Red Hat Enterprise Linux 9 web 控制台管理附加到虚拟机的主机设备。

主机设备是附加到主机系统的物理设备。根据您的需求，您可以让虚拟机直接访问这些硬件设备和组件。

您可以使用 Web 控制台：

- 查看设备
- 附加设备
- 删除设备

14.4.1. 使用 web 控制台查看附加到虚拟机的设备

在添加或修改附加到虚拟机(VM)的设备之前，您可能需要查看已附加到虚拟机的设备。以下流程提供了使用 Web 控制台查看这些设备的说明。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 在 **Virtual Machines** 界面中，点击您要查看其信息的虚拟机。
   这会打开一个新页面，其中包含有关虚拟机的详细信息。
2. 滚动到 **Host devices** 部分。

Host devices

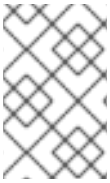
Type	Class	Model	Vendor	Source
usb		CHERRY Corded Device	Cherry GmbH	Device 002 Bus 001
usb		Optical Mouse	Lenovo	Device 003 Bus 001
pci	Network controller	Ethernet Connection I219-LM	Intel Corporation	Slot 0000:00:1f.6

其它资源

- 管理虚拟设备

14.4.2. 使用 web 控制台将设备附加到虚拟机

要为虚拟机添加特定的功能，您可以使用 web 控制台将主机设备附加到虚拟机。



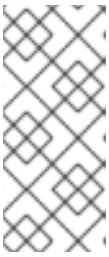
注意

不能同时附加多个主机设备。一次只能附加一个设备。

如需更多信息，请参阅 RHEL 9 已知问题。

先决条件

- 如果要附加 PCI 设备，请确保 **hostdev** 元素的 **managed** 属性的状态被设为 **yes**。



注意

将 PCI 设备附加到虚拟机时，请不要省略 **hostdev** 元素的 **managed** 属性，或者将其设为 **no**。如果您这样做了，当把 PCI 设备传给虚拟机时，会无法自动将 PCI 设备与主机分离。当您关闭虚拟机时，也不能自动将它们重新附加到主机上。

因此，主机可能会变得无响应，或者意外关闭。

您可以在虚拟机的 XML 配置中找到 **managed** 属性的状态。以下示例打开 **example-VM-1** 虚拟机的 XML 配置。

```
# virsh edit example-VM-1
```

- 备份虚拟机中的重要数据。
- **可选：** 备份虚拟机的 XML 配置。例如，要备份 **example-VM-1** 虚拟机：

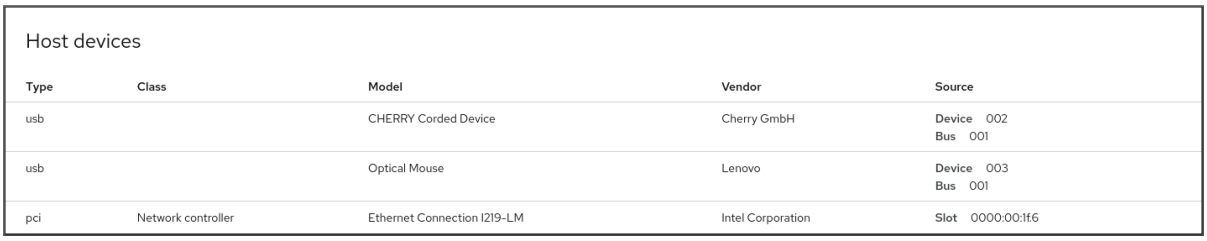
```
# virsh dumpxml example-VM-1 > example-VM-1.xml
```

- Web 控制台 VM 插件已安装在您的系统上。

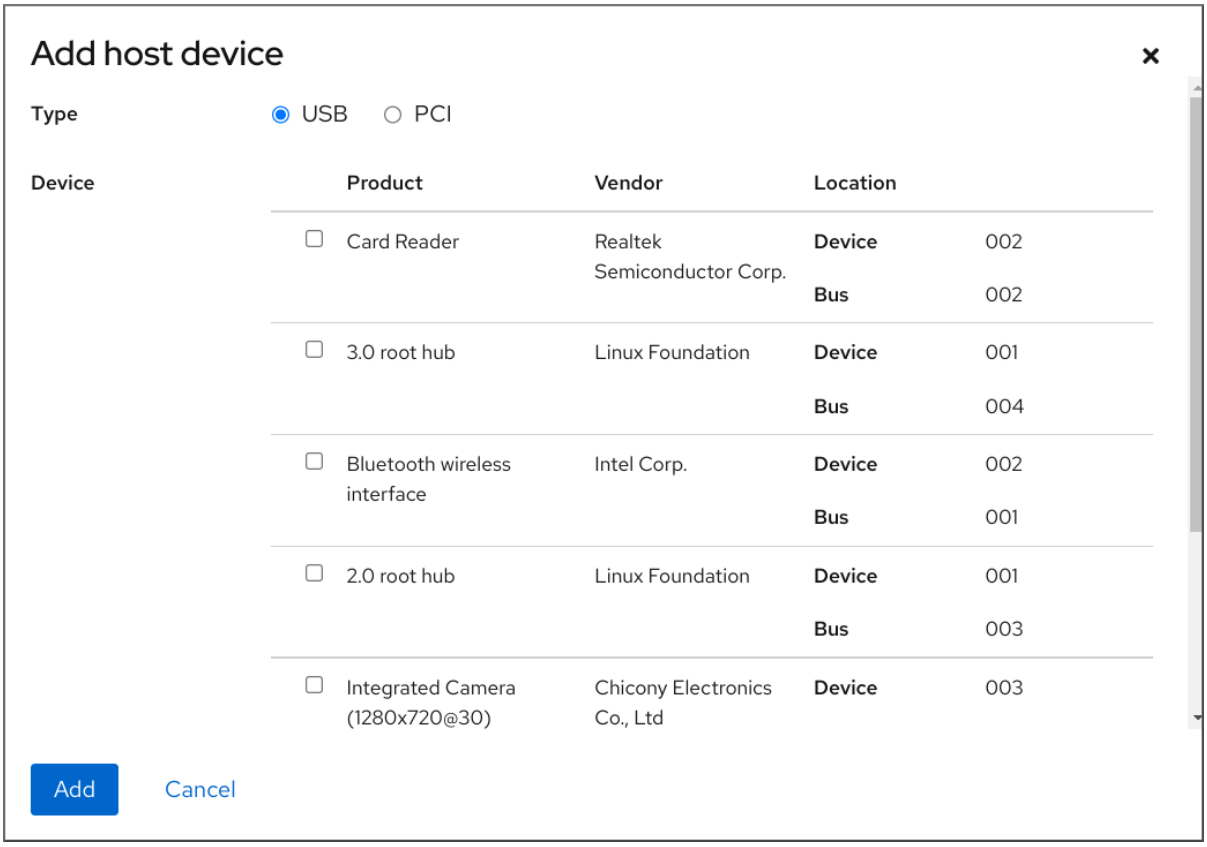
流程

1. 在 **Virtual Machines** 接口中，点击您要给其附加主机设备的虚拟机。
   这会打开一个新页面，其中包含有关所选虚拟机基本信息的 **概述**，以及访问虚拟机图形界面的 **控制台** 部分。
2. 滚动到 **Host devices**。

主机设备 部分显示有关附加到虚拟机的设备的信息，以及 **A添加** 或 **删除** 设备的选项。



3. 点击 **Add host device**。
此时会出现 **Add host device** 对话框。



4. 选择您要附加到虚拟机的设备。

5. 点 **添加**
所选设备被附加到虚拟机。

验证

- 运行虚拟机并检查该设备是否出现在 **Host devices** 部分中。

14.4.3. 使用 web 控制台从虚拟机中删除设备

要释放资源，修改虚拟机的功能或两个都做，您可以使用 Web 控制台来修改虚拟机，并删除不再需要的主机设备。

警告

使用 Web 控制台删除附加的 USB 主机设备可能会失败，因为设备和 USB 设备的总线号之间的关联不正确。

如需更多信息，请参阅 RHEL 9 已知问题。

作为临时解决方案，使用 *virsh* 工具从虚拟机的 XML 配置中删除 USB 设备的 <hostdev> 部分。以下示例打开 **example-VM-1** 虚拟机的 XML 配置：

```
# virsh edit <example-VM-1>
```

先决条件

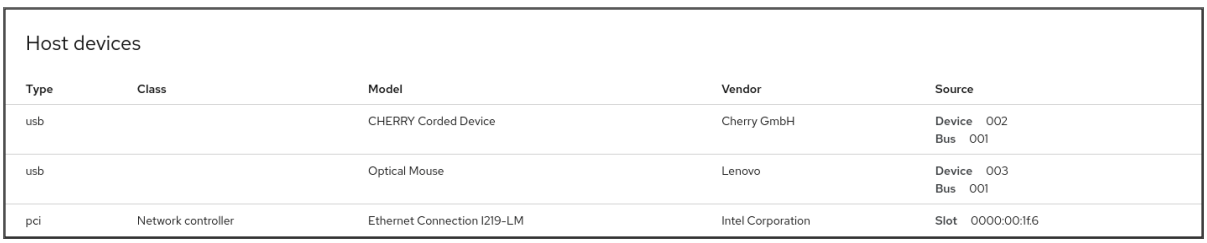
- Web 控制台 VM 插件已安装在您的系统上。
- 可选：使用 **virsh dumpxml *example-VM-1*** 备份虚拟机的 XML 配置，并将输出发送到文件中。例如，以下命令将 *testguest1* 虚拟机的配置备份为 **testguest1.xml** 文件：

```
# virsh dumpxml testguest1 > testguest1.xml
# cat testguest1.xml
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
  <name>testguest1</name>
  <uuid>ede29304-fe0c-4ca4-abcd-d246481acd18</uuid>
  [...]
</domain>
```

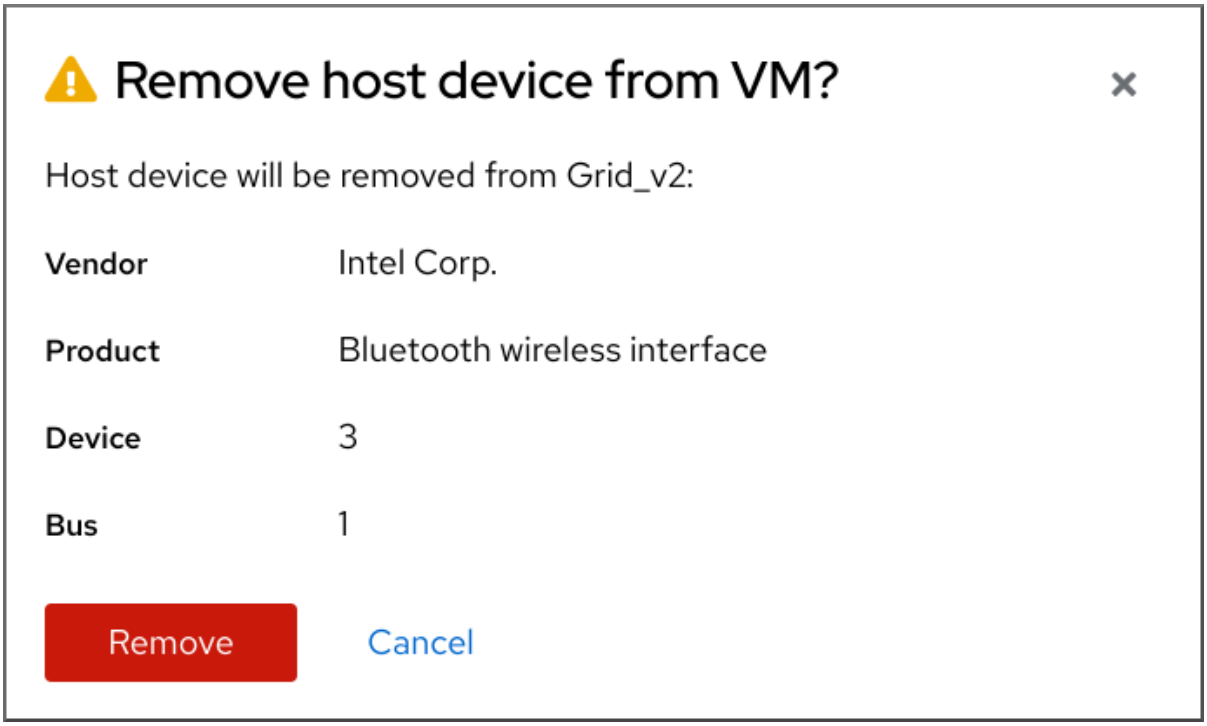
流程

1. 在 **Virtual Machines** 接口中，点击您要从中删除主机设备的虚拟机。
这会打开一个新页面，其中包含有关所选虚拟机基本信息的 **概述**，以及访问虚拟机图形界面的 **控制台** 部分。

2. 滚动到 **Host devices**。
主机设备 部分显示有关附加到虚拟机的设备的信息，以及 **A添加** 或 **删除** 设备的选项。



3. 点击您要从虚拟机中删除设备旁边的 **Remove** 按钮。
此时会出现删除设备确认对话框。



4. 点击 **Remove**。
该设备从虚拟机中删除。

故障排除

- 如果删除主机设备导致虚拟机无法启动，请使用 **virsh define** 工具通过重新载入之前备份的 XML 配置文件来恢复 XML 配置。

```
# virsh define testguest1.xml
```

14.5. 管理虚拟 USB 设备

使用虚拟机(VM)时，您可以访问并控制 USB 设备，如附加到主机系统的如闪存驱动器或 Web 相机。在这种情况下，主机系统会将设备的控制权传递给虚拟机。这也被称为 USB-passthrough。

以下部分提供有关使用命令行的信息：

- 将 USB 设备附加到虚拟机
- 从虚拟机中删除 USB 设备

14.5.1. 将 USB 设备附加到虚拟机

要将 USB 设备附加到虚拟机，您可以在虚拟机 XML 配置文件中包含 USB 设备信息。

先决条件

- 确定您要传递给虚拟机的设备已附加到主机。

流程

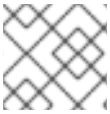
1. 找到您要附加到虚拟机的 USB 总线和设备值。

例如：以下命令显示附加到该主机的 USB 设备列表。在这个示例中，使用的设备作为设备 005 总线附加到总线 001 中。

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

2. 使用 **virt-xml** 工具及 **--add-device** 参数。
例如，以下命令将 USB 闪存驱动器附加到 **example-VM-1** 虚拟机。

```
# virt-xml example-VM-1 --add-device --hostdev 001.005
Domain 'example-VM-1' defined successfully.
```



注意

要将 USB 设备连接到正在运行的虚拟机，请将 **--update** 参数添加到上一命令中。

验证

- 运行虚拟机并测试该设备是否存在并正常工作。
- 使用 **virsh dumpxml** 命令查看设备的 XML 定义是否已添加到虚拟机 XML 配置文件中的 <devices> 部分。

```
# virsh dumpxml example-VM-1
[...]
<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>
[...]
```

其它资源

- **virt-xml (1)** 手册页
- 将设备附加到虚拟机

14.5.2. 从虚拟机中删除 USB 设备

要从虚拟机中删除 USB 设备，您可以从虚拟机 XML 配置中删除 USB 设备信息。

流程

1. 找到您要从虚拟机中删除的 USB 的 bus 和 device 值。

例如：以下命令显示附加到该主机的 USB 设备列表。在这个示例中，使用的设备作为设备 005 总线附加到总线 001 中。

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

2. 使用 **virt-xml** 工具及 **--remove-device** 参数。
例如，以下命令从 **example-VM-1** 虚拟机中删除附加到主机作为总线 001 上设备 005 的 USB 闪存驱动器。

```
# virt-xml example-VM-1 --remove-device --hostdev 001.005
Domain 'example-VM-1' defined successfully.
```

注意

要从正在运行的虚拟机中删除 USB 设备，请将 **--update** 参数添加到上一命令中。

验证

- 运行虚拟机并检查该设备是否已从设备列表中删除。

其它资源

- **virt-xml (1)** 手册页
- 将设备附加到虚拟机

14.6. 管理虚拟光驱

当使用虚拟机时，您可以访问保存在主机中 ISO 镜像中的信息。要做到这一点，请将 ISO 镜像作为虚拟光驱附加到虚拟机，比如 CD 驱动器或者 DVD 驱动器。

以下部分提供有关使用命令行的信息：

- 将驱动器和 ISO 镜像附加 到虚拟机
- 将 CD-ROM 附加到 正在运行的虚拟机
- 替换虚拟光驱中的 ISO 镜像
- 从虚拟光驱中删除 ISO 镜像
- 从虚拟机中删除驱动器
- 从正在运行的虚拟机中 删除 CD-ROM

14.6.1. 为虚拟机附加光驱

要将 ISO 镜像作为虚拟光驱附加，请编辑虚拟机的 XML 配置文件并添加新驱动器。

先决条件

- 您必须在主机机器上存储和复制 ISO 镜像的路径。

流程

- 使用带有 **--add-device** 参数的 **virt-xml** 工具：
  例如，以下命令将存储在 **/home/username/Downloads** 目录中的 **example-ISO-name** ISO 镜像附加到 **example-VM-name** 虚拟机。

```
# virt-xml example-VM-name --add-device --disk /home/username/Downloads/example-ISO-name.iso,device=cdrom
Domain 'example-VM-name' defined successfully.
```

验证

- 运行虚拟机并测试该设备是否存在并正常工作。

其它资源

- **man virt-xml** 命令
- 将设备附加到虚拟机

14.6.2. 使用 web 控制台将 CD-ROM 添加到正在运行的虚拟机上

您可以使用 web 控制台将 CD-ROM 插入到正在运行的虚拟机(VM)，而无需指定介质。

先决条件

- 您已在系统上安装了 Web 控制台虚拟机插件。

流程

1. 关闭虚拟机。
2. 在不指定源镜像的情况下附加一个虚拟 CD-ROM 设备。

```
# virt-xml vmname --add-device --disk target.dev=sda,device=cdrom
```

3. 运行虚拟机。
4. 打开 web 控制台，并在 **Virtual Machines** 界面中点击您要附加 CD-ROM 的虚拟机。
5. 滚动到 **磁盘** 。
   Disks 部分显示有关分配给虚拟机的磁盘的信息，以及用于 **Add** 或 **Edit** 磁盘的选项。
6. 点 **cdrom** 设备的 **Insert** 选项。

7. 为您要附加的文件选择一个 **Source** ：
   - **Custom Path** ：文件位于主机上的自定义目录中。
   - **Use existing** ： 文件位于您创建的存储池中。
8. 点 **Insert**。

验证

- 在 **Virtual Machines** 接口中，该文件将出现在 **Disks** 部分下。

14.6.3. 使用虚拟光驱替换 ISO 镜像

要替换作为虚拟光驱附加到虚拟机(VM)的 ISO 镜像，请编辑虚拟机的 XML 配置文件，并指定替换。

先决条件

- 您必须将 ISO 镜像存储在主机机器上。
- 您必须知道 ISO 镜像的路径。

流程

1. 定位 CD-ROM 附加到虚拟机的目标设备。您可以在虚拟机 XML 配置文件中找到这些信息。
   例如：以下命令显示 ***example-VM-name*** 虚拟机的 XML 配置文件，其中用于 CD-ROM 的目标设备是 **sda**。

```
# virsh dumpxml example-VM-name
...
<disk>
  ...
  <source file='$(/home/username/Downloads/example-ISO-name.iso)'/>
  <target dev='sda' bus='sata'/>
  ...
</disk>
...
```

2. 使用 **virt-xml** 工具及 **--edit** 参数。
   例如，以下命令使用存储在 **/dev/cdrom** 目录中的 **example-ISO-name-2** ISO 镜像替换附加到 **example-VM-name** 虚拟机目标 **sda** 处的 **example-ISO-name** ISO 镜像。

```
# virt-xml example-VM-name --edit target=sda --disk /dev/cdrom/example-ISO-name-2.iso
Domain 'example-VM-name' defined successfully.
```

验证

- 运行虚拟机并测试是否替换该设备并正常工作。

其它资源

- **man virt-xml** 命令

14.6.4. 从虚拟光驱中删除 ISO 镜像

要从附加到虚拟机(VM)的虚拟光驱中删除 ISO 镜像，请编辑虚拟机的 XML 配置文件。

流程

1. 定位 CD-ROM 附加到虚拟机的目标设备。您可以在虚拟机 XML 配置文件中找到这些信息。
例如：以下命令显示 **example-VM-name** 虚拟机的 XML 配置文件，其中用于 CD-ROM 的目标设备是 **sda**。

```
# virsh dumpxml example-VM-name
...
<disk>
  ...
  <source file='$(/home/username/Downloads/example-ISO-name.iso)'/>
  <target dev='sda' bus='sata'/>
  ...
</disk>
...
```

2. 使用 **virt-xml** 工具及 **--edit** 参数。
例如，以下命令从附加到 **example-VM-name** 虚拟机的 CD 驱动器中删除 **example-ISO-name** ISO 镜像。

```
# virt-xml example-VM-name --edit target=sda --disk path=
Domain 'example-VM-name' defined successfully.
```

验证

- 运行虚拟机，检查镜像已不再可用。

其它资源

- **man virt-xml** 命令

14.6.5. 从虚拟机中删除光驱

要删除附加到虚拟机的光驱，编辑虚拟机的 XML 配置文件。

流程

1. 定位 CD-ROM 附加到虚拟机的目标设备。您可以在虚拟机 XML 配置文件中找到这些信息。
例如：以下命令显示 **example-VM-name** 虚拟机的 XML 配置文件，其中用于 CD-ROM 的目标设备是 **sda**。

```
# virsh dumpxml example-VM-name
...
<disk type='file' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <target dev='sda' bus='sata'/>
  ...
</disk>
...
```

2. 使用带 **--remove-device** 参数的**virt-xml** 工具。
例如，以下命令从 **example-VM-name** 虚拟机中删除作为目标 **sda** 附加的光驱。

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

验证

- 确认该设备不再列在虚拟机 XML 配置文件中。

其它资源

- **man virt-xml** 命令

14.6.6. 使用 web 控制台从正在运行的虚拟机中删除 CD-ROM

您可以使用 web 控制台从正在运行的虚拟机(VM)中弹出 CD-ROM 设备。

先决条件

- 您已在系统上安装了 Web 控制台虚拟机插件。

流程

1. 在 **Virtual Machines** 接口中，点击您要从中删除 CD-ROM 的虚拟机。
2. 滚动到 **磁盘** 。
Disks 部分显示有关分配给虚拟机的磁盘的信息，以及用于 **Add** 或 **Edit** 磁盘的选项。

3. 点 cdrom 设备的 **Eject** 选项。
Eject media from VM?对话框将打开。
4. 单击 **Eject**。

验证

- 在 **Virtual Machines** 接口中，附加的文件不再显示在 **Disks** 部分下。

14.7. 管理 SR-IOV 设备

模拟虚拟设备通常使用比硬件网络设备更多的 CPU 和内存。这可能会限制虚拟机的性能。但是，如果虚拟化主机上的任何设备都支持单根 I/O 虚拟化(SR-IOV)，您可以使用此功能来提高设备性能，并可能还提高虚拟机的整体性能。

14.7.1. 什么是 SR-IOV?

单根 I/O 虚拟化(SR-IOV)是一种规范，它允许单个 PCI Express(PCIe)设备向主机系统呈现多个独立的 PCI 设备，称为 *虚拟功能* (VF)。这样的每个设备：

- 提供与原始 PCI 设备相同的或类似的服务。
- 出现在主机 PCI 总线的不同地址上。
- 可使用 VFIO 分配分配给不同的虚拟机。

例如，单个具有 SR-IOV 的网络设备可以向多个虚拟机显示 VF。虽然所有 VF 都使用相同的物理卡、相同的网络连接和相同的网线，但每个虚拟机都直接控制其自己的硬件网络设备，并且不使用主机的额外资源。

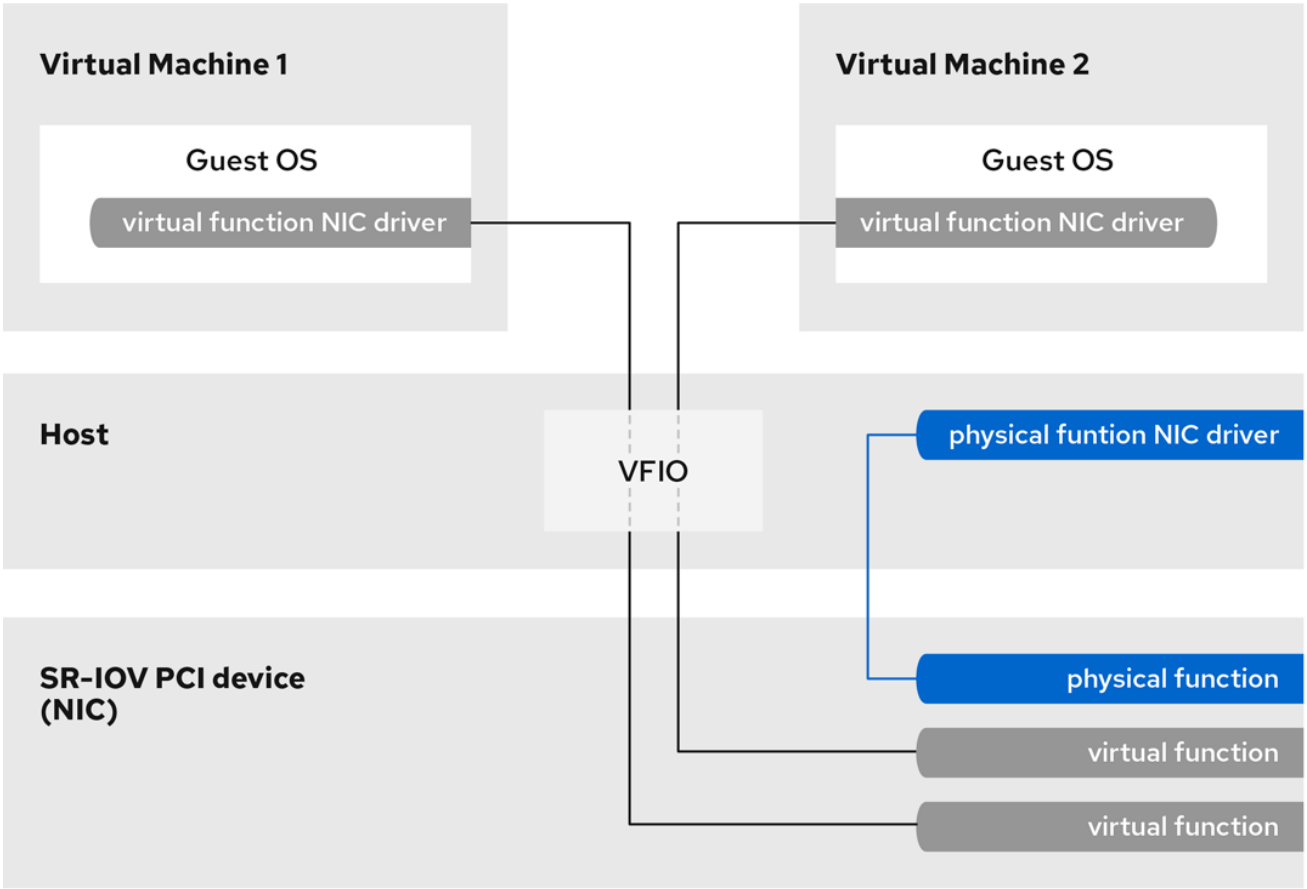
SR-IOV 的工作原理

SR-IOV 功能可能会因为引进了以下 PCI 功能：

- **物理功能(PF)** - 为主机提供设备（如网络）功能的 PCIe 功能，但也可以创建和管理一组 VF。每个具有 SR-IOV 功能的设备都有一个或多个 PF。
- **虚拟功能(VF)** - 充当独立设备的轻量级 PCIe 功能。每个 VF 都是从 PF 中派生的。一个设备可依赖于设备硬件的最大 VF 数。每个 VF 每次只能分配给一个虚拟机，但虚拟机可以分配多个 VF。

VM 将 VF 识别为虚拟设备。例如，由 SR-IOV 网络设备创建的 VF 显示为分配给虚拟机的网卡，其方式与主机系统上显示的物理网卡相同。

图 14.1. SR-IOV 架构



优点

使用 SR-IOV VF 而不是模拟设备的主要优点是：

- 提高的性能

- 减少主机 CPU 和内存资源使用量

例如，作为 vNIC 附加到虚拟机的 VF 性能几乎与物理 NIC 相同，并且优于半虚拟化或模拟的 NIC。特别是，当多个 VF 在单个主机上同时使用时，其性能优势可能非常显著。

缺点

- 要修改 PF 的配置，您必须首先将 PF 公开的 VF 数量改为零。因此，您还需要将这些 VF 提供的设备从分配给虚拟机的设备中删除。
- 附加了 VFIO 分配设备的虚拟机（包括 SR-IOV VF）无法迁移到另一台主机。在某些情况下，您可以通过将分配的设备与模拟的设备进行配对来临时解决这个限制。例如，您可以将分配的网络 VF 绑定 到模拟的 vNIC 中，并在迁移前删除 VF。
- 另外，分配了 VFIO 的设备需要固定虚拟机内存，这会增加虚拟机的内存消耗，并防止在虚拟机上使用内存膨胀。

其它资源

- SR-IOV 分配支持的设备
- 在 IBM Z 上配置 passthrough PCI 设备

14.7.2. 将 SR-IOV 网络设备附加到虚拟机

要将 SR-IOV 网络设备附加到 Intel 或 AMD 主机上的虚拟机(VM)，您必须从主机上支持 SR-IOV 的网络接口创建一个虚拟功能(VF)，并将 VF 作为设备分配给指定虚拟机。详情请查看以下步骤。

先决条件

- 您的主机的 CPU 和固件支持 I/O 内存管理单元(IOMMU)。
  - 如果使用 Intel CPU，它必须支持 Intel 的直接 I/O 虚拟化技术(VT-d)。
  - 如果使用 AMD CPU，则必须支持 AMD-Vi 功能。
- 主机系统使用访问控制服务(ACS)来为 PCIe 拓扑提供直接内存访问(DMA)隔离。与系统供应商一起验证这一点。
  如需更多信息，请参阅实施 SR-IOV 的硬件注意事项。
- 物理网络设备支持 SR-IOV。要验证系统上的任何网络设备是否支持 SR-IOV，请使用 **lspci -v** 命令，并在输出中查找 **单根 I/O 虚拟化(SR-IOV)** 。

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

- 用于创建 VF 的主机网络接口正在运行。例如：要激活 *eth1* 接口并验证它正在运行：

```
# ip link set eth1 up
# ip link show eth1
8: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT qlen 1000
  link/ether a0:36:9f:8f:3f:b8 brd ff:ff:ff:ff:ff:ff
  vf 0 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
  vf 1 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
  vf 2 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
  vf 3 MAC 00:00:00:00:00:00, spoof checking on, link-state auto
```

- 要使 SR-IOV 设备分配正常工作，必须在主机 BIOS 和内核中启用 IOMMU 功能。要做到这一点：
  - 在 Intel 主机上启用 VT-d：
    i. 使用 **intel_iommu=on** 和 **iommu=pt** 参数重新生成 GRUB 配置：

    ```
    # grubby --args="intel_iommu=on iommu=pt" --update-kernel=ALL
    ```

    ii. 重启主机。
  - 在 AMD 主机上启用 AMD-Vi：
    i. 使用 **iommu=pt** 参数重新生成 GRUB 配置：

    ```
    # grubby --args="iommu=pt" --update-kernel=ALL
    ```

    ii. 重启主机。

流程

1. **可选：** 确认您的网络设备可使用的最大 VF 数。要做到这一点，请使用以下命令，将 *eth1* 替换为您的 SR-IOV 兼容网络设备。

```
# cat /sys/class/net/eth1/device/sriov_totalvfs
7
```

2. 使用以下命令来创建虚拟功能(VF)：

```
# echo VF-number > /sys/class/net/network-interface/device/sriov_numvfs
```

在命令中，替换：

- 使用您要在其上创建 PF 的 VF 数替换 *VF-number*。
- 使用 VF 要创建的网络接口的名称替换 *network-interface*。

以下示例从 eth1 网络接口创建 2 个 VF：

```
# echo 2 > /sys/class/net/eth1/device/sriov_numvfs
```

3. 确定已添加了 VF：

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

4. 通过为您用于创建 VF 的网络接口创建一个 udev 规则，使创建的 VF 持久化。例如，对于 *eth1* 接口，创建 **/etc/udev/rules.d/eth1.rules** 文件，并添加以下行：

```
ACTION=="add", SUBSYSTEM=="net", ENV{ID_NET_DRIVER}=="ixgbe",
ATTR{device/sriov_numvfs}="2"
```

这样可确保使用 **ixgbe** 驱动程序的两个 VF 在主机启动时可自动对 **eth1** 接口可用。如果不需要持久性 SR-IOV 设备，请跳过这一步。

警告

目前，当试图在 Broadcom NetXtreme II BCM57810 适配器上保留 VF 时，上述设置无法正常工作。另外，基于这些适配器将 VF 附加到 Windows 虚拟机当前还不可靠。

5. 将新添加的 VF 接口设备热插到正在运行的虚拟机中。

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

验证

- 如果过程成功，客户机操作系统会检测新的网络接口卡。

14.7.3. SR-IOV 分配支持的设备

并非所有设备都可用于 SR-IOV。在 RHEL 9 中，以下设备已被测试并验证为与 SR-IOV 兼容。

网络设备

- Intel 82599ES 10 千兆以太网控制器 - 使用 **ixgbe** 驱动程序
- Intel 以太网控制器 XL710 系列 - 使用 **i40e** 驱动程序
- Intel 以太网络适配器 XXV710 - 使用 **i40e** 驱动程序
- Intel 82576 千兆以太网控制器 - 使用 **igb** 驱动程序
- Broadcom NetXtreme II BCM57810 - 使用 **bnx2x** 驱动程序

- QSFP 的以太网控制器 E810-C - 使用 **ice** 驱动程序
- SFC9220 10/40G 以太网控制器 - 使用 **sfc** 驱动程序
- FastLinQ QL41000 系列 10/25/40/50GbE 控制器 - 使用 **qede** 驱动程序
- Mellanox ConnectX-5 以太网适配器卡
- Mellanox MT2892 系列 [ConnectX-6 Dx]

14.8. 将 DASD 设备附加到 IBM Z 中的虚拟机

通过使用 **vfio-ccw** 功能，您可以将直接访问存储设备(DASD)作为介质设备分配给 IBM Z 主机上的虚拟机(VM)。例如，虚拟机可以访问 z/OS 数据集，或向 z/OS 机器提供分配的 DASD。

先决条件

- 您有一个 FICON 协议支持的具有 IBM Z 硬件架构的系统。
- 您有一个 Linux 操作系统的目标虚拟机。
- *driverctl* 软件包已安装。

```
# dnf install driverctl
```

- 在主机上已载入了必要的 **vfio** 内核模块。

```
# lsmod | grep vfio
```

  这个命令的输出必须包含以下模块：

  - **vfio_ccw**
  - **vfio_mdev**
  - **vfio_iommu_type1**

- 您有一个备用 DASD 设备供虚拟机独占使用，您知道设备的标识符。
  以下流程使用 **0.0.002c** 作为示例。在执行这些命令时，请使用 DASD 设备的标识符替换 **0.0.002c**。

流程

1. 获取 DASD 设备的子通道标识符。

```
# 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  f0  f0  ff   02111221 00000000
```

   在本例中，子频道标识符被检测为 **0.0.29a8**。在此流程的以下命令中，将 **0.0.29a8** 替换为检测到的设备的子通道标识符。

2. 如果上一步中的 **lscss** 命令只显示标头输出，且没有设备信息，请执行以下步骤：

a. 从 **cio_ignore** 列表中删除该设备。

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

b. 在客户机操作系统中，编辑虚拟机的内核命令行，将带有 **!** 标记的设备标识符添加到以 **cio_ignore=** 开头的行（如果它还没有存在）。

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

c. 在主机上重复第 1 步，以获取子通道标识符。

3. 将子通道绑定到 **vfio_ccw** 直通驱动程序。

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

注意

这会将 *0.0.29a8* 子通道永久绑定到 **vfio_ccw**，这意味着 DASD 在主机上将不可用。如果需要在主机上使用该设备，您必须首先删除到 'vfio_ccw'的自动绑定，并将子通道重新绑定到默认驱动程序：

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

4. 定义并启动 DASD 介质设备。

```
# cat nodedev.xml
<device>
   <parent>css_0_0_29a8</parent>
   <capability type="mdev">
      <type id="vfio_ccw-io"/>
   </capability>
</device>

# virsh nodedev-define nodedev.xml
Node device 'mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8' defined from
'nodedev.xml'

# virsh nodedev-start mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8
Device mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8 started
```

5. 如果虚拟机正在运行，请关闭虚拟机。

6. 显示之前定义的设备的 UUID，并保存它以供下一步使用。

```
# virsh nodedev-dumpxml mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8

<device>
  <name>mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8</name>
  <parent>css_0_0_29a8</parent>
  <capability type='mdev'>
    <type id='vfio_ccw-io'/>
    <uuid>30820a6f-b1a5-4503-91ca-0c10ba12345a</uuid>
    <iommuGroup number='0'/>
    <attr name='assign_adapter' value='0x02'/>
```

```
    <attr name='assign_domain' value='0x002b'/>
  </capability>
</device>
```

7. 将介质设备附加到虚拟机。为此，请使用 **virsh edit** 工具编辑虚拟机的 XML 配置，将以下部分添加到 XML 中，并将 **uuid** 值替换为您在上一步中获取的 UUID。

```
<hostdev mode='subsystem' type='mdev' model='vfio-ccw'>
  <source>
    <address uuid="30820a6f-b1a5-4503-91ca-0c10ba12345a"/>
  </source>
</hostdev>
```

8. **可选：** 将介质设备配置为在主机引导时自动启动。

```
# virsh nodedev-autostart mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8
```

验证

1. 确保正确配置了介质设备。

```
# virsh nodedev-info mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8
Name:           mdev_30820a6f_b1a5_4503_91ca_0c10ba12345a_0_0_29a8
Parent:         css_0_0_0121
Active:         yes
Persistent:     yes
Autostart:      yes
```

2. 获取 **libvirt** 分配给中介 DASD 设备的标识符。为此，可显示虚拟机的 XML 配置，并查找 **vfio-ccw** 设备。

```
# 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>
```

在本例中，分配的设备的标识符是 **0.0.0009**。

3. 启动虚拟机并登录到其客户端操作系统。

4. 在客户端操作系统中，确认 DASD 设备已被列出。例如：

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

5. 在客户端操作系统中，在线设置设备。例如：

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

其它资源

- 关于**cio_ignore** 的 IBM 文档
- 在运行时配置内核参数

14.9. 使用 WEB 控制台将 WATCHDOG 设备附加到虚拟机

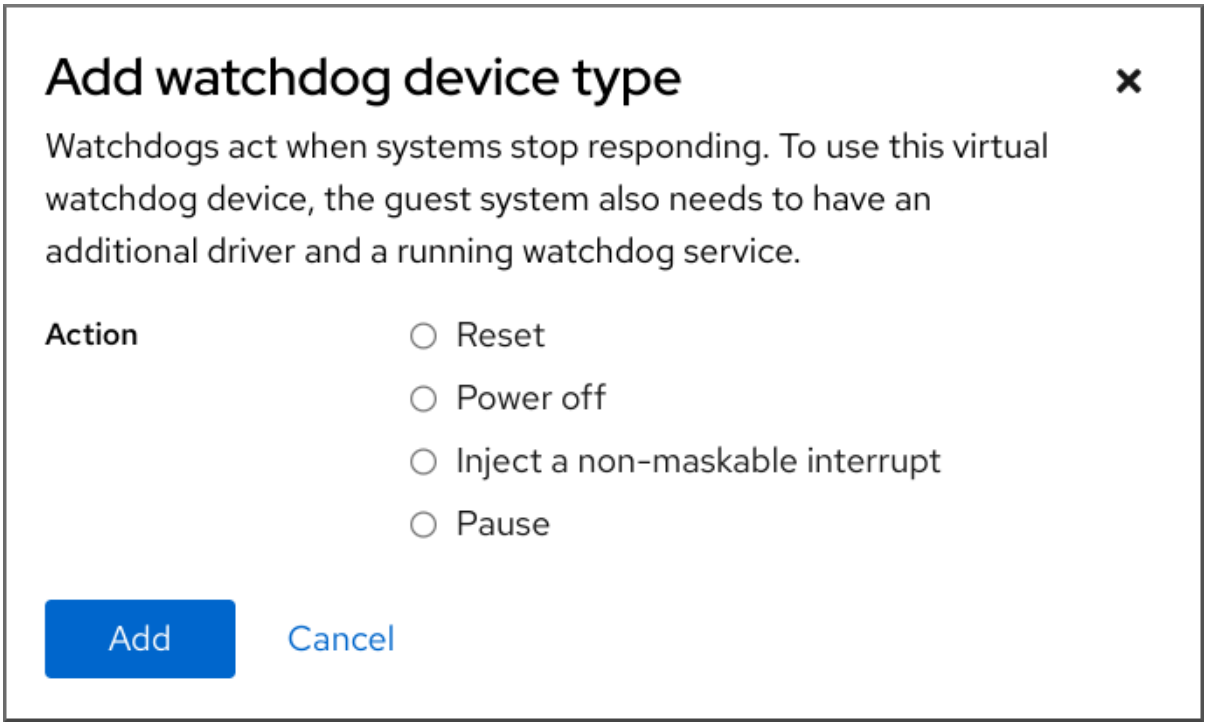
要强制虚拟机(VM)在停止响应时执行指定的操作，您可以将虚拟 watchdog 设备附加到虚拟机。

先决条件

- 您已在系统上安装了 Web 控制台虚拟机插件。如需更多信息，请参阅 第 8.2 节 “设置 web 控制台以管理虚拟机”。

流程

1. 在命令行界面中，安装 watchdog 服务。
   # dnf install watchdog
2. 关闭虚拟机。
3. 将 watchdog 服务添加到虚拟机。
   # virt-xml *vmname* --add-device --watchdog action=reset --update
4. 运行虚拟机。
5. 打开 web 控制台，并在 web 控制台的 **Virtual Machines** 界面中，点击您要添加 watchdog 设备的虚拟机。
6. 点 Overview 窗格中 **Watchdog** 字段旁边的 **add**。
   此时会出现 **Add watchdog device type** 对话框。
7. 如果虚拟机停止响应，选择您希望 watchdog 设备执行的操作。



8. 点**添加**。

验证

- 您选择的操作在 Overview 窗格中的 **Watchdog** 字段旁边可见。

14.10. 将 PCI 设备附加到 IBM Z 上的虚拟机

通过使用 **vfio-pci** 设备驱动程序，您可以在 pass-through 模式下将 PCI 设备分配给 IBM Z 主机上的虚拟机(VM)。例如，这使虚拟机可以使用 NVMe 闪存磁盘来处理数据库。

先决条件

- 您有一个具有 IBM Z 硬件架构的主机系统。
- 您有一个 Linux 操作系统的目标虚拟机。
- 在主机上已载入了必要的 **vfio** 内核模块。

```
# lsmod | grep vfio
```

这个命令的输出必须包含以下模块：

  - **vfio_pci**
  - **vfio_pci_core**
  - **vfio_iommu_type1**

流程

1. 获取您要使用的设备的 PCI 地址标识符。

```
# lspci -nkD
```

```
0000:00:00.0 0000: 1014:04ed
 Kernel driver in use: ism
 Kernel modules: ism
0001:00:00.0 0000: 1014:04ed
 Kernel driver in use: ism
 Kernel modules: ism
0002:00:00.0 0200: 15b3:1016
 Subsystem: 15b3:0062
 Kernel driver in use: mlx5_core
 Kernel modules: mlx5_core
0003:00:00.0 0200: 15b3:1016
 Subsystem: 15b3:0062
 Kernel driver in use: mlx5_core
 Kernel modules: mlx5_core
```

2. 打开您要将 PCI 设备附加到的虚拟机的 XML 配置。

```
# virsh edit vm-name
```

3. 将以下 **<hostdev>** 配置添加到 XML 文件的 **<devices>** 部分。
将 **address** 行上的值替换为设备的 PCI 地址。例如，如果设备地址是 **0003:00:00.0**，请使用以下配置：

```
<hostdev mode="subsystem" type="pci" managed="yes">
  <driver name="vfio"/>
   <source>
    <address domain="0x0003" bus="0x00" slot="0x00" function="0x0"/>
   </source>
   <address type="pci"/>
</hostdev>
```

4. **可选：** 要修改客户机操作系统如何检测 PCI 设备，您还可以向 **<address>** element 元素中添加 **<zpci>** 子元素。在 **<zpci>** 行中，您可以调整 **uid** 和 **fid** 值，它修改客户端操作系统中设备的 PCI 地址和功能 ID。

```
<hostdev mode="subsystem" type="pci" managed="yes">
  <driver name="vfio"/>
   <source>
    <address domain="0x0003" bus="0x00" slot="0x00" function="0x0"/>
   </source>
   <address type="pci">
     <zpci uid="0x0008" fid="0x001807"/>
   </address>
</hostdev>
```

在本例中：

- **uid="0x0008"** 将虚拟机中设备的域 PCI 地址设置为 **0008:00:00.0**。
- **fid="0x001807"** 将设备的插槽值设置为 **0x001807**。因此，虚拟机文件系统中的设备配置被保存到 **/sys/bus/pci/slots/00001087/address** 中。
如果没有指定这些值，**libvirt** 会自动配置它们。

5. 保存 XML 配置。

6. 如果虚拟机正在运行，请将其关闭。

```
# virsh shutdown vm-name
```

验证

1. 启动虚拟机并登录到其客户机操作系统。
2. 在客户端操作系统中，确认列出了 PCI 设备。
   例如，如果设备地址是 **0003:00:00.0**，请使用以下命令：

```
# lspci -nkD | grep 0003:00:00.0

0003:00:00.0 8086:9a09 (rev 01)
```

第 15 章 为虚拟机管理存储

虚拟机(VM)与物理机一样，需要用于数据、程序和系统文件的存储。作为虚拟机管理员，您可以将物理或基于网络的存储分配给虚拟机作为虚拟存储。您还可以修改存储是如何呈现给虚拟机的，无论底层硬件是什么。

以下小节提供有关不同类型的虚拟机存储、它们是如何工作的，以及如何使用 CLI 或 Web 控制台管理它们的信息。

15.1. 了解虚拟机存储

如果您对虚拟机(VM)存储不熟悉，或者不确定它是如何工作的，以下章节提供了有关虚拟机存储各个组件的概括介绍、它是如何工作的、管理基础知识以及红帽提供的支持的解决方案。

您可以找到以下相关信息：

- 存储池
- 存储卷
- 使用 libvirt 管理存储
- 虚拟机存储概述
- 支持和不支持的存储池类型

15.1.1. 存储池简介

存储池是由 **libvirt** 管理的文件、目录或存储设备，来为虚拟机(VM)提供存储。您可以将存储池划分为存储卷，用于存储虚拟机镜像或作为额外存储附加到虚拟机。

此外，多个虚拟机可以共享相同的存储池，从而更好地分配存储资源。

- 存储池可以是持久的或临时的：
  - 主机系统重启后，持久性存储池会保留下来。您可以使用 **virsh pool-define** 来创建持久性存储池。
  - 临时存储池仅在主机重启前存在。您可以使用 **virsh pool-create** 命令来创建临时性存储池。

存储池存储类型

存储池可以是本地的也可以基于网络的（共享）：

- **本地存储池**
  本地存储池直接附加到主机服务器。它们包括本地设备中的本地目录、直接附加磁盘、物理分区以及逻辑卷管理（LVM）卷组。

  本地存储池可用于开发、测试，以及不需要迁移或有大量虚拟机的小型部署。

- **网络（共享）存储池**
  网络存储池包括使用标准协议在网络上共享的存储设备。

15.1.2. 存储卷简介

存储池划分为 **存储卷**。存储卷是物理分区、LVM 逻辑卷、基于文件的磁盘映像，以及由 **libvirt** 处理的其他存储类型的抽象。无论底层硬件是什么，存储卷都以本地存储设备（如磁盘）的形式出现在虚拟机中。

在主机上，存储卷通过其名称以及从中派生的存储池的标识符来引用。在 **virsh** 命令行上，它采用 **--pool *storage_pool volume_name*** 的形式。

例如：要在 *guest_images* 池中显示名为 *firstimage* 的卷信息。

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

15.1.3. 使用 libvirt 进行存储管理

通过使用 **libvirt** 远程协议，您可以管理虚拟机存储的所有方面。这些操作也可以在远程主机上执行。因此，使用 **libvirt** 的管理应用程序（如 RHEL web 控制台）可用来执行配置虚拟机存储的所有所需的任务。

您可以使用 **libvirt** API 来查询存储池中卷的列表，或者获取有关该存储池中容量、分配和可用存储的信息。对于支持它的存储池，您还可以使用 **libvirt** API 来创建、克隆、调整大小和删除存储卷。另外，您可以使用 **libvirt** API 来上传数据到存储卷，从存储卷下载数据，或者从存储卷中删除数据。

15.1.4. 存储管理概述

为了说明用于管理存储的可用选项，以下示例介绍了使用 **mount -t nfs nfs.example.com:/path/to/share /path/to/data** 的 NFS 服务器的一个示例。

作为存储管理员：

- 您可以在虚拟化主机上定义 NFS 存储池来描述导出的服务器路径和客户端目标路径。因此，**libvirt** 可以在 **libvirt** 启动时自动挂载存储，或者在 **libvirt** 运行时根据需要自动挂载存储。
- 您可以根据名称简单地将存储池和存储卷添加到虚拟机。您不需要添加卷的目标路径。因此，即使目标客户端路径有变化，也不会影响虚拟机。
- 您可以将存储池配置为自动启动。执行此操作时，**libvirt** 会在**libvirt** 启动时自动将 NFS 共享磁盘挂载到指定的目录上。**libvirt** 将共享挂载到指定的目录上，类似于命令 **mount nfs.example.com:/path/to/share /vmdata**。
- 您可以使用 **libvirt** API 查询存储卷路径。这些存储卷基本上是 NFS 共享磁盘中的文件。然后，您可以将这些路径复制到虚拟机 XML 定义中的 部分，该定义描述了虚拟机块设备的源存储。
- 对于 NFS，您可以使用一个使用 **libvirt** API 的应用程序来在存储池中创建和删除存储卷（NFS 共享中的文件），不超过池大小的限制（共享的存储容量）。
  请注意，并非所有存储池类型都支持创建和删除卷。
- 当不再需要时，您可以停止存储池。停止存储池(**pool-destroy**) 可撤销启动操作，在这种情况下，即卸载 NFS 共享。销毁操作不会修改共享中的数据，即使该命令的名称看似象要删除。如需更多信息，请参阅 **man virsh**。

15.1.5. 支持和不支持的存储池类型

支持的存储池类型

以下是 RHEL 支持的存储池类型列表：

- 基于目录的存储池
- 基于磁盘的存储池
- 基于分区的存储池
- 基于 iSCSI 的存储池
- 基于 LVM 的存储池
- 基于 NFS 的存储池
- 使用 vHBA 设备基于 SCSI 的存储池
- 基于多路径的存储池
- 基于 RBD 的存储池

不支持的存储池类型

以下是 RHEL 不支持的 **libvirt** 存储池类型的列表：

- 基于 Sheepdog 的存储池
- 基于 Vstorage 的存储池
- 基于 ZFS 的存储池
- iscsi-direct 存储池
- glusterfs 存储池

15.2. 使用 CLI 管理虚拟机存储池

您可以使用 CLI 管理存储池的以下方面来为虚拟机分配存储：

- 查看存储池信息
- 创建存储池
  - 使用 CLI 创建基于目录的存储池
  - 使用 CLI 创建基于磁盘的存储池
  - 使用 CLI 创建基于文件系统的存储池
  - 使用 CLI 创建基于 iSCSI 的存储池
  - 使用 CLI 创建基于 LVM 的存储池
  - 使用 CLI 创建基于 NFS 的存储池
  - 使用 CLI 创建带有 vHBA 设备的基于 SCSI 的存储池
- 删除存储池

15.2.1. 使用 CLI 查看存储池信息

通过使用 CLI，您可以查看所有存储池的列表，其中包含有关存储池的有限或完整的详情。您还可以过滤列出的存储池。

流程

- 使用 **virsh pool-list** 命令来查看存储池信息。

```
# virsh pool-list --all --details
 Name                 State    Autostart  Persistent    Capacity  Allocation   Available
 default              running  yes        yes         48.97 GiB   23.93 GiB   25.03 GiB
 Downloads            running  yes        yes        175.62 GiB   62.02 GiB  113.60 GiB
 RHEL-Storage-Pool    running  yes        yes        214.62 GiB   93.02 GiB  168.60 GiB
```

其它资源

- **virsh pool-list --help** 命令

15.2.2. 使用 CLI 创建基于目录的存储池

基于目录的存储池是基于现有挂载的文件系统中的目录。例如，当您想要将文件系统上的剩余空间用于其他用途时，这非常有用。您可以使用 **virsh** 工具来创建基于目录的存储池。

先决条件

- 确定您的管理程序支持目录存储池：

```
# virsh pool-capabilities | grep "'dir' supported='yes'"
```

如果命令显示任何输出结果，则代表支持目录池。

流程

1. **创建存储池**
   使用 **virsh pool-define-as** 命令来定义和创建目录类型的存储池。例如，要创建使用 **/guest_images** 目录的名为 **guest_images_dir** 的存储池：

```
# virsh pool-define-as guest_images_dir dir --target "/guest_images"
Pool guest_images_dir defined
```

   如果您已经有要创建的存储池的 XML 配置，也可以根据 XML 定义池。详情请参阅基于目录的存储池参数。

2. **创建存储池目标路径**
   使用 **virsh pool-build** 命令，来为事先格式化的文件系统存储池创建存储池目标路径，初始化存储源设备，并定义数据格式。

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

# ls -la /guest_images
```

```
total 8
drwx------.  2 root root 4096 May 31 19:38 .
dr-xr-xr-x. 25 root root 4096 May 31 19:38 ..
```

3. **验证是否已创建池**
   使用 **virsh pool-list** 命令来验证池是否已被创建。

```
# virsh pool-list --all

 Name                 State      Autostart
-------------------------------------------
 default              active     yes
 guest_images_dir     inactive   no
```

4. **启动存储池**
   使用 **virsh pool-start** 命令来挂载存储池。

```
# virsh pool-start guest_images_dir
 Pool guest_images_dir started
```

注意

virsh pool-start 命令仅适用于持久性存储池。临时存储池创建时会自动启动。

5. **[可选]自动启动过程**
   默认情况下，使用 **virsh** 命令定义的存储池不会被设置为在每次虚拟化服务启动时自动启动。使用 **virsh pool-autostart** 命令将存储池配置为自动启动。

```
# virsh pool-autostart guest_images_dir
 Pool guest_images_dir marked as autostarted
```

验证

- 使用 **virsh pool-info** 命令来验证存储池是否处于 ***running*** 状态。检查报告的大小是否与预期一样，以及是否正确配置了自动启动。

```
# virsh pool-info guest_images_dir
 Name:           guest_images_dir
 UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
 State:          running
 Persistent:     yes
 Autostart:      yes
 Capacity:       458.39 GB
 Allocation:     197.91 MB
 Available:      458.20 GB
```

15.2.3. 使用 CLI 创建基于磁盘的存储池

在基于磁盘的存储池中，池是基于磁盘分区的。例如，当您希望整个磁盘分区专门用于虚拟机(VM)存储时，这非常有用。您可以使用 **virsh** 工具来创建基于磁盘的存储池。

先决条件

- 确定您的管理程序支持基于磁盘的存储池：

```
# virsh pool-capabilities | grep "'disk' supported='yes'"
```

如果命令显示任何输出结果，则代表支持基于磁盘的池。

- 准备一个设备，以此作为存储池的基础。因此，最好使用分区（例如 **/dev/sdb1**）或 LVM 卷。如果您提供了一个虚拟机，对整个磁盘或块设备（例如：**/dev/sdb**）具有写权限，则虚拟机可以对其进行分区，或者在其上创建自己的 LVM 组。这可能会导致主机上的系统错误。
但是，如果您需要为存储池使用整个块设备，红帽建议保护设备上的任何重要分区不受 **os-prober** 功能的影响。要做到这一点，编辑 **/etc/default/grub** 文件，并应用以下配置之一：

  - 禁用 **os-prober**。

```
GRUB_DISABLE_OS_PROBER=true
```

  - 防止 **os-prober** 发现特定的分区。例如：

```
GRUB_OS_PROBER_SKIP_LIST="5ef6313a-257c-4d43@/dev/sdb1"
```

- 在创建存储池前，备份所选存储设备上的任何数据。根据所使用的 **libvirt** 版本，给存储池指定专用的磁盘可能会重新格式化并擦除当前存储在磁盘设备上的所有数据。

流程

1. **创建存储池**
使用 **virsh pool-define-as** 命令来定义和创建磁盘类型的存储池。以下示例创建一个名为 **guest_images_disk** 的存储池，它使用 **/dev/sdb** 设备，并挂载在 /dev 目录。

```
# virsh pool-define-as guest_images_disk disk --source-format=gpt --source-dev=/dev/sdb --
target /dev
Pool guest_images_disk defined
```

如果您已经有要创建的存储池的 XML 配置，也可以根据 XML 定义池。详情请参阅基于磁盘的存储池参数。

2. **创建存储池目标路径**
使用 **virsh pool-build** 命令，为事先格式化的文件系统存储池创建存储池目标路径、初始化存储源设备，并定义数据格式。

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

注意

只有基于磁盘、基于文件系统和逻辑存储池才需要构建目标路径。如果 **libvirt** 检测到源存储设备的数据格式与所选存储池类型不同，则构建会失败，除非指定了 ***overwrite*** 选项。

3. **验证是否已创建池**
使用 **virsh pool-list** 命令来验证池是否已被创建。

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

```
Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images_disk    inactive   no
```

4. **启动存储池**
使用 **virsh pool-start** 命令来挂载存储池。

```
# virsh pool-start guest_images_disk
  Pool guest_images_disk started
```

注意

virsh pool-start 命令仅适用于持久性存储池。临时存储池创建时会自动启动。

5. **[可选]自动启动过程**
默认情况下，使用 **virsh** 命令定义的存储池不会被设置为在每次虚拟化服务启动时自动启动。使用 **virsh pool-autostart** 命令将存储池配置为自动启动。

```
# virsh pool-autostart guest_images_disk
  Pool guest_images_disk marked as autostarted
```

验证

- 使用 **virsh pool-info** 命令来验证存储池是否处于 ***running*** 状态。检查报告的大小是否与预期一样，以及是否正确配置了自动启动。

```
# virsh pool-info guest_images_disk
  Name:           guest_images_disk
  UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
  State:          running
  Persistent:     yes
  Autostart:      yes
  Capacity:       458.39 GB
  Allocation:     197.91 MB
  Available:      458.20 GB
```

15.2.4. 使用 CLI 创建基于文件系统的存储池

当您要在未挂载的文件系统上创建存储池时，请使用基于文件系统的存储池。此存储池基于给定的文件系统挂载点。您可以使用 **virsh** 工具来创建基于文件系统的存储池。

先决条件

- 确保您的管理程序支持基于文件系统的存储池：

```
# virsh pool-capabilities | grep "'fs' supported='yes'"
```

如果该命令显示任何输出结果，则代表支持基于文件的池。

准备一个设备，以此作为存储池的基础。同此，最好使用分区（例如 /dev/sdb1），或 LVM 卷。如

- 准备一个设备，以此作为存储池的基础。因此，最好使用分区（例如 **/dev/sdb1**）或 LVM 卷。如果您提供了一个虚拟机，对整个磁盘或块设备（例如：**/dev/sdb**）具有写权限，则虚拟机可以对其进行分区，或者在其上创建自己的 LVM 组。这可能会导致主机上的系统错误。
  但是，如果您需要为存储池使用整个块设备，红帽建议保护设备上的任何重要分区不受 **os-prober** 功能的影响。要做到这一点，编辑 **/etc/default/grub** 文件，并应用以下配置之一：

  - 禁用 **os-prober**。

    ```
    GRUB_DISABLE_OS_PROBER=true
    ```

  - 防止 **os-prober** 发现特定的分区。例如：

    ```
    GRUB_OS_PROBER_SKIP_LIST="5ef6313a-257c-4d43@/dev/sdb1"
    ```

流程

1. **创建存储池**
   使用 **virsh pool-define-as** 命令定义和创建文件系统类型的存储池。例如，要创建一个名为 **guest_images_fs** 的存储池，它使用 **/dev/sdc1** 分区，并挂载在 /guest_images 目录上：

   ```
   # virsh pool-define-as guest_images_fs fs --source-dev /dev/sdc1 --target /guest_images
   Pool guest_images_fs defined
   ```

   如果您已经有要创建的存储池的 XML 配置，也可以根据 XML 定义池。详情请参阅基于文件系统的存储池参数。

2. **定义存储池目标路径**
   使用 **virsh pool-build** 命令，为事先格式化的文件系统存储池创建存储池目标路径、初始化存储源设备，并定义数据格式。

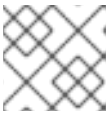
   ```
   # virsh pool-build guest_images_fs
    Pool guest_images_fs built

   # ls -la /guest_images
    total 8
    drwx------.  2 root root 4096 May 31 19:38 .
    dr-xr-xr-x. 25 root root 4096 May 31 19:38 ..
   ```

3. **验证是否已创建池**
   使用 **virsh pool-list** 命令来验证池是否已被创建。

   ```
   # virsh pool-list --all

    Name                 State      Autostart
    -------------------------------------------
    default              active     yes
    guest_images_fs      inactive   no
   ```

4. **启动存储池**
   使用 **virsh pool-start** 命令来挂载存储池。

   ```
   # virsh pool-start guest_images_fs
    Pool guest_images_fs started
   ```



注意

virsh pool-start 命令仅适用于持久性存储池。临时存储池创建时会自动启动。

5. **可选：** 打开自动启动
默认情况下，使用 **virsh** 命令定义的存储池不会被设置为在每次虚拟化服务启动时自动启动。使用 **virsh pool-autostart** 命令将存储池配置为自动启动。

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

验证

1. 使用 **virsh pool-info** 命令来验证存储池是否处于 ***running*** 状态。检查报告的大小是否与预期一样，以及是否正确配置了自动启动。

```
# virsh pool-info guest_images_fs
  Name:           guest_images_fs
  UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
  State:          running
  Persistent:     yes
  Autostart:      yes
  Capacity:       458.39 GB
  Allocation:     197.91 MB
  Available:      458.20 GB
```

2. 验证文件系统的目标路径中存在 **lost+found** 目录，这表示挂载该设备。

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

# ls -la /guest_images
  total 24
  drwxr-xr-x.  3 root root  4096 May 31 19:47 .
  dr-xr-xr-x. 25 root root  4096 May 31 19:38 ..
  drwx------.  2 root root 16384 May 31 14:18 lost+found
```

15.2.5. 使用 CLI 创建基于 iSCSI 的存储池

互联网小型计算机系统接口(iSCSI)是基于 IP 的存储网络标准，用于连接数据存储设施。如果要在 iSCSI 服务器上拥有一个存储池，您可以使用 **virsh** 工具来创建基于 iSCSI 的存储池。

先决条件

- 确定您的管理程序支持基于 iSCSI 的存储池：

```
# virsh pool-capabilities | grep "'iscsi' supported='yes'"
```

如果该命令显示任何输出结果，则代表支持基于 iSCSI 的池。

流程

1. **创建存储池**

使用 **virsh pool-define-as** 命令来定义和创建 iSCSI 类型的存储池。例如，要创建一个名为 **guest_images_iscsi** 的存储池，它使用 **server1.example.com** 上的 **iqn.2010-05.com.example.server1:iscsirhel7guest** IQN，并挂载在 **/dev/disk/by-path** 路径上：

```
# virsh pool-define-as --name guest_images_iscsi --type iscsi --source-host
server1.example.com --source-dev iqn.2010-05.com.example.server1:iscsirhel7guest --
target /dev/disk/by-path
Pool guest_images_iscsi defined
```

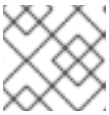
如果您已经有要创建的存储池的 XML 配置，也可以根据 XML 定义池。详情请查看基于 iSCSI 的存储池参数。

2. **验证是否已创建池**
使用 **virsh pool-list** 命令来验证池是否已被创建。

```
# virsh pool-list --all

 Name               State      Autostart
 -----------------------------------------
 default            active     yes
 guest_images_iscsi   inactive   no
```

3. **启动存储池**
使用 **virsh pool-start** 命令来挂载存储池。

```
# virsh pool-start guest_images_iscsi
 Pool guest_images_iscsi started
```



注意

virsh pool-start 命令仅适用于持久性存储池。临时存储池创建时会自动启动。

4. **[可选]自动启动过程**
默认情况下，使用 **virsh** 命令定义的存储池不会被设置为在每次虚拟化服务启动时自动启动。使用 **virsh pool-autostart** 命令将存储池配置为自动启动。

```
# virsh pool-autostart guest_images_iscsi
 Pool guest_images_iscsi marked as autostarted
```

验证

- 使用 **virsh pool-info** 命令来验证存储池是否处于 ***running*** 状态。检查报告的大小是否与预期一样，以及是否正确配置了自动启动。

```
# virsh pool-info guest_images_iscsi
 Name:           guest_images_iscsi
 UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
 State:          running
 Persistent:     yes
 Autostart:      yes
 Capacity:       458.39 GB
 Allocation:     197.91 MB
 Available:      458.20 GB
```

15.2.6. 使用 CLI 创建基于 LVM 的存储池

如果要拥有一个存储池，其是 LVM 卷组的一部分，您可以使用 **virsh** 工具来创建基于 LVM 的存储池。

建议

在创建基于 LVM 的存储池前请注意以下几点：

- 基于 LVM 的存储池不能为 LVM 提供完整的灵活性。
- **libvirt** 支持精简逻辑卷，但不提供精简存储池的功能。
- 基于 LVM 的存储池是卷组。您可以使用 **virsh** 工具创建卷组，但使用这种方法，您在创建的卷组中只能有一个设备。要创建具有多个设备的卷组，请使用 LVM 工具，请参阅 如何在 Linux 中使用 LVM 创建卷组。
  有关卷组的详情，请参考 *Red Hat Enterprise Linux Logical Volume Manager Administration Guide*。
- 基于 LVM 的存储池需要一个完整磁盘分区。如果您使用 **virsh** 命令激活一个新分区或设备，分区将被格式化，所有数据都被清除。在这些过程中，如果您使用主机的现有卷组，则不会删除任何内容。

先决条件

- 确定您的管理程序支持基于 LVM 的存储池：

```
# virsh pool-capabilities | grep "'logical' supported='yes'"
```

  如果命令显示任何输出结果，则支持基于 LVM 的池。

流程

1. **创建存储池**
   使用 **virsh pool-define-as** 命令来定义和创建 LVM 类型的存储池。例如，以下命令创建名为 **guest_images_lvm** 的存储池，该池使用 **lvm_vg** 卷组，并挂载在 **/dev/lvm_vg** 目录上：

```
# virsh pool-define-as guest_images_lvm logical --source-name lvm_vg --target
/dev/lvm_vg
Pool guest_images_lvm defined
```

   如果您已经有要创建的存储池的 XML 配置，也可以根据 XML 定义池。详情请查看基于 LVM 的存储池参数。

2. **验证是否已创建池**
   使用 **virsh pool-list** 命令来验证池是否已被创建。

```
# virsh pool-list --all

 Name                 State      Autostart
-------------------------------------------
 default              active     yes
 guest_images_lvm     inactive   no
```

3. **启动存储池**
使用 **virsh pool-start** 命令来挂载存储池。

```
# virsh pool-start guest_images_lvm
 Pool guest_images_lvm started
```

注意

virsh pool-start 命令仅适用于持久性存储池。临时存储池创建时会自动启动。

4. **[可选]自动启动过程**
默认情况下，使用 **virsh** 命令定义的存储池不会被设置为在每次虚拟化服务启动时自动启动。使用 **virsh pool-autostart** 命令将存储池配置为自动启动。

```
# virsh pool-autostart guest_images_lvm
 Pool guest_images_lvm marked as autostarted
```

验证

- 使用 **virsh pool-info** 命令来验证存储池是否处于 ***running*** 状态。检查报告的大小是否与预期一样，以及是否正确配置了自动启动。

```
# virsh pool-info guest_images_lvm
 Name:           guest_images_lvm
 UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
 State:          running
 Persistent:     yes
 Autostart:      yes
 Capacity:       458.39 GB
 Allocation:     197.91 MB
 Available:      458.20 GB
```

15.2.7. 使用 CLI 创建基于 NFS 的存储池

如果要在网络文件系统(NFS)服务器上拥有一个存储池，您可以使用 **virsh** 工具来创建基于 NFS 的存储池。

先决条件

- 确定您的管理程序支持基于 NFS 的存储池：

```
# virsh pool-capabilities | grep "<value>nfs</value>"
```

如果该命令显示任何输出结果，则代表支持基于 NFS 的池。

流程

1. **创建存储池**
使用 virsh **pool-define-as** 命令来定义和创建 NFS 类型的存储池。例如，要创建一个名为 **guest_images_netfs** 的存储池，它使用 IP 为 **111.222.111.222** 的 NFS 服务器，使用目标目录 **/var/lib/libvirt/images/nfspool** 挂载在服务器目录 **/home/net_mount** 上：

```
# virsh pool-define-as --name guest_images_netfs --type netfs --source-
host='111.222.111.222' --source-path='/home/net_mount' --source-format='nfs' --
target='/var/lib/libvirt/images/nfspool'
```

如果您已经有要创建的存储池的 XML 配置，也可以根据 XML 定义池。详情请查看基于 NFS 的存储池参数。

2. **验证是否已创建池**
使用 **virsh pool-list** 命令来验证池是否已被创建。

```
# virsh pool-list --all

 Name                 State      Autostart
-----------------------------------------
 default              active     yes
 guest_images_netfs   inactive   no
```

3. **启动存储池**
使用 **virsh pool-start** 命令来挂载存储池。

```
# virsh pool-start guest_images_netfs
 Pool guest_images_netfs started
```

注意

virsh pool-start 命令仅适用于持久性存储池。临时存储池创建时会自动启动。

4. **[可选]自动启动过程**
默认情况下，使用 **virsh** 命令定义的存储池不会被设置为在每次虚拟化服务启动时自动启动。使用 **virsh pool-autostart** 命令将存储池配置为自动启动。

```
# virsh pool-autostart guest_images_netfs
 Pool guest_images_netfs marked as autostarted
```

验证

- 使用 **virsh pool-info** 命令来验证存储池是否处于 ***running*** 状态。检查报告的大小是否与预期一样，以及是否正确配置了自动启动。

```
# virsh pool-info guest_images_netfs
 Name:           guest_images_netfs
 UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
 State:          running
 Persistent:     yes
 Autostart:      yes
 Capacity:       458.39 GB
 Allocation:     197.91 MB
 Available:      458.20 GB
```

15.2.8. 使用 CLI 创建带有 vHBA 设备的基于 SCSI 的存储池

如果要在小型计算机系统接口(SCSI)设备上有一个存储池，您的主机必须能够使用虚拟主机总线适配器(vHBA)连接到 SCSI 设备。然后，您可以使用 **virsh** 工具来创建基于 SCSI 的存储池。

先决条件

- 确定您的管理程序支持基于 SCSI 的存储池：

```
# virsh pool-capabilities | grep "'scsi' supported='yes'"
```

如果该命令显示任何输出结果，则代表支持基于 SCSI 的池。

- 在使用 vHBA 设备创建基于 SCSI 的存储池前，先创建一个 vHBA。如需更多信息，请参阅 创建 vHBA。

流程

1. **创建存储池**
使用 **virsh pool-define-as** 命令定义并创建使用 vHBA 的 SCSI 存储池。例如，以下命令会创建一个名为 **guest_images_vhba** 的存储池，它使用由 **scsi_host3** 父适配器识别的 vHBA、全球范围的端口号 **5001a4ace3ee047d**，以及全球范围的节点号 **5001a4a93526d0a1**。存储池挂载在 **/dev/disk/** 目录上：

```
# virsh pool-define-as guest_images_vhba scsi --adapter-parent scsi_host3 --adapter-wwnn 5001a4a93526d0a1 --adapter-wwpn 5001a4ace3ee047d --target /dev/disk/
Pool guest_images_vhba defined
```

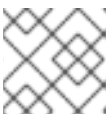
如果您已经有要创建的存储池的 XML 配置，也可以根据 XML 定义池。详情请查看使用 vHBA 设备 的基于 SCSI 的存储池的参数。

2. **验证是否已创建池**
使用 **virsh pool-list** 命令来验证池是否已被创建。

```
# virsh pool-list --all

 Name                 State      Autostart
-------------------------------------------
 default              active     yes
 guest_images_vhba    inactive   no
```

3. **启动存储池**
使用 **virsh pool-start** 命令来挂载存储池。

```
# virsh pool-start guest_images_vhba
 Pool guest_images_vhba started
```



注意

virsh pool-start 命令仅适用于持久性存储池。临时存储池创建时会自动启动。

4. **[可选]自动启动过程**
默认情况下，使用 **virsh** 命令定义的存储池不会被设置为在每次虚拟化服务启动时自动启动。使用 **virsh pool-autostart** 命令将存储池配置为自动启动。

```
# virsh pool-autostart guest_images_vhba
  Pool guest_images_vhba marked as autostarted
```

验证

- 使用 **virsh pool-info** 命令来验证存储池是否处于 ***running*** 状态。检查报告的大小是否与预期一样，以及是否正确配置了自动启动。

```
# virsh pool-info guest_images_vhba
 Name:           guest_images_vhba
 UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
 State:          running
 Persistent:     yes
 Autostart:      yes
 Capacity:       458.39 GB
 Allocation:     197.91 MB
 Available:      458.20 GB
```

15.2.9. 使用 CLI 删除存储池

要从主机系统中删除存储池，您必须停止池，并删除其 XML 定义。

流程

1. 使用 **virsh pool-list** 命令列出定义的存储池。

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
Downloads            active     yes
RHEL-Storage-Pool    active     yes
```

2. 使用 **virsh pool-destroy** 命令停止您要删除的存储池。

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

3. **可选** ：对于某些类型的存储池，您可以使用 **virsh pool-delete** 命令删除存储池所在的目录。请注意，为此，目录必须为空。

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

4. 使用 **virsh pool-undefine** 命令删除存储池的定义。

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

验证

- 确认删除了存储池。

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

15.3. 使用 WEB 控制台管理虚拟机存储池

通过使用 RHEL web 控制台，您可以管理存储池，来将存储分配给虚拟机。

您可以使用 Web 控制台：

- 查看存储池信息。
- 创建存储池：
  - 创建基于目录的存储池。
  - 创建基于 NFS 的存储池。
  - 创建基于 iSCSI 的存储池。
  - 创建基于 LVM 的存储池。
- 删除存储池。
- 取消激活存储池。

15.3.1. 使用 Web 控制台查看存储池信息

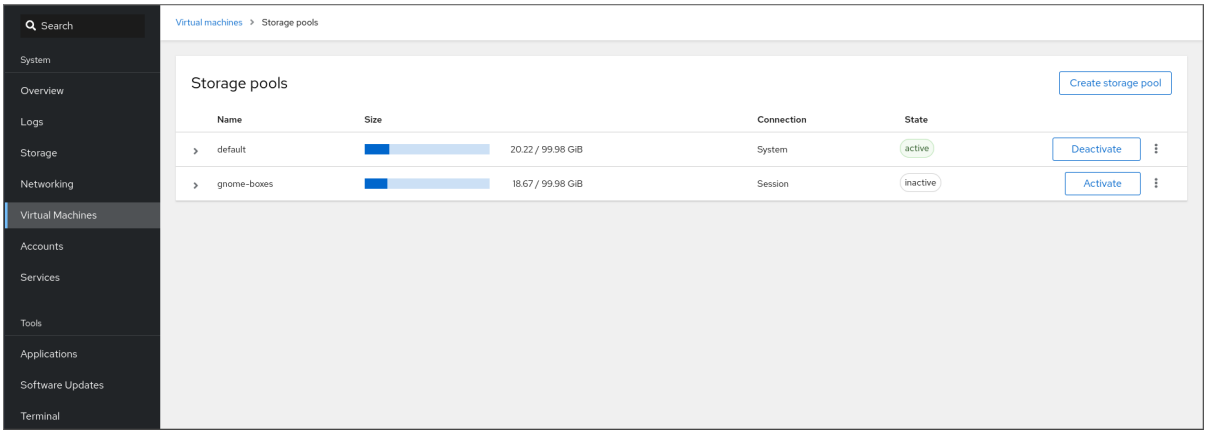
通过使用 Web 控制台，您可以查看系统上可用存储池的详细信息。存储池可用于为您的虚拟机创建磁盘映像。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 点 **Virtual Machines** 接口顶部的 **Storage Pools**。
   此时会出现**存储池**窗口，显示配置的存储池列表。



该信息包括：

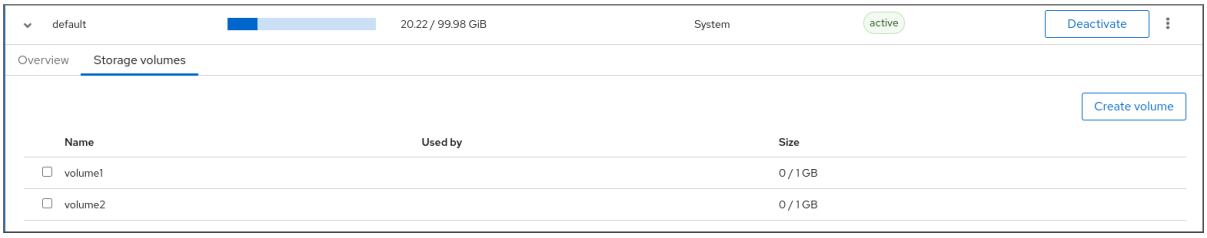
- **Name** - 存储池的名称。
- **Size** - 当前分配和存储池的总容量。
- **connection** - 用于访问存储池的连接。
- **State** - 存储池的状态。

2. 点击您要查看其信息的存储池旁的箭头。
行会展开，以显示概述窗格，其中含有所选存储池的详细信息。

该信息包括：

- **Target path** - 存储池的位置。
- **Persistent** - 表示存储池是否具有持久性配置。
- **Autostart** - 表示在系统启动时存储池是否自动启动。
- **类型** - 存储池的类型。

3. 要查看与存储池相关联的存储卷的列表，请单击 **Storage Volumes**。
这时将出现存储卷窗格，显示已配置的存储卷的列表。



该信息包括：

- **Name** - 存储卷的名称。
- **Used by** - 当前使用存储卷的虚拟机。
- **Size** - 卷的大小。

其它资源

- 使用 web 控制台查看虚拟机信息

15.3.2. 使用 Web 控制台创建基于目录的存储池

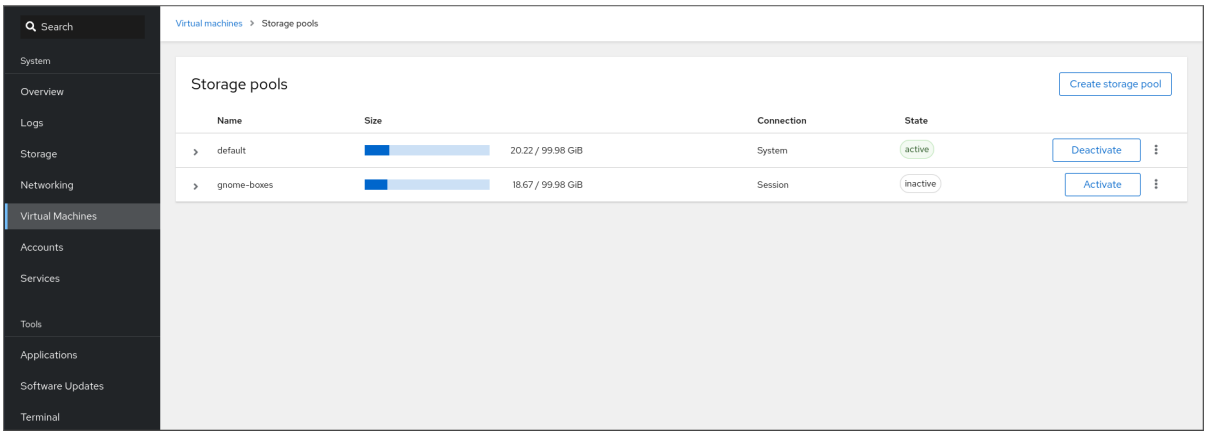
基于目录的存储池是基于现有挂载的文件系统中的目录。例如，当您想要将文件系统上的剩余空间用于其他用途时，这非常有用。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

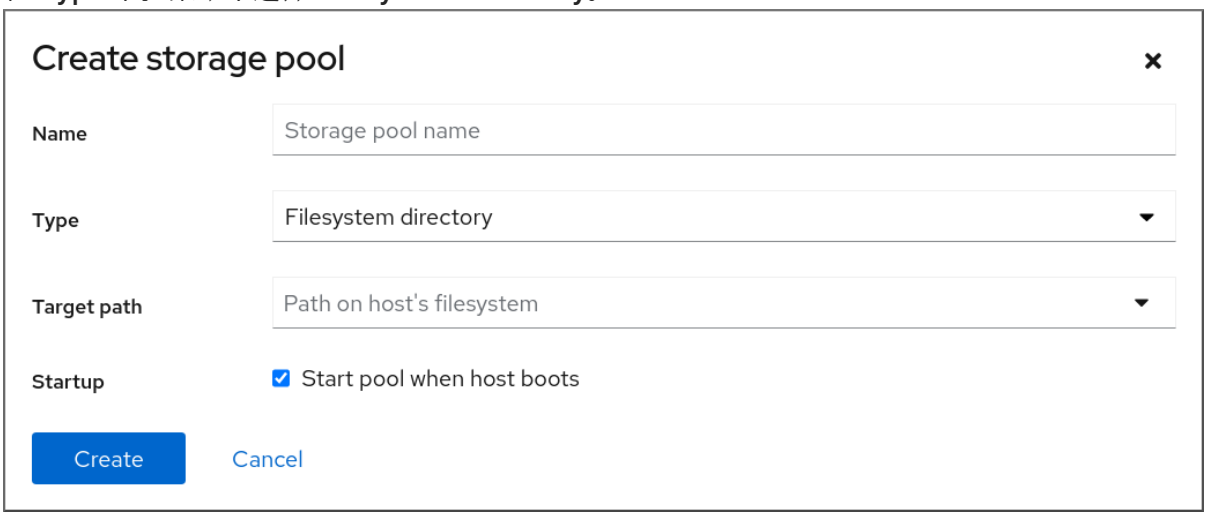
1. 在 RHEL web 控制台中，点 **Virtual Machines** 选项卡中的 **Storage pool**。
此时会出现 **存储池** 窗口，显示配置的存储池列表（若有）。



2. 单击 **Create storage pool**。
此时会出现 **Create 存储池** 对话框。

3. 输入存储池的名称。

4. 在 **Type** 下拉菜单中选择 **Filesystem directory**。



注意

如果您没有在下拉菜单中选择 **Filesystem 目录** 选项，则您的管理程序不支持基于目录的存储池。

5. 输入以下信息：

- **Target path** - 存储池的位置。
- **启动」** - 主机引导时是否启动存储池。

6. 点击 **Create**。

创建存储池时，**Create Storage Pool**对话框将关闭，新的存储池会出现在存储池列表中。

其它资源

- 了解存储池
- 使用 Web 控制台查看存储池信息

15.3.3. 使用 Web 控制台创建基于 NFS 的存储池

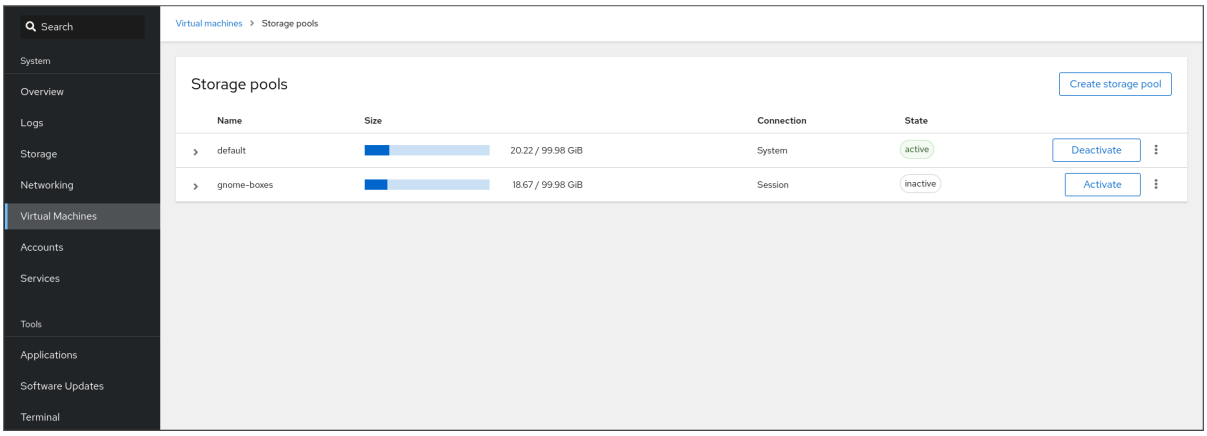
基于 NFS 的存储池是基于服务器上托管的文件系统。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

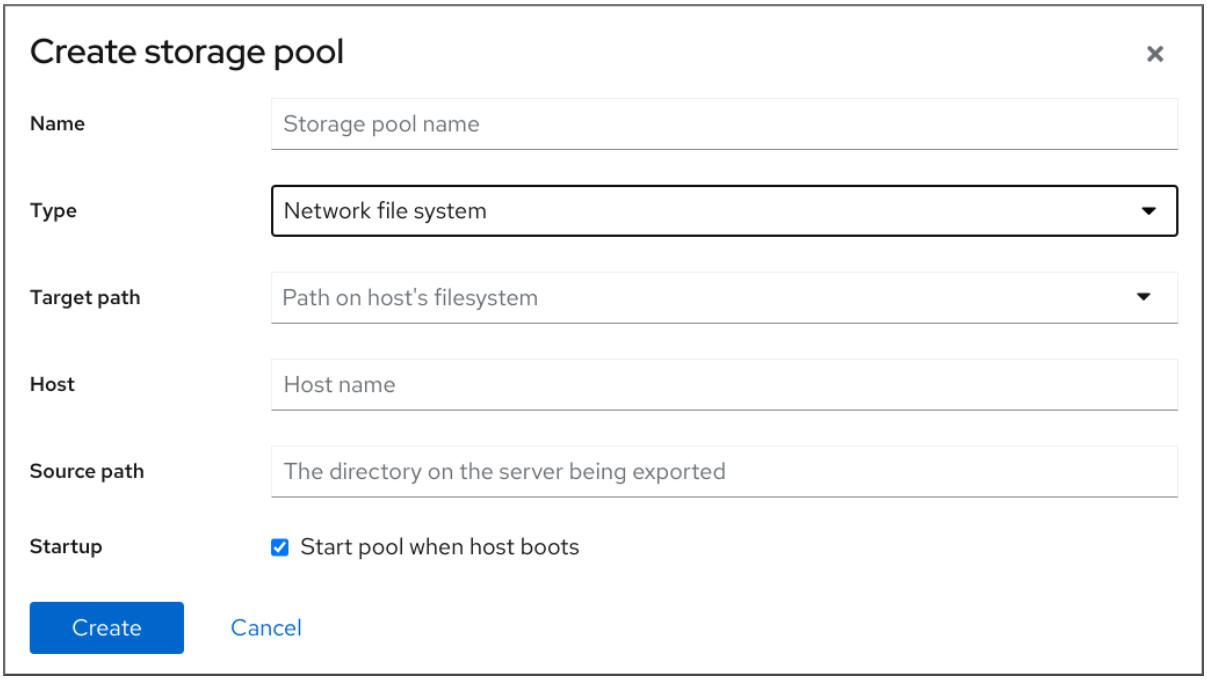
1. 在 RHEL web 控制台中，点 **Virtual Machines** 选项卡中的 **Storage pool**。
此时会出现 **存储池** 窗口，显示配置的存储池列表（若有）。



2. 单击 **Create storage pool**。
此时会出现 **Create 存储池** 对话框。

3. 输入存储池的名称。

4. 在 **Type** 下拉菜单中选择 **Network File System**。



注意

如果您在下拉菜单中选择 **网络文件系统** 选项，则您的管理程序不支持基于 NFS 的存储池。

5. 输入其他信息：
   - **目标路径** - 指定目标的路径。这将是用于存储池的路径。
   - **主机** - 挂载点所在的网络服务器的主机名。这可以是主机名或 IP 地址。
   - **源路径** - 网络服务器中使用的目录。
   - 启动」 - 主机引导时是否启动存储池。
6. 点击 **Create**。
   已创建存储池。这会关闭 **Create storage pool** 对话框，新的存储池会出现在存储池列表中。

其它资源

- 了解存储池
- 使用 Web 控制台查看存储池信息

15.3.4. 使用 Web 控制台创建基于 iSCSI 的存储池

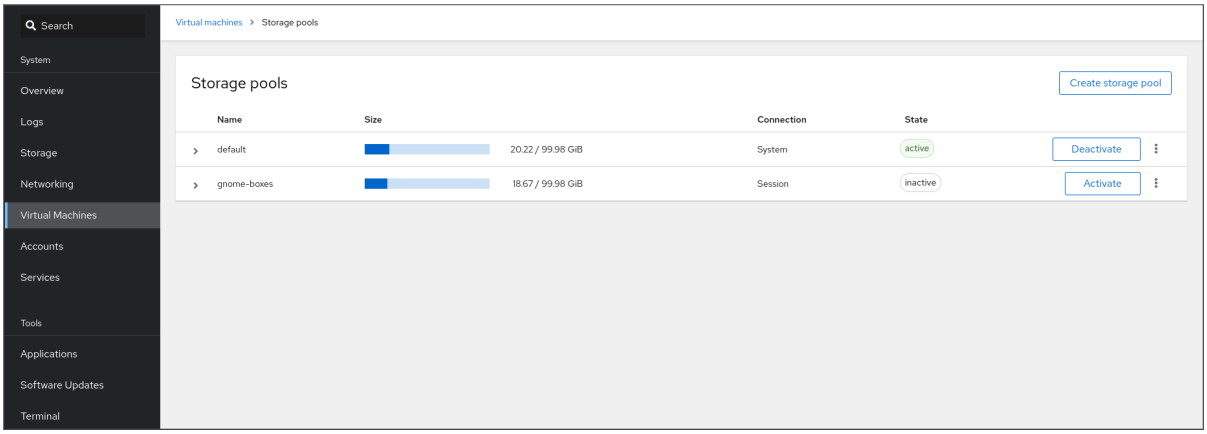
基于 iSCSI 的存储池是基于互联网小型计算机系统接口(iSCSI)，这是一种基于 IP 的存储网络标准，用于连接数据存储设施。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

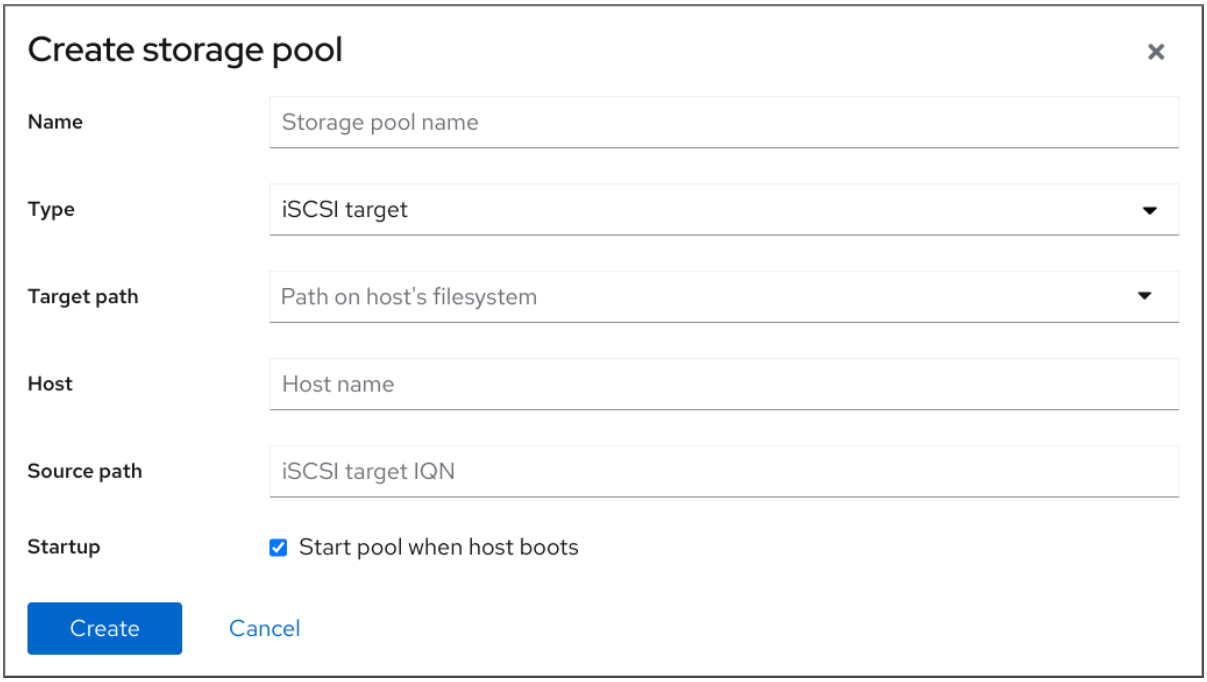
1. 在 RHEL web 控制台中，点 **Virtual Machines** 选项卡中的 **Storage pool**。
此时会出现 **存储池** 窗口，显示配置的存储池列表（若有）。



2. 单击 **Create storage pool**。
此时会出现 **Create 存储池** 对话框。

3. 输入存储池的名称。

4. 在 **Type** 下拉菜单中选择 **iSCSI 目标**。



5. 输入其他信息：

   - **目标路径** - 指定目标的路径。这将是用于存储池的路径。
   - **主机** - ISCSI 服务器的主机名或 IP 地址。
   - **源路径** - iSCSI 目标的唯一 iSCSI 限定名称(IQN)。
   - 启动」 - 主机引导时是否启动存储池。

6. 点击 **Create**。
已创建存储池。这会关闭 **Create storage pool** 对话框，新的存储池会出现在存储池列表中。

其它资源

- 了解存储池
- 使用 Web 控制台查看存储池信息

15.3.5. 使用 Web 控制台创建基于磁盘的存储池

基于磁盘的存储池使用整个磁盘分区。

警告

- 根据所使用的 libvirt 版本，在存储池中指定一个磁盘可能会重新格式化并清除当前存储在磁盘设备上的所有数据。强烈建议您在创建存储池前备份存储设备中的数据。
- 当整个磁盘或块设备传递给虚拟机时，虚拟机可能会对其分区或者创建自己的 LVM 组。这可能导致主机机器检测到这些分区或者 LVM 组并导致错误。在手动创建分区或 LVM 组并将其传递给虚拟机时，也可以发生这些错误。

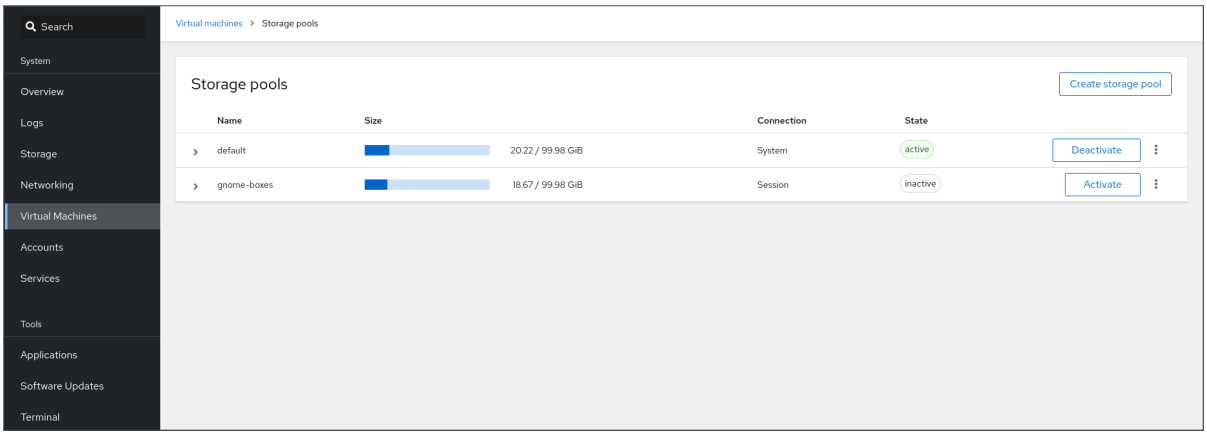
  要避免这些错误，请改为使用基于文件的存储池。

先决条件

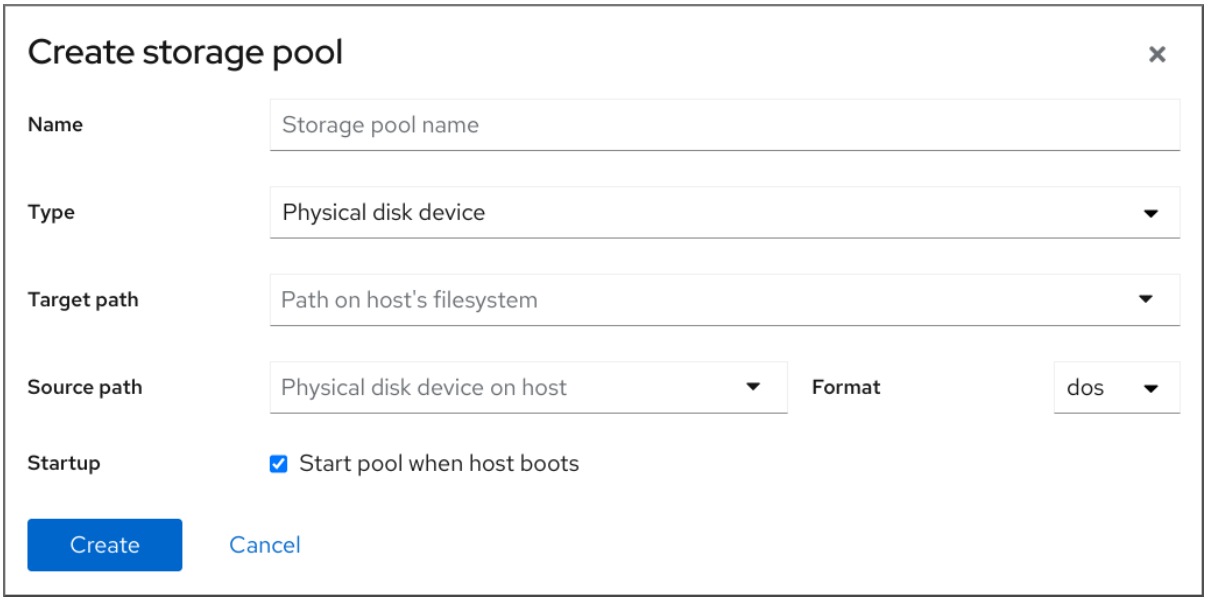
- Web 控制台 VM 插件 已安装在您的系统上。

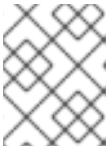
流程

1. 在 RHEL web 控制台中，点 **Virtual Machines** 选项卡中的 **Storage pool**。
   此时会出现 **存储池** 窗口，显示配置的存储池列表（若有）。



2. 单击 **Create storage pool**。
   此时会出现 **Create 存储池** 对话框。
3. 输入存储池的名称。
4. 在 **Type** 下拉菜单中选择 **物理磁盘设备**。





注意

如果您在下拉菜单中选择 **物理磁盘设备** 选项，则您的管理程序不支持基于磁盘的存储池。

5. 输入其他信息：
   - **目标路径** - 指定目标设备的路径。这将是用于存储池的路径。
   - **源路径** - 指定存储设备的路径。例如： **/dev/sdb**。
   - **格式** - 分区表的类型。
   - 启动」 - 主机引导时是否启动存储池。
6. 点击 **Create**。
   已创建存储池。这会关闭 **Create storage pool** 对话框，新的存储池会出现在存储池列表中。

其它资源

- 了解存储池
- 使用 Web 控制台查看存储池信息

15.3.6. 使用 Web 控制台创建基于 LVM 的存储池

基于 LVM 的存储池是基于卷组的，您可以使用逻辑卷管理器(LVM)来管理它。卷组是多个物理卷的组合，它可创建单个存储结构。

注意

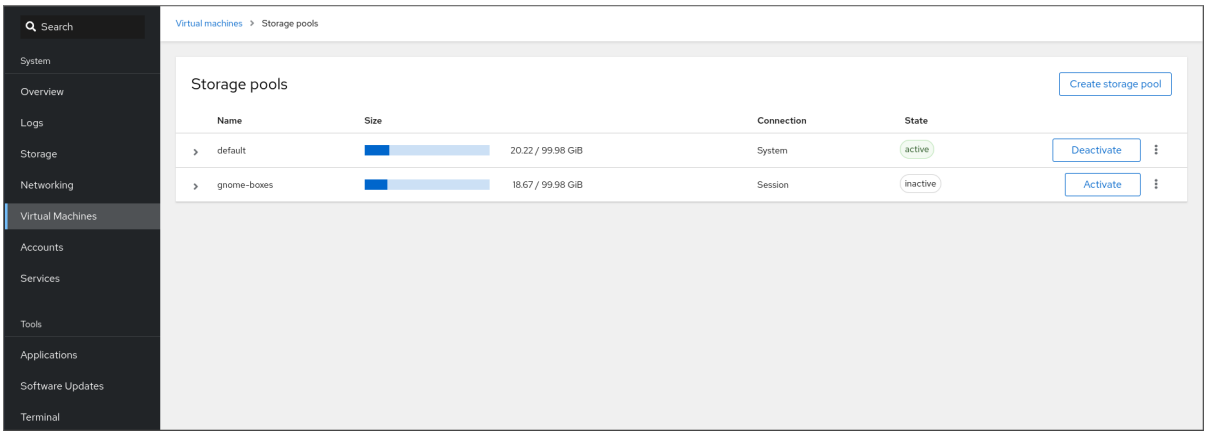
- 基于 LVM 的存储池不能为 LVM 提供完整的灵活性。
- **libvirt** 支持精简逻辑卷，但不提供精简存储池的功能。
- 基于 LVM 的存储池需要一个完整磁盘分区。如果您使用 **virsh** 命令激活一个新分区或设备，分区将被格式化，所有数据都被清除。在这些过程中，如果您使用主机的现有卷组，则不会删除任何内容。
- 要创建具有多个设备的卷组，请使用 LVM 工具，请参阅 如何在 Linux 中 使用 LVM 创建卷组。
  有关卷组的详情，请参考 *Red Hat Enterprise Linux Logical Volume Manager Administration Guide*。

先决条件

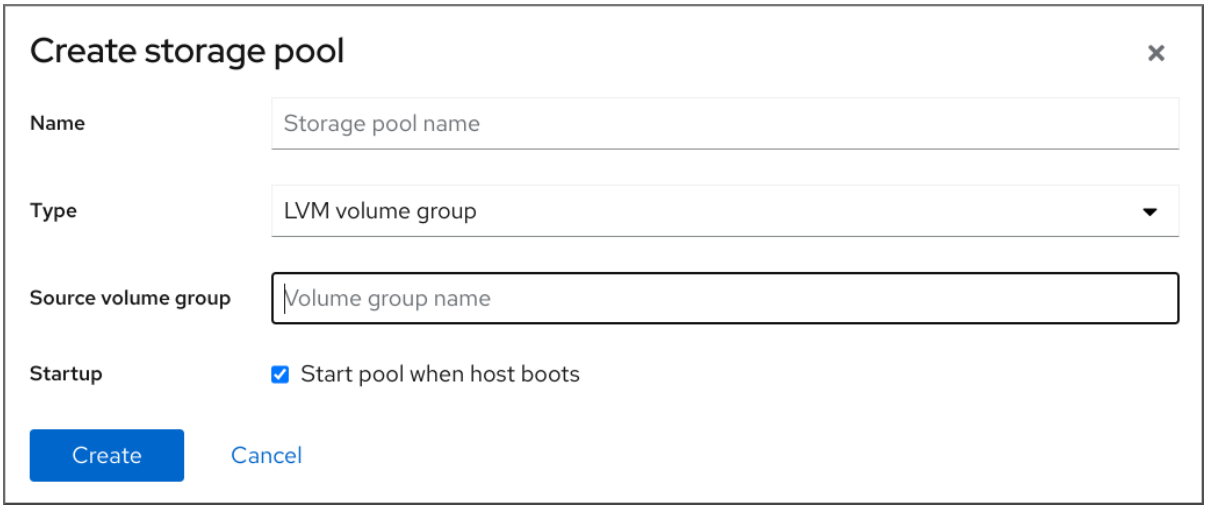
- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 在 RHEL web 控制台中，点 **Virtual Machines** 选项卡中的 **Storage pool**。
   此时会出现 **存储池** 窗口，显示配置的存储池列表（若有）。



2. 单击 **Create storage pool**。
   此时会出现 **Create 存储池** 对话框。
3. 输入存储池的名称。
4. 在 **Type** 下拉菜单中选择 **LVM 卷组**。



注意

如果您在下拉菜单中选择 **LVM 卷组** 选项，则您的管理程序不支持基于 LVM 的存储池。

5. 输入其他信息：
   - **源卷组** - 要使用的 LVM 卷组名称。
   - 启动」 - 主机引导时是否启动存储池。
6. 点击 **Create**。
   已创建存储池。这会关闭 **Create storage pool** 对话框，新的存储池会出现在存储池列表中。

其它资源

- 了解存储池
- 使用 Web 控制台查看存储池信息

15.3.7. 使用 Web 控制台删除存储池

您可以移除存储池，从而释放主机或网络上的资源，以提高系统性能。删除存储池也会释放资源,然后可供其他虚拟机(VM)使用。

重要

除非明确指定，否则删除存储池不会同时删除该池中的存储卷。

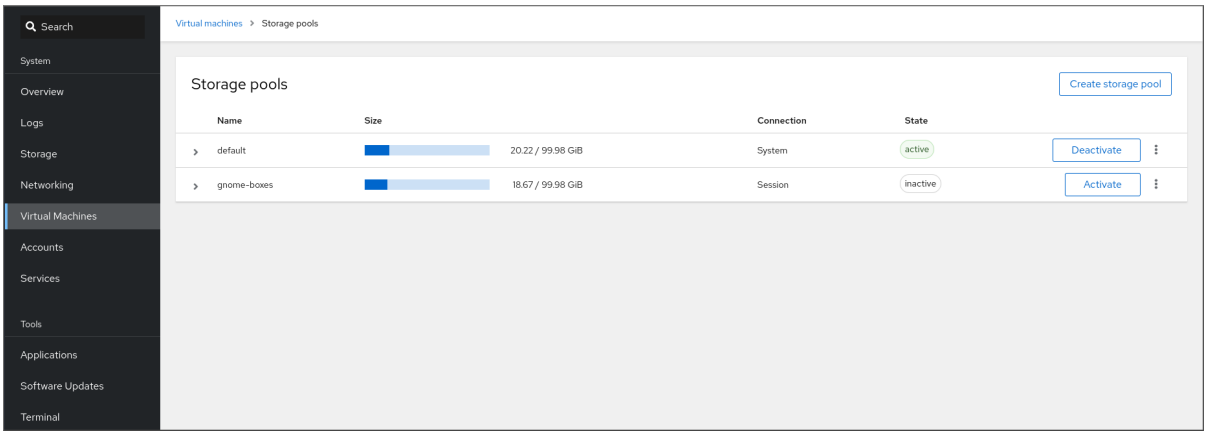
要临时停用一个存储池而不是删除它，请参阅 使用 Web 控制台停用存储池

先决条件

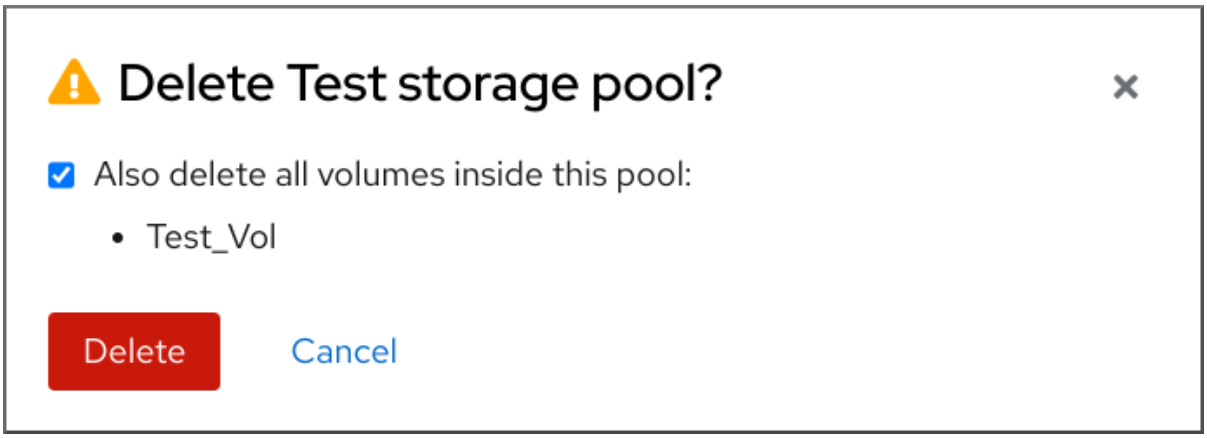
- Web 控制台 VM 插件 已安装在您的系统上。
- 从虚拟机中分离磁盘。
- 如果要删除关联的存储卷以及池，激活这个池。

流程

1. 点 **Virtual Machines** 选项卡上的 **Storage Pools**。
   此时会出现**存储池**窗口，显示配置的存储池列表。



2. 点您要删除的存储池的 Menu 按钮 ⋮，然后点 **Delete**。
   此时会出现确认对话框。



3. **可选：** 要删除池中的存储卷，请在对话框中选择相应的复选框。

4. 点击 **Delete**。
   存储池已删除。如果您已在上一步中选择了复选框，则相关的存储卷也会被删除。

其它资源

- 了解存储池
- 使用 Web 控制台查看存储池信息

15.3.8. 使用 Web 控制台停用存储池

如果您不想永久删除存储池，您可以临时取消激活它。

当您取消激活存储池时，无法在那个池中创建新卷。但是，在池中拥有卷的任何虚拟机(VM)将继续运行。这在很多方面都很有用，例如，您可以限制池中可以创建的卷数量以提高系统性能。

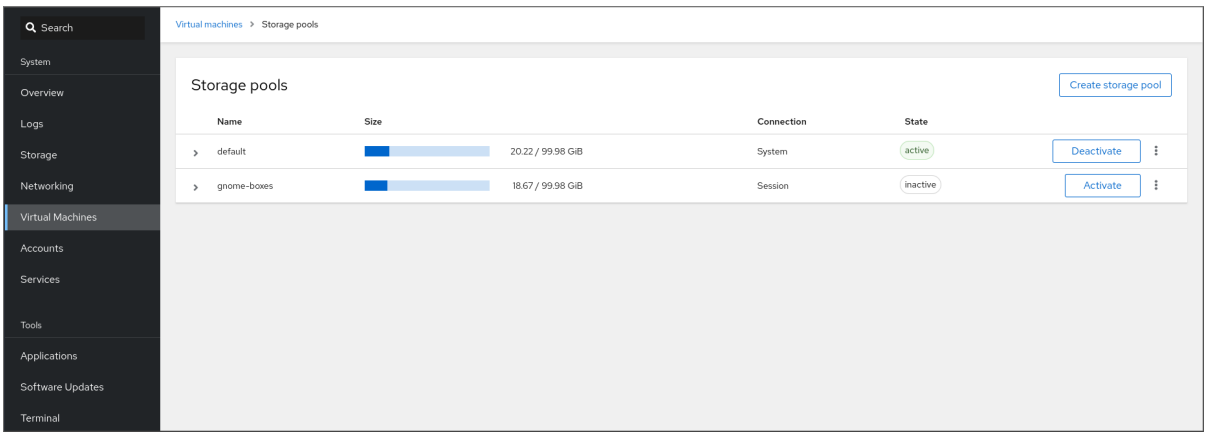
要使用 RHEL web 控制台停用一个存储池，请参阅以下流程。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 点击 Virtual Machines 选项卡顶部的 **Storage Pools**。此时会出现存储池窗口，显示配置的存储池列表。



2. 点存储池行中的 **Deactivate**。
存储池将停用。

其它资源

- 了解存储池
- 使用 Web 控制台查看存储池信息

15.4. 创建存储池的参数

根据您需要的存储池类型，您可以修改其 XML 配置文件，并定义特定的存储池类型。本节提供了关于创建各种类型的存储池所需的 XML 参数以及示例。

15.4.1. 基于目录的存储池参数

当您想使用 XML 配置文件创建或修改基于目录的存储池时，您必须包含某些必要的参数。有关这些参数的更多信息，请参阅下表。

您可以使用 **virsh pool-define** 命令来根据指定文件中的 XML 配置创建存储池。例如：

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

参数

下表提供了基于目录的存储池 XML 文件所需的参数列表。

表 15.1. 基于目录的存储池参数

描述	XML
存储池的类型	**<pool type='dir'>**

描述	XML
存储池的名称	**<name>*name*</name>**
指定目标的路径。这将是用于存储池的路径。	**<target>** **<path>*target_path*</path>** **</target>**

示例

以下是一个基于 **/guest_images** 目录的存储池的 XML 文件示例：

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

其它资源

- 使用 CLI 创建基于目录的存储池

15.4.2. 基于磁盘的存储池参数

当您想使用 XML 配置文件创建或修改基于磁盘的存储池时，您必须包含某些必要的参数。有关这些参数的更多信息，请参阅下表。

您可以使用 **virsh pool-define** 命令来根据指定文件中的 XML 配置创建存储池。例如：

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

参数

下表提供了基于磁盘存储池的 XML 文件所需的参数列表。

表 15.2. 基于磁盘的存储池参数

描述	XML
存储池的类型	**<pool type='disk'>**
存储池的名称	**<name>*name*</name>**
指定存储设备的路径。例如：**/dev/sdb**。	**<source>** **<path>*source_path*</path>** **</source>**

描述	XML
指定目标设备的路径。这将是用于存储池的路径。	**<target>** **<path>*target_path*</path>** **</target>**

示例

以下是基于磁盘存储池的 XML 文件示例：

```
<pool type='disk'>
  <name>phy_disk</name>
  <source>
    <device path='/dev/sdb'/>
    <format type='gpt'/>
  </source>
  <target>
    <path>/dev</path>
  </target>
</pool>
```

其它资源

- 使用 CLI 创建基于磁盘的存储池

15.4.3. 基于文件系统的存储池参数

当您想使用 XML 配置文件创建或修改基于文件系统的存储池时，您必须包含某些必要的参数。有关这些参数的更多信息，请参阅下表。

您可以使用 **virsh pool-define** 命令来根据指定文件中的 XML 配置创建存储池。例如：

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

参数

下表提供了基于文件系统的存储池 XML 文件所需的参数列表。

表 15.3. 基于文件系统的存储池参数

描述	XML
存储池的类型	**<pool type='fs'>**
存储池的名称	**<name>*name*</name>**
指定分区的路径。例如：**/dev/sdc1**	**<source>** **<device path=*device_path* />**

描述	XML
文件系统类型，例如 ext4。	**<format type=*fs_type* /></source>**
指定目标的路径。这将是用于存储池的路径。	**<target><path>*path-to-pool*</path></target>**

示例

以下是基于 **/dev/sdc1** 分区的存储池的一个 XML 文件示例：

```
<pool type='fs'>
  <name>guest_images_fs</name>
  <source>
    <device path='/dev/sdc1'/>
    <format type='auto'/>
  </source>
  <target>
    <path>/guest_images</path>
  </target>
</pool>
```

其它资源

- 使用 CLI 创建基于文件系统的存储池

15.4.4. 基于 iSCSI 的存储池参数

当您想使用 XML 配置文件创建或修改基于 iSCSI 的存储池时，您必须包含某些必要的参数。有关这些参数的更多信息，请参阅下表。

您可以使用 **virsh pool-define** 命令来根据指定文件中的 XML 配置创建存储池。例如：

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

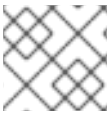
参数

下表提供了基于 iSCSI 存储池的 XML 文件所需的参数列表。

表 15.4. 基于 iSCSI 的存储池参数

描述	XML
存储池的类型	**<pool type='iscsi'>**
存储池的名称	**<name>*name*</name>**

描述	XML
主机的名称	**<source>** **<host name=*hostname* />**
iSCSI IQN	**<device path= *iSCSI_IQN* />** **</source>**
指定目标的路径。这将是用于存储池的路径。	**<target>** **<path>*/dev/disk/by-path*</path>** **</target>**
[可选] iSCSI initiator 的 IQN。只有 ACL 将 LUN 限制为特定发起方时才需要。	**<initiator>** **<iqn name='*initiator0*' />** **</initiator>**



注意

可使用 **virsh find-storage-pool-sources-as** iscsi 命令确定 iSCSI initiator 的 IQN 。

示例

以下是基于指定 iSCSI 设备的存储池的 XML 文件示例：

```
<pool type='iscsi'>
  <name>iSCSI_pool</name>
  <source>
    <host name='server1.example.com'/>
    <device path='iqn.2010-05.com.example.server1:iscsirhel7guest'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

其它资源

- 使用 CLI 创建基于 iSCSI 的存储池

15.4.5. 基于 LVM 的存储池参数

当您想使用 XML 配置文件创建或修改基于 LVM 的存储池时，您必须包含某些必要的参数。有关这些参数的更多信息，请参阅下表。

您可以使用 **virsh pool-define** 命令来根据指定文件中的 XML 配置创建存储池。例如：

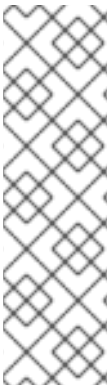
```
# virsh pool-define ~/guest_images.xml
  Pool defined from guest_images_logical
```

参数

下表提供了基于 LVM 的存储池 XML 文件所需的参数列表。

表 15.5. 基于 LVM 的存储池参数

描述	XML
存储池的类型	**<pool type='logical'>**
存储池的名称	**<name>*name*</name>**
存储池设备的路径	**<source>** **<device path='*device_path*' />`**
卷组名称	**<name>*VG-name*</name>**
虚拟组格式	**<format type='lvm2'/>** **</source>**
目标路径	**<target>** **<path=*target_path* />** **</target>**



注意

如果逻辑卷组由多个磁盘分区组成，则可能会列出多个源设备。例如：

```
<source>
  <device path='/dev/sda1'/>
  <device path='/dev/sdb3'/>
  <device path='/dev/sdc2'/>
  ...
</source>
```

示例

以下是基于指定 LVM 的存储池的 XML 文件示例：

```
<pool type='logical'>
  <name>guest_images_lvm</name>
  <source>
    <device path='/dev/sdc'/>
    <name>libvirt_lvm</name>
    <format type='lvm2'/>
  </source>
  <target>
    <path>/dev/libvirt_lvm</path>
  </target>
</pool>
```

其它资源

- 使用 CLI 创建基于 LVM 的存储池

15.4.6. 基于 NFS 的存储池参数

当您想使用 XML 配置文件创建或修改基于 NFS 的存储池时，您必须包含某些必要的参数。有关这些参数的更多信息，请参阅下表。

您可以使用 **virsh pool-define** 命令来根据指定文件中的 XML 配置创建存储池。例如：

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

参数

下表提供了基于 NFS 的存储池 XML 文件所需的参数列表。

表 15.6. 基于 NFS 的存储池参数

描述	XML
存储池的类型	**<pool type='netfs'>**
存储池的名称	**<name>*name*</name>**
挂载点所在的网络服务器的主机名。这可以是主机名或 IP 地址。	**<source>** **<host name=*hostname* />**
存储池的格式	下面是其中之一： **<format type='nfs' />** **<format type='cifs' />**
网络服务器上使用的目录	**<dir path=*source_path* />** **</source>**
指定目标的路径。这将是用于存储池的路径。	**<target>** **<path>*target_path*</path>** **</target>**

示例

以下是基于 **file_server** NFS 服务器的 **/home/net_mount** 目录的存储池的一个 XML 文件示例：

```
<pool type='netfs'>
  <name>nfspool</name>
  <source>
    <host name='file_server'/>
    <format type='nfs'/>
    <dir path='/home/net_mount'/>
  </source>
  <target>
    <path>/var/lib/libvirt/images/nfspool</path>
  </target>
</pool>
```

其它资源

- 使用 CLI 创建基于 NFS 的存储池

15.4.7. 使用 vHBA 设备的基于 SCSI 的存储池的参数

要为使用虚拟主机适配器总线(vHBA)设备的基于 SCSi 的存储池创建或修改 XML 配置文件，您必须在 XML 配置文件中包括某些必要的参数。有关所需参数的更多信息，请参阅下表。

您可以使用 **virsh pool-define** 命令来根据指定文件中的 XML 配置创建存储池。例如：

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

参数

下表提供了使用 vHBA 的基于 SCSI 的存储池 XML 文件所需的参数列表。

表 15.7. 使用 vHBA 设备的基于 SCSI 的存储池的参数

描述	XML
存储池的类型	**<pool type='scsi'>**
存储池的名称	**<name>*name*</name>**
vHBA 的标识符。**parent** 属性是可选的。	**<source>** **<adapter type='fc_host'** **[parent=*parent_scsi_device*]** **wwn='*WWN*'** **wwpn='*WWPN*' />** **</source>**
目标路径。这将是用于存储池的路径。	**<target>** **<path=*target_path* />** **</target>**

重要

当 **<path>** 字段是 **/dev/** 时，**libvirt** 会为卷设备路径生成一个唯一的短设备路径。例如：**/dev/sdc**。否则会使用物理主机路径。例如：**/dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016044602198-lun-0**。唯一的短设备路径允许多个存储池在多个虚拟机(VM)中列出相同的卷。如果多个虚拟机使用了物理主机路径，则可能会出现重复的设备类型警告。

注意

可以在 **<adapter>** 字段中使用 **parent** 属性来通过不同的路径标识 NPIV LUN 的物理 HBA 父级。此字段 **scsi_hostN** 与 **vports** 和 **max_vports** 属性相结合，以完成父识别。**parent**、**parent_wwnn**、**parent_wwpn** 或 **parent_fabric_wwn** 属性提供了不同程度的保证，确保在主机重启后使用相同的 HBA。

- 如果没有指定 **parent**，**libvirt** 将使用支持 NPIV 的第一个 **scsi_hostN** 适配器。
- 如果只指定了 **parent**，则在配置中添加了额外的 SCSI 主机适配器时可能会出现问题。
- 如果指定了 **parent_wwnn** 或 **parent_wwpn**，则在主机重启后使用相同的 HBA。
- 如果使用 **parent_fabric_wwn**，则在主机重新启动后，将选择同一结构上的 HBA，而不考虑所用的 **scsi_hostN**。

示例

以下是使用 vHBA 的基于 SCSI 的存储池的 XML 文件示例。

- 它是 HBA 中唯一存储池的存储池：

```
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <source>
    <adapter type='fc_host' wwnn='5001a4a93526d0a1' wwpn='5001a4ace3ee047d'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

- 存储池是使用单个 vHBA 的多个存储池中的一个，并使用 **parent** 属性来识别 SCSI 主机设备：

```
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <source>
    <adapter type='fc_host' parent='scsi_host3' wwnn='5001a4a93526d0a1'
wwpn='5001a4ace3ee047d'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

其它资源

- 使用 CLI 创建带有 vHBA 设备的基于 SCSI 的存储池

15.5. 使用 CLI 管理虚拟机存储卷

您可以使用 CLI 管理存储卷的以下方面来为虚拟机分配存储：

- 查看存储卷信息

- 创建存储卷
- 删除存储卷

15.5.1. 使用 CLI 查看存储卷信息

使用命令行，您可以查看主机上所有可用存储池的列表，以及指定存储池的详情

流程

1. 使用 **virsh vol-list** 命令列出指定存储池中的存储卷。

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

2. 使用 **virsh vol-info** 命令列出指定存储池中的存储卷。

```
# virsh vol-info --pool RHEL-Storage-Pool --vol RHEL_Volume.qcow2
Name:           RHEL_Volume.qcow2
Type:           file
Capacity:       60.00 GiB
Allocation:     13.93 GiB
```

15.5.2. 使用 CLI 创建并分配存储卷

要获取磁盘镜像，并将其作为虚拟磁盘附加到虚拟机(VM)，请创建一个存储卷，并将其 XML 配置分配到虚拟机。

先决条件

- 主机上存在带有未分配空间的存储池。
  - 要验证，列出主机上的存储池：

```
# virsh pool-list --details

Name             State    Autostart   Persistent   Capacity     Allocation   Available
-------------------------------------------------------------------------------------------
default          running  yes         yes          48.97 GiB    36.34 GiB    12.63 GiB
Downloads        running  yes         yes          175.92 GiB   121.20 GiB   54.72 GiB
VM-disks         running  yes         yes          175.92 GiB   121.20 GiB   54.72 GiB
```

  - 如果您没有现有的存储池，请创建一个。如需更多信息，请参阅为虚拟机管理存储。

流程

1. 使用 **virsh vol-create-as** 命令创建存储卷。例如，要基于 **guest-images-fs** 存储池创建一个 20 GB 的 qcow2 卷：

```
# virsh vol-create-as --pool guest-images-fs --name vm-disk1 --capacity 20 --format qcow2
```

重要 ：特定的存储池类型不支持 **virsh vol-create-as** 命令，而是需要特定的进程来创建存储卷：

- **基于 iSCSI** - 事先在 iSCSI 服务器中准备 iSCSI LUN。
- **基于多路径** - 使用 **multipathd** 命令来准备或管理多路径。
- **基于 vHBA** - 事先准备光纤通道卡。

2. 创建一个 XML 文件，并在其中添加以下几行。此文件将用于将存储卷作为磁盘添加到虚拟机。

```
<disk type='volume' device='disk'>
   <driver name='qemu' type='qcow2'/>
   <source pool='guest-images-fs' volume='vm-disk1'/>
   <target dev='hdk' bus='ide'/>
</disk>
```

本例指定一个在上一步中创建的 **vm-disk1** 卷的虚拟磁盘，并将卷设为 **ide** 总线上的磁盘 **hdk**。根据您的环境修改对应的参数。

重要 ：对于特定的存储池类型，您必须使用不同的 XML 格式来描述存储卷磁盘。

- 对于 *基于多路径的*池：

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

- 对于 *基于 RBD 存储的*池：

```
<disk type='network' device='disk'>
  <driver name='qemu' type='raw'/>
  <source protocol='rbd' name='pool/image'>
    <host name='mon1.example.org' port='6321'/>
  </source>
  <target dev='vdc' bus='virtio'/>
</disk>
```

3. 使用 XML 文件为虚拟机分配存储卷。例如，要将 **~/vm-disk1.xml** 中定义的磁盘分配给 **testguest1** 虚拟机，请使用以下命令：

```
# virsh attach-device --config testguest1 ~/vm-disk1.xml
```

验证

- 在虚拟机的客户机操作系统中，确认磁盘映像已作为未格式化和未分配的磁盘提供。

15.5.3. 使用 CLI 删除存储卷

要从主机系统中删除存储卷，您必须停止池并删除它的 XML 定义。

先决条件

- 任何使用您要删除的存储卷的虚拟机都会被关闭。

流程

1. 使用 **virsh vol-list** 命令列出指定存储池中的存储卷。

```
# virsh vol-list --pool RHEL-SP
 Name                 Path
------------------------------------------------------------------
 .bash_history        /home/VirtualMachines/.bash_history
 .bash_logout         /home/VirtualMachines/.bash_logout
 .bash_profile        /home/VirtualMachines/.bash_profile
 .bashrc              /home/VirtualMachines/.bashrc
 .git-prompt.sh       /home/VirtualMachines/.git-prompt.sh
 .gitconfig           /home/VirtualMachines/.gitconfig
 vm-disk1             /home/VirtualMachines/vm-disk1
```

2. **可选** ：使用 **virsh vol-wipe** 命令来擦除存储卷。例如，要擦除与存储池 **RHEL-SP** 关联的名为 **vm-disk1** 的存储卷：

```
# virsh vol-wipe --pool RHEL-SP vm-disk1
Vol vm-disk1 wiped
```

3. 使用 **virsh vol-delete** 命令来删除存储卷。例如，要删除与存储池 **RHEL-SP** 关联的名为 **vm-disk1** 的存储卷：

```
# virsh vol-delete --pool RHEL-SP vm-disk1
Vol vm-disk1 deleted
```

验证

- 再次使用 **virsh vol-list** 命令来验证存储卷是否已被删除。

```
# virsh vol-list --pool RHEL-SP
 Name                 Path
------------------------------------------------------------------
 .bash_history        /home/VirtualMachines/.bash_history
 .bash_logout         /home/VirtualMachines/.bash_logout
 .bash_profile        /home/VirtualMachines/.bash_profile
 .bashrc              /home/VirtualMachines/.bashrc
 .git-prompt.sh       /home/VirtualMachines/.git-prompt.sh
 .gitconfig           /home/VirtualMachines/.gitconfig
```

15.6. 使用 CLI 管理虚拟磁盘镜像

虚拟磁盘镜像是 虚拟存储卷 的一种类型，以类似于作为硬盘为物理机器提供存储的方式向虚拟机(VM)提供存储。

在创建新虚拟机 时，**libvirt** 会自动创建一个新磁盘镜像，除非您另有指定。但是，根据您的用例，您可能希望创建和管理独立于虚拟机的磁盘镜像。

15.6.1. 使用 qemu-img 创建虚拟磁盘镜像

如果您需要一个独立于新虚拟机(VM)的新的虚拟磁盘镜像 创建一个存储卷 对您来说是不可行的，您可以使用 **qemu-img** 命令行工具。

流程

- 使用 **qemu-img** 工具创建一个虚拟磁盘镜像：

```
# qemu-img create -f <format> <image-name> <size>
```

例如，以下命令创建一个名为 *test-image* 的 qcow2 磁盘镜像，其大小为 30GB。

```
# qemu-img create -f qcow2 test-image 30G

Formatting 'test-img', fmt=qcow2 cluster_size=65536 extended_l2=off compression_type=zlib
size=32212254720 lazy_refcounts=off refcount_bits=16
```

验证

- 显示有关您创建的镜像的信息，并检查它是否有所需的大小，且没有任何损坏：

```
# qemu-img info <test-img>
image: test-img
file format: qcow2
virtual size: 30 GiB (32212254720 bytes)
disk size: 196 KiB
cluster_size: 65536
Format specific information:
    compat: 1.1
    compression type: zlib
    lazy refcounts: false
    refcount bits: 16
    corrupt: false
    extended l2: false
```

其它资源

- 使用 CLI 创建并分配存储卷
- 使用 web 控制台向虚拟机添加新磁盘
- **qemu-img** 手册页

15.6.2. 检查虚拟磁盘镜像的一致性

在将磁盘镜像附加到虚拟机(VM)前，请确保磁盘镜像没有问题，如损坏或碎片太多。为此，您可以使用 **qemu-img check** 命令。

如果需要，您还可以使用这个命令尝试修复磁盘镜像。

先决条件

- 任何使用磁盘镜像的虚拟机(VM)都必须关闭。

流程

1. 对您要测试的镜像使用 **qemu-img check** 命令。例如：

```
# qemu-img check <test-name.qcow2>

No errors were found on the image.
327434/327680 = 99.92% allocated, 0.00% fragmented, 0.00% compressed clusters
Image end offset: 21478375424
```

如果检查在磁盘镜像上发现问题，命令的输出类似如下：

```
167 errors were found on the image.
Data may be corrupted, or further writes to the image may corrupt it.

453368 leaked clusters were found on the image.
This means waste of disk space, but no harm to data.

259 internal errors have occurred during the check.
Image end offset: 21478375424
```

2. 使用带有 **-r all** 选项的 **qemu-img check** 命令修复它们。但请注意，这可能只修复一些问题。

警告

修复磁盘镜像可能会导致数据损坏或其他问题。在尝试修复前备份磁盘镜像。

```
# qemu-img check -r all <test-name.qcow2>

[...]
122 errors were found on the image.
Data may be corrupted, or further writes to the image may corrupt it.

250 internal errors have occurred during the check.
Image end offset: 27071414272
```

此输出显示修复后磁盘镜像上发现的问题数。

3. 如果需要进一步的磁盘镜像修复，您可以在 **guestfish** shell 中使用各种 **libguestfs** 工具。

其它资源

- **qemu-img** 手册页

- **guestfish** 手册页

15.6.3. 调整虚拟磁盘镜像大小

如果现有磁盘镜像需要额外的空间，您可以使用 **qemu-img resize** 工具更改镜像的大小，以适合您的用例。

先决条件

- 您已创建了磁盘镜像的备份。
- 任何使用磁盘镜像的虚拟机(VM)都必须关闭。

> **警告**
>
> 调整正在运行的虚拟机的磁盘镜像大小可能导致数据损坏或其他问题。

- 主机的硬盘有足够的空闲空间，用于预期的磁盘镜像大小。
- 可选：您已确保磁盘镜像没有数据损坏或类似的问题。具体说明请参阅 检查虚拟磁盘镜像的一致性。

流程

1. 确定您要调整大小的虚拟机的磁盘镜像文件的位置。例如：

```
# virsh domblklist <vm-name>

 Target   Source
------------------------------------------------------------
 vda      /home/username/disk-images/example-image.qcow2
```

2. 可选：备份当前磁盘镜像。

```
# cp <example-image.qcow2> <example-image-backup.qcow2>
```

3. 使用 **qemu-img resize** 工具来调整镜像大小。
例如，要将 *<example-image.qcow2>* 大小增加 10GB：

```
# qemu-img resize <example-image.qcow2> +10G
```

4. 调整磁盘镜像中文件系统、分区或物理卷的大小，以使用额外的空间。要在 RHEL 客户机操作系统中这样做，请使用 管理存储设备 和 管理文件系统 中的说明。

验证

1. 显示调整了大小的镜像的信息，并查看它是否有预期的大小：

```
# qemu-img info <converted-image.qcow2>

image: converted-image.qcow2
file format: qcow2
virtual size: 30 GiB (32212254720 bytes)
disk size: 196 KiB
cluster_size: 65536
Format specific information:
    compat: 1.1
    compression type: zlib
    lazy refcounts: false
    refcount bits: 16
    corrupt: false
    extended l2: false
```

2. 检查调整了大小的磁盘镜像是否有潜在的错误。具体说明请参阅 检查虚拟磁盘镜像的一致性。

其它资源

- **qemu-img** 手册页
- 管理存储设备
- 管理文件系统

15.6.4. 在虚拟磁盘镜像格式之间转换

您可以使用 **qemu-img convert** 命令将虚拟磁盘镜像转换为不同的格式。例如，如果要将磁盘镜像附加到运行在不同 hypervisor 上的虚拟机(VM)，在虚拟磁盘镜像格式之间转换可能是必要的。

先决条件

- 任何使用磁盘镜像的虚拟机(VM)都必须关闭。

流程

- 使用 **qemu-im convert** 命令将现有虚拟磁盘镜像转换为不同的格式。例如，将 *raw* 磁盘镜像转换为 *QCOW2* 磁盘镜像：

```
# qemu-img convert -f raw <original-image.img> -O qcow2 <converted-image.qcow2>
```

验证

1. 显示转换的镜像的信息，并查看它是否有预期格式和大小。

```
# qemu-img info <converted-image.qcow2>

image: converted-image.qcow2
file format: qcow2
virtual size: 30 GiB (32212254720 bytes)
disk size: 196 KiB
cluster_size: 65536
Format specific information:
    compat: 1.1
```

```
compression type: zlib
lazy refcounts: false
refcount bits: 16
corrupt: false
extended l2: false
```

2. 检查磁盘镜像是否有潜在的错误。有关说明，请参阅 检查虚拟磁盘镜像的一致性。

其它资源

- 检查虚拟磁盘镜像的一致性
- **qemu-img** 手册页

15.7. 使用 WEB 控制台管理虚拟机存储卷

通过使用 RHEL，您可以管理用来为虚拟机(VM)分配存储的存储卷。

您可以使用 RHEL web 控制台进行：

- 创建存储卷。
- 删除存储卷。

15.7.1. 使用 Web 控制台创建存储卷

要创建可正常工作的虚拟机(VM)，您需要将本地存储设备分配给虚拟机的，该设备可存储虚拟机镜像及虚拟机相关的数据。您可以在存储池中创建存储卷，并将其分配为作为存储磁盘的虚拟机。

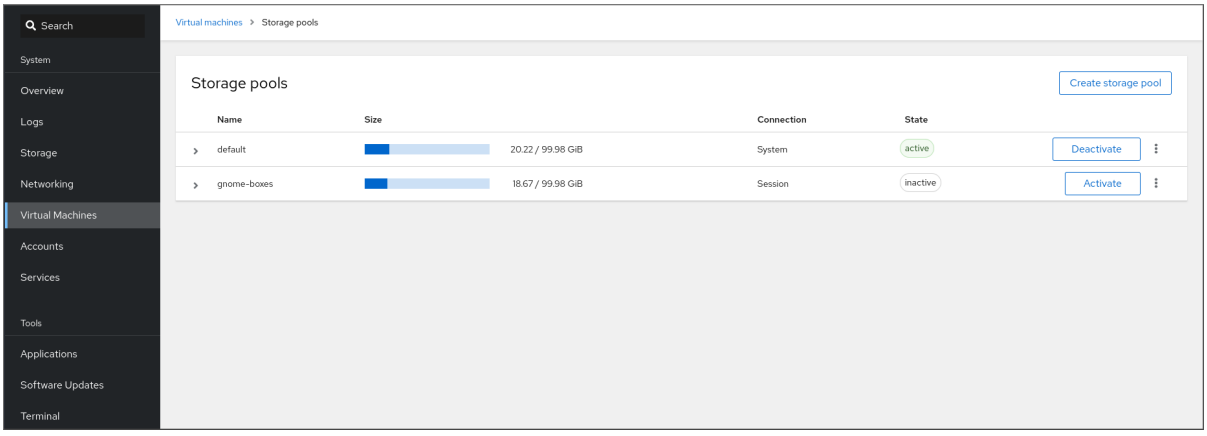
要使用 Web 控制台创建存储卷，请参阅以下流程。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

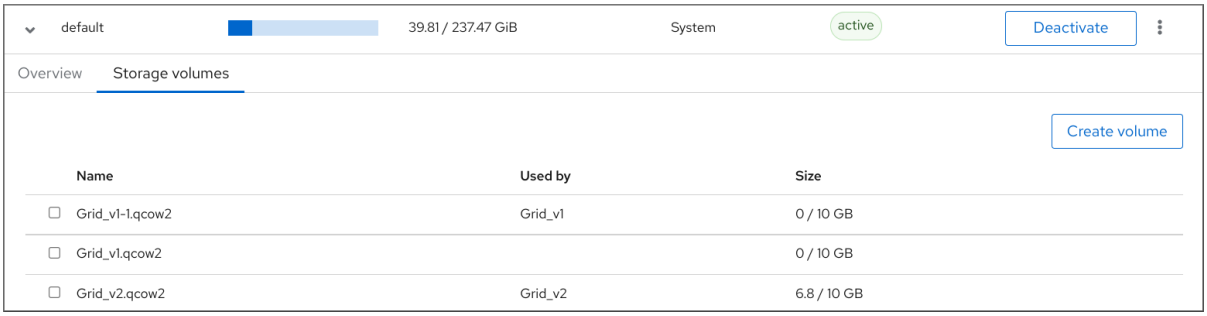
流程

1. 点击 Virtual Machines 选项卡顶部的 **Storage Pools**。此时会出现存储池窗口，显示配置的存储池列表。



2. 在 **Storage Pools** 窗口中，点击您要创建存储卷的存储池。
行会展开，以显示包含所选存储池基本信息的 Overview 窗格。

3. 点展开行中的 Overview 选项卡旁的 **Storage Volumes**。
Storage Volume 选项卡会出现有关现有存储卷的基本信息。



4. 点**创建卷**。
此时会出现 **Create storage volume** 对话框。

5. 在 Create Storage Volume 对话框中输入以下信息：
   - **Name** - 存储卷的名称。
   - **size** - MiB 或 GiB 存储卷的大小。
   - **格式** - 存储卷的格式。支持的类型为 **qcow2** 和 **raw**。
6. 点击 **Create**。
存储卷创建后，创建存储卷对话框关闭，新的存储卷会出现在存储卷列表中。

其它资源

- 了解存储卷
- 使用 web 控制台向虚拟机添加新磁盘

15.7.2. 使用 Web 控制台删除存储卷

您可以删除存储卷来释放存储池中的空间，或删除与失效的虚拟机(VM)相关联的存储条目。

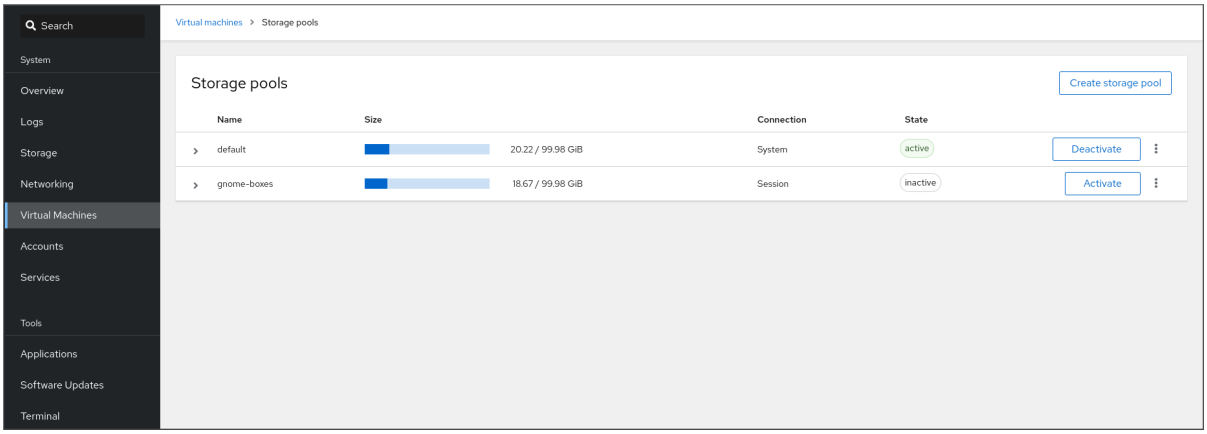
要使用 RHEL web 控制台删除存储卷，请参阅以下流程。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。
- 任何使用您要删除的存储卷的虚拟机都会被关闭。

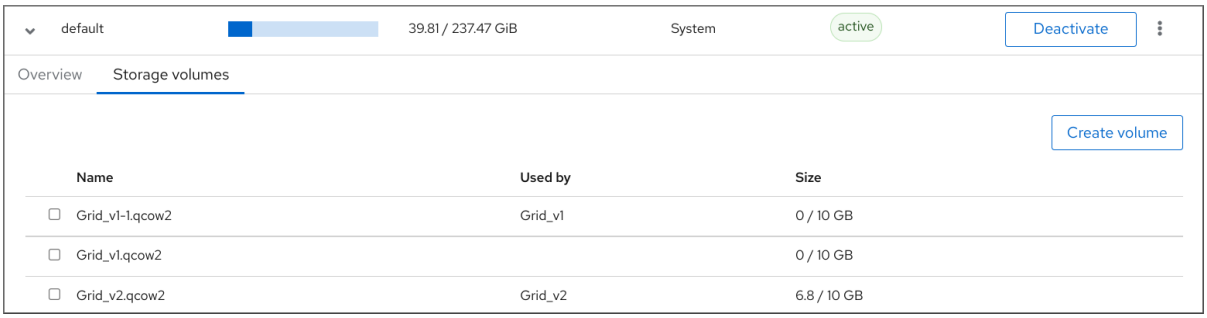
流程

1. 点击 Virtual Machines 选项卡顶部的 **Storage Pools**。此时会出现存储池窗口，显示配置的存储池列表。

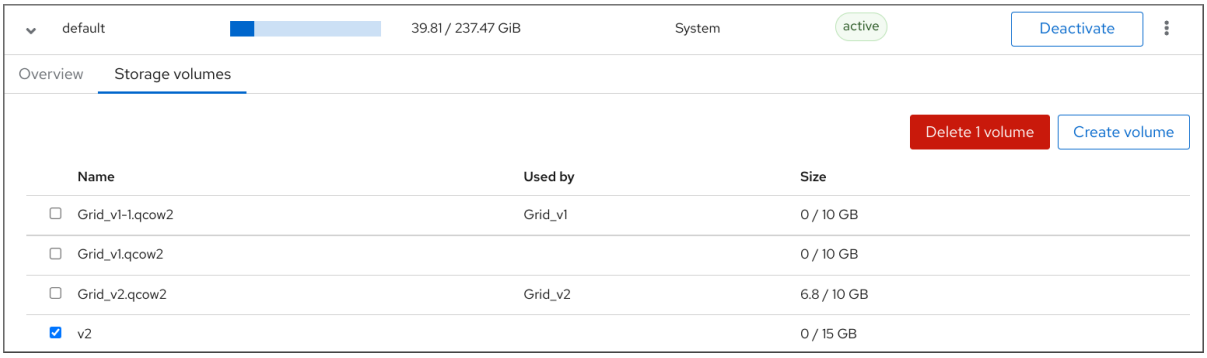


2. 在 **Storage Pools** 窗口中，单击您要从中删除存储卷的存储池。行会展开，以显示包含所选存储池基本信息的 Overview 窗格。

3. 点展开行中的 Overview 选项卡旁的 **Storage Volumes**。Storage Volume 选项卡会出现有关现有存储卷的基本信息。



4. 选择您要删除的存储卷。



5. 单击 **Delete 1 Volume**

其它资源

- 了解存储卷

15.8. 使用 WEB 控制台管理虚拟机存储磁盘

使用 RHEL，您可以管理附加到虚拟机的存储磁盘。

您可以使用 RHEL web 控制台进行：

- 查看虚拟机磁盘信息。
- 向虚拟机添加新磁盘。
- 将磁盘附加到虚拟机。
- 从虚拟机中分离磁盘。

15.8.1. 在 web 控制台中查看虚拟机磁盘信息

通过使用 web 控制台，您可以查看分配给所选虚拟机(VM)的磁盘的详细信息。

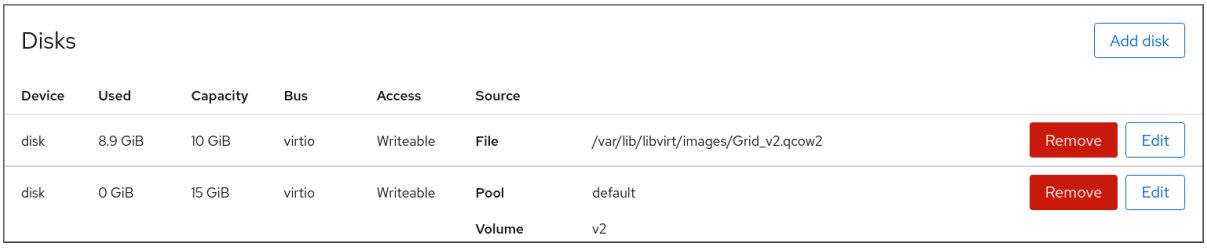
先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 点击您要查看其信息的虚拟机。
   此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟机的图形界面的控制台部分。

2. 滚动到 **磁盘** 。
   Disks 部分显示有关分配给虚拟机的磁盘的信息，以及用于 **Add** 或 **Edit** 磁盘的选项。



该信息包括：

- **Device** - 磁盘的设备类型。
- **Used** - 当前分配的磁盘数量。
- **Capacity** - 存储卷的最大大小。
- **Bus** - 模拟的磁盘设备的类型。

- **Access** - 磁盘是否是 **可写** 或 **只读**。对于 **raw** 磁盘，您还可以将访问权限设为 **可写和共享**。
- **Source** - 磁盘设备或者文件。

其它资源

- 使用 web 控制台查看虚拟机信息

15.8.2. 使用 web 控制台向虚拟机添加新磁盘

您可以通过创建新存储卷，并使用 RHEL 9 web 控制台将其附加到虚拟机来向虚拟机添加新磁盘。

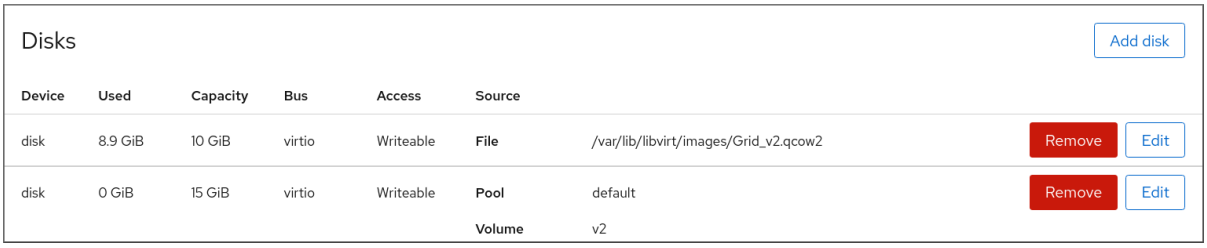
先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

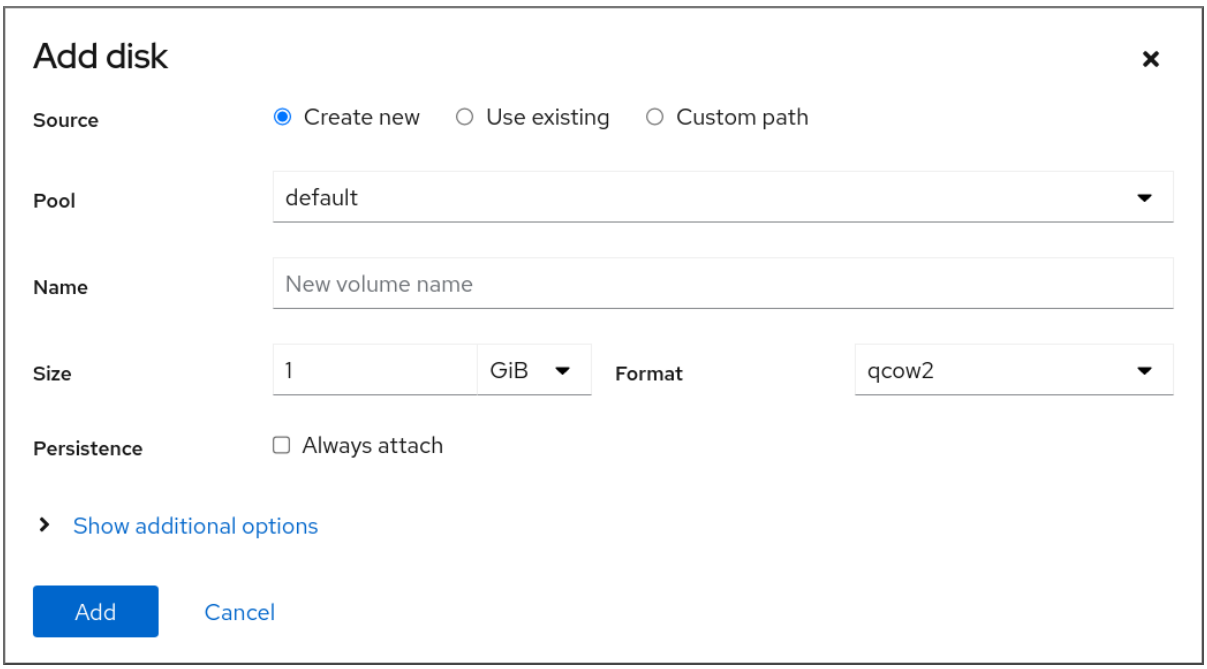
流程

1. 在 **Virtual Machines** 界面中，单击要为其创建并附加新磁盘的虚拟机。
   此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟机的图形界面的控制台部分。

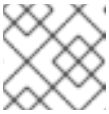
2. 滚动到 **磁盘** 。
   Disks 部分显示有关分配给虚拟机的磁盘的信息，以及用于 **Add** 或 **Edit** 磁盘的选项。



3. 点 **Add Disk**。
   此时会出现 Add Disk 对话框。



4. 选择 *Create New* 选项。
5. 配置新磁盘。
   - **Pool** - 选择创建虚拟磁盘的存储池。
   - **Name** - 为要创建的虚拟磁盘输入一个名称。
   - **Size** - 输入大小并选择要创建的虚拟磁盘的单元（MiB 或 GiB）。
   - **Format** - 选择要创建的虚拟磁盘的格式。支持的类型为 **qcow2** 和 **raw**。
   - **Persistence** - 如果选中，虚拟磁盘是永久的。如果没有选择，虚拟磁盘就是临时的。

     注意

     临时磁盘只能添加到正在运行的虚拟机中。

   - **其它选项** - 为虚拟磁盘设置附加配置。
     - **Cache** - 选择缓存机制。
     - **Bus** - 选择要模拟的磁盘设备的类型。
     - **Disk Identifier** - 为附加磁盘设置标识符，可用于多路径存储设置。在使用授权给特定磁盘序列号的专有软件时，标识符也很有用。
6. 点**添加**。
   虚拟磁盘已创建并连接到虚拟机。

其它资源

- 在 web 控制台中查看虚拟机磁盘信息
- 使用 web 控制台将现有磁盘附加到虚拟机
- 使用 web 控制台将磁盘从虚拟机中分离

15.8.3. 使用 web 控制台将现有磁盘附加到虚拟机

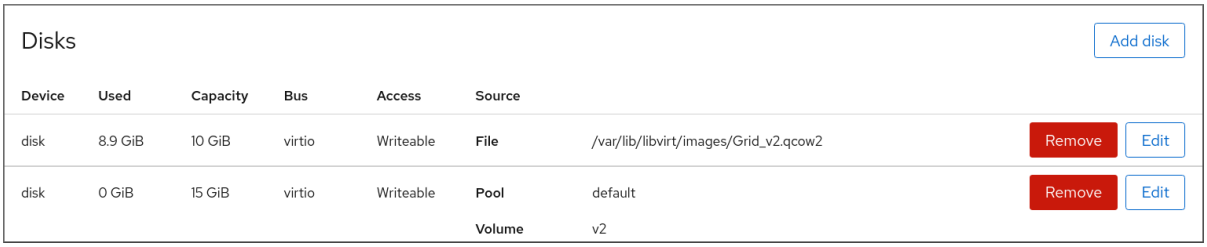
使用 web 控制台，您可以将现有存储卷作为磁盘附加到虚拟机。

先决条件

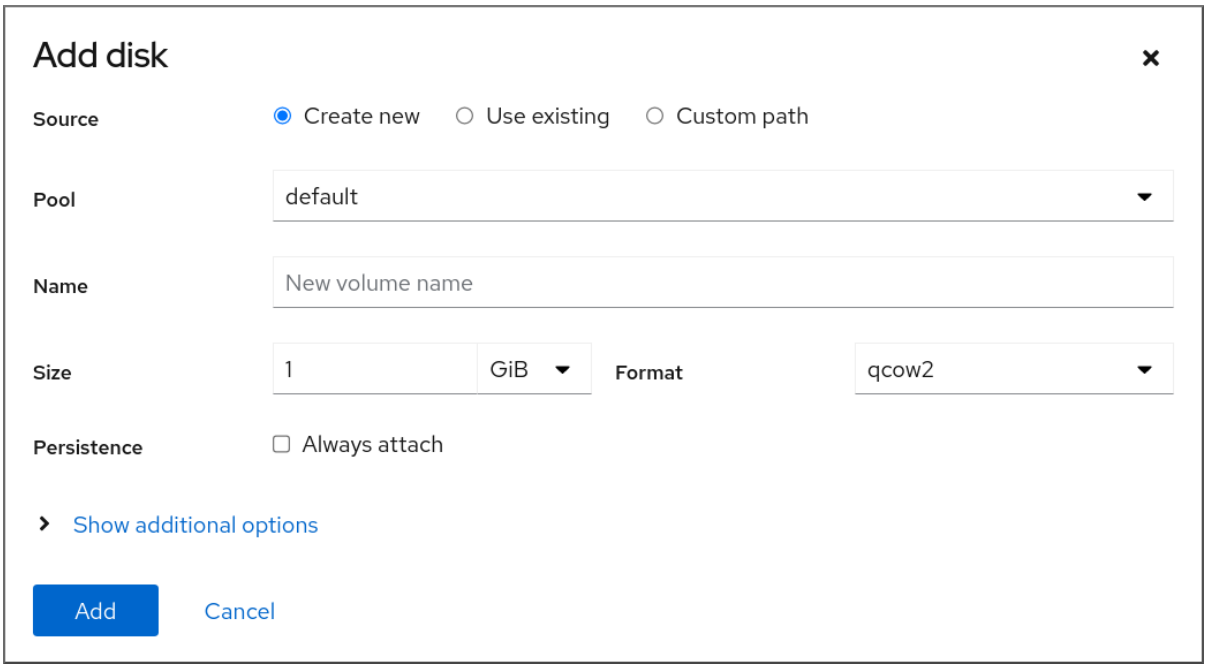
- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 在 **Virtual Machines** 界面中，单击要为其创建并附加新磁盘的虚拟机。
   此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟机的图形界面的控制台部分。
2. 滚动到 **磁盘** 。
   Disks 部分显示有关分配给虚拟机的磁盘的信息，以及用于 **Add** 或 **Edit** 磁盘的选项。



3. 点 **Add Disk**。
此时会出现 Add Disk 对话框。



4. 点**使用现有**按钮。
正确的配置字段会出现在 Add Disk 对话框中。

5. 为虚拟机配置磁盘。

   - **Pool** - 选择要从中附加虚拟磁盘的存储池。
   - **Volume** - 选择将被附加的存储卷。
   - **Persistence** - 虚拟机运行时可用。选中 **Always attach** 复选框，使虚拟磁盘持久可用。清除复选框，使虚拟磁盘为临时磁盘。
   - **其它选项** - 为虚拟磁盘设置附加配置。
     - **Cache** - 选择缓存机制。
     - **Bus** - 选择要模拟的磁盘设备的类型。
     - **Disk Identifier** - 为附加磁盘设置标识符，可用于多路径存储设置。在使用授权给特定磁盘序列号的专有软件时，标识符也很有用。

6. 点 **Add**
所选虚拟磁盘附加到虚拟机。

其它资源

- 在 web 控制台中查看虚拟机磁盘信息
- 使用 web 控制台向虚拟机添加新磁盘
- 使用 web 控制台将磁盘从虚拟机中分离

15.8.4. 使用 web 控制台将磁盘从虚拟机中分离

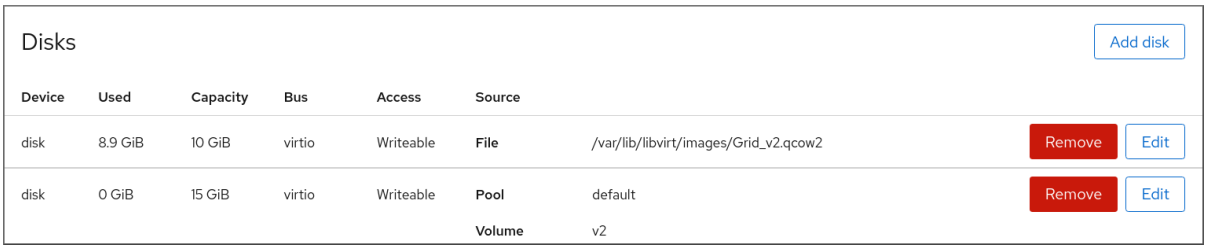
使用 web 控制台，您可以将磁盘从虚拟机(VM)中分离。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 在 **Virtual Machines** 界面中，单击您要从中分离磁盘的虚拟机。
   此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟机的图形界面的控制台部分。
2. 滚动到 **Disks**。
   Disks 部分显示有关分配给虚拟机的磁盘的信息，以及用于 **Add** 或 **Edit** 磁盘的选项。



3. 在您要分离的磁盘行的右侧，点菜单按钮 ⋮。
4. 在出现的下拉菜单中，点 **Remove** 按钮。
   此时会出现 **Remove disk from VM?** 确认对话框。
5. 在确认对话框中，单击 **Remove**。可选，如果您还想删除磁盘镜像，请单击 **Remove and delete file**。
   虚拟磁盘与虚拟机分离。

其它资源

- 在 web 控制台中查看虚拟机磁盘信息
- 使用 web 控制台向虚拟机添加新磁盘
- 使用 web 控制台将现有磁盘附加到虚拟机

15.9. 使用 LIBVIRT SECRET 保护 ISCSI 存储池

可以使用 **virsh** 配置用户名和密码参数，来保护 iSCSI 存储池的安全。您可以在定义池之前或之后配置它，但必须启动池才能使验证设置生效。

以下提供了使用 **libvirt** secret 保护基于 iSCSI 的存储池的说明。

注意

如果在创建 iSCSI 目标时定义了 ***user_ID*** 和 ***密码***，则需要这个流程。

先决条件

- 确保您已创建了一个基于 iSCSI 的存储池。如需更多信息，请参阅 使用 CLI 创建基于 iSCSI 的存储池。

流程

1. 创建具有质询握手身份验证协议(CHAP)用户名的 libvirt secret 文件。例如：

```
<secret ephemeral='no' private='yes'>
   <description>Passphrase for the iSCSI example.com server</description>
   <usage type='iscsi'>
      <target>iscsirhel7secret</target>
   </usage>
</secret>
```

2. 使用 **virsh secret-define** 命令定义 libvirt secret：

```
# virsh secret-define secret.xml
```

3. 使用 **virsh secret-list** 命令验证 UUID：

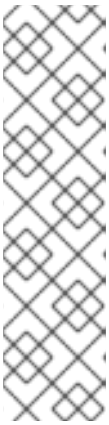
```
# virsh secret-list
UUID                                   Usage
--------------------------------------------------------------------
2d7891af-20be-4e5e-af83-190e8a922360   iscsi iscsirhel7secret
```

4. 使用 **virsh secret-set-value** 命令，为上一步输出中的 UUID 分配 secret。这样可保证 CHAP 用户名和密码位于由 libvirt 控制的 secret 列表中。例如：

```
# virsh secret-set-value --interactive 2d7891af-20be-4e5e-af83-190e8a922360
Enter new value for secret:
Secret value set
```

5. 使用 **virsh edit** 命令在存储池的 XML 文件中添加一个身份验证条目，并添加 **<auth>** 元素，指定 **authentication type**、**username** 和 **secret usage**。例如：

```
<pool type='iscsi'>
  <name>iscsirhel7pool</name>
    <source>
      <host name='192.0.2.1'/>
      <device path='iqn.2010-05.com.example.server1:iscsirhel7guest'/>
      <auth type='chap' username='_example-user_'>
        <secret usage='iscsirhel7secret'/>
      </auth>
    </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```



注意

<auth> 子元素位于虚拟机的 **<pool>** 和 **<disk>** XML 元素的不同位置。对于 **<pool>**，**<auth>** 在 **<source>** 元素中指定，这描述了在哪里查找池源，因为身份验证是某些池源（iSCSI 和 RBD）的属性。对于是域子元素的 **<disk>**，对 iSCSI 或 RBD 磁盘的身份验证是磁盘的一个属性。另外，磁盘的 **<auth>** 子元素与存储池的 <auth> 子元素不同。

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

6. 要激活更改，激活存储池。如果池已启动，停止并重启存储池：

```
# virsh pool-destroy iscsirhel7pool
# virsh pool-start iscsirhel7pool
```

15.10. 创建 VHBA

虚拟主机总线适配器(vHBA)设备将主机系统连接到 SCSI 设备，对于创建基于 SCSI 的存储池是必需的。

您可以通过在 XML 配置文件中定义 vHBA 设备来创建它。

流程

1. 使用 **virsh nodedev-list --cap vports** 命令定位主机系统上的 HBA。
以下示例显示了支持 vHBA 的两个 HBA 的主机：

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

2. 使用 **virsh nodedev-dumpxml *HBA_device*** 命令查看 HBA 的详情。

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

命令的输出列出了用于创建 vHBA 的 **<name>**、**<wwnn>** 和 **<wwpn>** 字段。**<max_vports>** 显示支持的 vHBA 的最大数。例如：

```
<device>
  <name>scsi_host3</name>
  <path>/sys/devices/pci0000:00/0000:00:04.0/0000:10:00.0/host3</path>
  <parent>pci_0000_10_00_0</parent>
  <capability type='scsi_host'>
    <host>3</host>
    <unique_id>0</unique_id>
    <capability type='fc_host'>
      <wwnn>20000000c9848140</wwnn>
      <wwpn>10000000c9848140</wwpn>
      <fabric_wwn>2002000573de9a81</fabric_wwn>
    </capability>
    <capability type='vport_ops'>
      <max_vports>127</max_vports>
```

```
      <vports>0</vports>
    </capability>
  </capability>
</device>
```

在本例中，**<max_vports>** 值显示在 HBA 配置中总共有 127 个虚拟端口可用。 **<vports>** 值显示当前正在使用的虚拟端口数。这些值在创建 vHBA 后更新。

3. 为 vHBA 主机创建类似如下的 XML 文件。在这些示例中，该文件名为 **vhba_host3.xml**。本例使用 **scsi_host3** 来描述父 vHBA。

```
<device>
  <parent>scsi_host3</parent>
  <capability type='scsi_host'>
    <capability type='fc_host'>
    </capability>
  </capability>
</device>
```

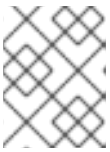
这个示例使用 WWNN/WWPN 对描述父 vHBA。

```
<device>
  <name>vhba</name>
  <parent wwnn='20000000c9848140' wwpn='10000000c9848140'/>
  <capability type='scsi_host'>
    <capability type='fc_host'>
    </capability>
  </capability>
</device>
```

注意

WWNN 和 WWPN 值必须与上一步中 HBA 详情中的值匹配。

<parent> 字段指定与这个 vHBA 设备关联的 HBA 设备。下一步中使用 **<device>** 标签中的详细信息，来为主机创建新的 vHBA 设备。有关 **nodedev** XML 格式的更多信息，请参阅 libvirt 上游页面。



注意

virsh 命令不提供定义 **parent_wwnn**、**parent_wwpn** 或 **parent_fabric_wwn** 属性的方式。

4. 使用 **virsh nodev-create** 命令，根据上一步中创建的 XML 文件创建 VHBA。

```
# virsh nodedev-create vhba_host3
Node device scsi_host5 created from vhba_host3.xml
```

验证

- 使用 **virsh nodedev-dumpxml** 命令验证新 vHBA 的详情(scsi_host5)：

```
# virsh nodedev-dumpxml scsi_host5
```

```
<device>
  <name>scsi_host5</name>
  <path>/sys/devices/pci0000:00/0000:00:04.0/0000:10:00.0/host3/vport-3:0-0/host5</path>
  <parent>scsi_host3</parent>
  <capability type='scsi_host'>
    <host>5</host>
    <unique_id>2</unique_id>
    <capability type='fc_host'>
      <wwnn>5001a4a93526d0a1</wwnn>
      <wwpn>5001a4ace3ee047d</wwpn>
      <fabric_wwn>2002000573de9a81</fabric_wwn>
    </capability>
  </capability>
</device>
```

其它资源

- 使用 CLI 创建带有 vHBA 设备的基于 SCSI 的存储池

第 16 章 在虚拟机中管理 GPU 设备

要在 RHEL 9 主机上增强虚拟机(VM)的图形性能，您可以将主机 GPU 分配给虚拟机。

- 您可以从主机中分离 GPU，并将 GPU 直接控制传递给虚拟机。
- 您可以从物理 GPU 创建多个中介设备，并将这些设备作为虚拟 GPU(vGPU)分配给多个客户机。目前仅支持所选的 NVIDIA GPU，且只能为单个客户机分配一个中介设备。

重要

目前只在 Intel 64 和 AMD64 系统上支持 GPU 分配。

16.1. 为虚拟机分配 GPU

要访问并控制附加到主机系统的 GPU，您必须将主机系统配置为将 GPU 的直接控制传给虚拟机(VM)。

注意

如果您要查找有关分配虚拟 GPU 的信息，请参阅管理 NVIDIA vGPU 设备。

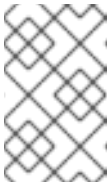
先决条件

- 您必须在主机机器内核中启用 IOMMU 支持。
  - 在 Intel 主机上，您必须启用 VT-d：
    1. 使用 **intel_iommu=on** 和 **iommu=pt** 参数重新生成 GRUB 配置：

       ```
       # grubby --args="intel_iommu=on iommu_pt" --update-kernel DEFAULT
       ```

    2. 重启主机。
  - 在 AMD 主机上，您必须启用 AMD-Vi。
    请注意，在 AMD 主机上，默认启用 IOMMU，您可以添加 **iommu=pt** 将其切换到直通模式：
    1. 使用 **iommu=pt** 参数重新生成 GRUB 配置：

       ```
       # grubby --args="iommu=pt" --update-kernel DEFAULT
       ```

       注意

       pt 选项只为直通模式中使用的设备启用 IOMMU，并提供更好的主机性能。但是，并非所有硬件都支持这个选项。即使未启用这个选项，您仍然可以分配设备。

    2. 重启主机。

流程

1. 防止驱动程序绑定到 GPU。
   a. 识别 GPU 连接到的 PCI 总线地址。

```
# lspci -Dnn | grep VGA
0000:02:00.0 VGA compatible controller [0300]: NVIDIA Corporation GK106GL [Quadro K4000] [10de:11fa] (rev a1)
```

b. 防止主机的图形驱动程序使用 GPU。要做到这一点，使用带有 pci-stub 驱动程序的 GPU PCI ID。
例如，以下命令可防止驱动程序绑定到附加在 **10de:11fa** 总线的 GPU：

```
# grubby --args="pci-stub.ids=10de:11fa" --update-kernel DEFAULT
```

c. 重启主机。

2. **可选：** 如果因为支持的限制而无法将某些 GPU 功能（如音频）直通给虚拟机，您可以修改 IOMMU 组中端点的驱动程序绑定，使其只能直通必要的 GPU 功能。

a. 将 GPU 设置转换为 XML，并记下要防止附加到主机驱动程序的端点的 PCI 地址。
为此，通过向地址添加 **pci_** 前缀，将 GPU 的 PCI 总线地址转换为 libvirt 兼容的格式，并将分隔符转换为下划线。

例如，以下命令显示附加在 **0000:02:00.0** 总线地址的 GPU 的 XML 配置。

```
# virsh nodedev-dumpxml pci_0000_02_00_0
```

```
<device>
 <name>pci_0000_02_00_0</name>
 <path>/sys/devices/pci0000:00/0000:00:03.0/0000:02:00.0</path>
 <parent>pci_0000_00_03_0</parent>
 <driver>
  <name>pci-stub</name>
 </driver>
 <capability type='pci'>
  <domain>0</domain>
  <bus>2</bus>
  <slot>0</slot>
  <function>0</function>
  <product id='0x11fa'>GK106GL [Quadro K4000]</product>
  <vendor id='0x10de'>NVIDIA Corporation</vendor>
  <iommuGroup number='13'>
   <address domain='0x0000' bus='0x02' slot='0x00' function='0x0'/>
   <address domain='0x0000' bus='0x02' slot='0x00' function='0x1'/>
  </iommuGroup>
  <pci-express>
   <link validity='cap' port='0' speed='8' width='16'/>
   <link validity='sta' speed='2.5' width='16'/>
  </pci-express>
 </capability>
</device>
```

b. 防止端点附加到主机驱动程序。
在本例中，要将 GPU 分配给虚拟机，防止与音频功能对应的端点 **<address domain='0x0000' bus='0x02' slot='0x00' function='0x1'/>** 附加到主机音频驱动程序，请将端点附加到 VFIO-PCI。

```
# driverctl set-override 0000:02:00.1 vfio-pci
```

3. 将 GPU 附加到虚拟机

   a. 使用 PCI 总线地址为 GPU 创建 XML 配置文件。
   例如，您可以使用 GPU 总线地址中的参数创建以下 XML 文件，GPU-Assign.xml。

```
<hostdev mode='subsystem' type='pci' managed='yes'>
 <driver name='vfio'/>
 <source>
  <address domain='0x0000' bus='0x02' slot='0x00' function='0x0'/>
 </source>
</hostdev>
```

   b. 将文件保存到主机系统上。

   c. 将文件与虚拟机的 XML 配置合并。
   例如，以下命令将 GPU XML 文件 GPU-Assign.xml 与 **System1** 虚拟机的 XML 配置文件合并：

```
# virsh attach-device System1 --file /home/GPU-Assign.xml --persistent
Device attached successfully.
```

注意

GPU 是作为辅助图形设备附加到虚拟机的。不支持将 GPU 分配为主图形设备，红帽不建议删除虚拟机 XML 配置中的主模拟图形设备。

验证

- 该设备会出现在虚拟机 XML 配置的 **<devices>** 部分中。如需更多信息，请参阅示例虚拟机 XML 配置。

已知问题

- 可附加到虚拟机的 GPU 数量受分配的 PCI 设备的最大数量限制，在 RHEL 9 中，当前为 64。但是，将多个 GPU 附加到虚拟机可能会导致客户端上内存映射 I/O (MMIO) 出现问题，这可能会导致 GPU 无法对虚拟机使用。
  要临时解决这个问题，请设置较大的 64 位 MMIO 空间并配置 vCPU 物理地址位，以使扩展 64 位 MMIO 空间可以寻址。

- 将 NVIDIA GPU 设备附加到使用 RHEL 9 客户机操作系统的虚拟机当前会禁用该虚拟机上的 Wayland 会话，而改为加载 Xorg 会话。这是因为 NVIDIA 驱动程序和 Wayland 之间的不兼容。

16.2. 管理 NVIDIA VGPU 设备

vGPU 功能使得可以将物理 NVIDIA GPU 设备划分为多个虚拟设备，称为 **中介设备** 。然后可将这些 mediated devices 分配给多个虚拟机（VM）作为虚拟 GPU。因此，这些虚拟机可以共享单个物理 GPU 的性能。

重要

将物理 GPU 分配给使用或不使用中介设备的虚拟机，使主机无法使用 GPU。

16.2.1. 设置 NVIDIA vGPU 设备

要设置 NVIDIA vGPU 功能，您需要为 GPU 设备下载 NVIDIA vGPU 驱动程序，创建介质设备，并将其分配给预期的虚拟机。具体步骤请查看以下说明。

先决条件

- 您的 GPU 支持 vGPU 介质设备。有关支持创建 vGPU 的 NVIDIA GPU 的最新列表，请参阅 NVIDIA vGPU 软件文档。
  - 如果您不知道主机正在使用哪个 GPU，请安装 *lshw* 软件包，并使用 **lshw -C display** 命令。以下示例显示系统使用与 vGPU 兼容的 NVIDIA Tesla P4 GPU。

```
# lshw -C display

*-display
       description: 3D controller
       product: GP104GL [Tesla P4]
       vendor: NVIDIA Corporation
       physical id: 0
       bus info: pci@0000:01:00.0
       version: a1
       width: 64 bits
       clock: 33MHz
       capabilities: pm msi pciexpress cap_list
       configuration: driver=vfio-pci latency=0
       resources: irq:16 memory:f6000000-f6ffffff memory:e0000000-efffffff
memory:f0000000-f1ffffff
```

流程

1. 下载 NVIDIA vGPU 驱动程序并在您的系统中安装它们。具体步骤请查看 NVIDIA 文档。
2. 如果 NVIDIA 软件安装程序没有创建 **/etc/modprobe.d/nvidia-installer-disable-nouveau.conf** 文件，请在 **/etc/modprobe.d/** 中创建一个任何名称的 **conf** 文件，并在文件中添加以下行：

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

3. 为当前内核重新生成初始 ramdisk，然后重新启动。

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

4. 检查内核是否已加载了 **nvidia_vgpu_vfio** 模块，**nvidia-vgpu-mgr.service** 服务是否正在运行。

```
# lsmod | grep nvidia_vgpu_vfio
nvidia_vgpu_vfio 45011 0
nvidia 14333621 10 nvidia_vgpu_vfio
mdev 20414 2 vfio_mdev,nvidia_vgpu_vfio
vfio 32695 3 vfio_mdev,nvidia_vgpu_vfio,vfio_iommu_type1

# systemctl status nvidia-vgpu-mgr.service
nvidia-vgpu-mgr.service - NVIDIA vGPU Manager Daemon
   Loaded: loaded (/usr/lib/systemd/system/nvidia-vgpu-mgr.service; enabled; vendor preset:
```

```
disabled)
   Active: active (running) since Fri 2018-03-16 10:17:36 CET; 5h 8min ago
 Main PID: 1553 (nvidia-vgpu-mgr)
 [...]
```

另外，如果基于 NVIDIA Ampere GPU 设备创建 vGPU，请确保为物理 GPU 启用虚拟功能。具体步骤请查看 NVIDIA 文档。

5. 生成设备 UUID。

```
# uuidgen
30820a6f-b1a5-4503-91ca-0c10ba58692a
```

6. 根据检测到的 GPU 硬件，使用 mediated 设备配置准备 XML 文件。例如，以下命令会在 0000:01:00.0 PCI 总线上运行的 NVIDIA Tesla P4 卡中配置 **nvidia-63** vGPU 类型的介质设备，并使用上一步中生成的 UUID。

```
<device>
   <parent>pci_0000_01_00_0</parent>
   <capability type="mdev">
      <type id="nvidia-63"/>
      <uuid>30820a6f-b1a5-4503-91ca-0c10ba58692a</uuid>
   </capability>
</device>
```

7. 根据您准备的 XML 文件定义 vGPU 介质设备。例如：

```
# virsh nodedev-define vgpu-test.xml
Node device mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0 created
from vgpu-test.xml
```

8. **可选：** 验证中介设备是否被列为不活动状态。

```
# virsh nodedev-list --cap mdev --inactive
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
```

9. 启动您创建的 vGPU 介质设备。

```
# virsh nodedev-start mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Device mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0 started
```

10. **可选：** 确保 中介设备被列为活动状态。

```
# virsh nodedev-list --cap mdev
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
```

11. 将 vGPU 设备设置为在主机重启后自动启动

```
# virsh nodedev-autostart
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Device mdev_d196754e_d8ed_4f43_bf22_684ed698b08b_0000_9b_00_0 marked as
autostarted
```

12. 将 mediated 设备附加到要共享 vGPU 资源的虚拟机。要做到这一点，请将以下行以及之前生成的 UUID 添加到虚拟机 XML 配置的 **<devices/>** 部分。

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci' display='on'>
  <source>
    <address uuid='30820a6f-b1a5-4503-91ca-0c10ba58692a'/>
  </source>
</hostdev>
```

请注意，每个 UUID 每次只能分配给一个虚拟机。另外，如果虚拟机没有 QEMU 视频设备，如 **virtio-vga**，在 **<hostdev>** 行中添加 **ramfb='on'** 参数。

13. 要使 vGPU 中介设备的全部功能在分配的虚拟机上可用，请在虚拟机上设置 NVIDIA vGPU 虚拟机软件许可。有关详情和说明，请参阅 NVIDIA Virtual GPU Software License Server User Guide。

验证

1. 查询您创建的 vGPU 的功能，并确保它列为 active 和 persistent。

```
# virsh nodedev-info mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Name:           virsh nodedev-autostart
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Parent:         pci_0000_01_00_0
Active:         yes
Persistent:     yes
Autostart:      yes
```

2. 启动虚拟机并验证客户端操作系统是否检测到 mediated device 作为 NVIDIA GPU。例如，如果虚拟机使用 Linux：

```
# lspci -d 10de: -k
07:00.0 VGA compatible controller: NVIDIA Corporation GV100GL [Tesla V100 SXM2 32GB]
(rev a1)
        Subsystem: NVIDIA Corporation Device 12ce
        Kernel driver in use: nvidia
        Kernel modules: nouveau, nvidia_drm, nvidia
```

已知问题

- 将 NVIDIA vGPU 中介设备分配给使用 RHEL 9 客户机操作系统的虚拟机当前会禁用该虚拟机上的 Wayland 会话，而改为加载 Xorg 会话。这是因为 NVIDIA 驱动程序和 Wayland 之间的不兼容。

其它资源

- NVIDIA vGPU 软件文档
- **man virsh** 命令

16.2.2. 删除 NVIDIA vGPU 设备

要更改分配的 vGPU 介质设备 的配置，您需要从分配的虚拟机中删除现有设备。具体步骤请查看以下操作：

先决条件

- 要从中删除该设备的虚拟机关闭。

流程

1. 获取您要删除的介质设备的 ID。

```
# virsh nodedev-list --cap mdev
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
```

2. 停止 vGPU mediated 设备的运行实例。

```
# virsh nodedev-destroy mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Destroyed node device 'mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0'
```

3. **可选：** 确保中介设备已被停用。

```
# virsh nodedev-info mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Name:           virsh nodedev-autostart
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Parent:         pci_0000_01_00_0
Active:         no
Persistent:     yes
Autostart:      yes
```

4. 从虚拟机 XML 配置中删除该设备。为此，使用 **virsh edit** 工具编辑虚拟机的 XML 配置，并删除 mdev 的配置段。这个片段类似如下：

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci'>
  <source>
    <address uuid='30820a6f-b1a5-4503-91ca-0c10ba58692a'/>
  </source>
</hostdev>
```

请注意，停止和分离 mediated 设备不会删除它，而是将其保留为 **defined**。因此，您可以重启并把设备附加到不同的虚拟机。

5. **可选：** 要删除已停用的中介设备，请删除其定义。

```
# virsh nodedev-undefine
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
Undefined node device 'mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0'
```

验证

- 如果您只停止和分离该设备，请确保介质设备被列为 inactive。

```
# virsh nodedev-list --cap mdev --inactive
mdev_30820a6f_b1a5_4503_91ca_0c10ba58692a_0000_01_00_0
```

- 如果您也删除了该设备，请确保以下命令不会显示它。

```
# virsh nodedev-list --cap mdev
```

-

其它资源

- **man virsh** 命令

16.2.3. 获取有关您系统的 NVIDIA vGPU 信息

要评估可用的 vGPU 功能，您可以获取系统上关于中介设备的其它信息，例如：

- 可创建给定类型的 mediated 设备
- 您的系统中已经配置了哪些介质设备。

流程

- 要查看您主机上可以支持 vGPU 介质设备的可用 GPU 设备，请使用 **virsh nodedev-list --cap mdev_types** 命令。例如，下面显示了有两个 NVIDIA Quadro RTX6000 设备的系统。

```
# virsh nodedev-list --cap mdev_types
pci_0000_5b_00_0
pci_0000_9b_00_0
```

- 要显示特定 GPU 设备支持的 vGPU 类型以及其他元数据，请使用 **virsh nodedev-dumpxml** 命令。

```
# virsh nodedev-dumpxml pci_0000_9b_00_0
<device>
 <name>pci_0000_9b_00_0</name>
 <path>/sys/devices/pci0000:9a/0000:9a:00.0/0000:9b:00.0</path>
 <parent>pci_0000_9a_00_0</parent>
 <driver>
  <name>nvidia</name>
 </driver>
 <capability type='pci'>
  <class>0x030000</class>
  <domain>0</domain>
  <bus>155</bus>
  <slot>0</slot>
  <function>0</function>
  <product id='0x1e30'>TU102GL [Quadro RTX 6000/8000]</product>
  <vendor id='0x10de'>NVIDIA Corporation</vendor>
  <capability type='mdev_types'>
   <type id='nvidia-346'>
    <name>GRID RTX6000-12C</name>
    <deviceAPI>vfio-pci</deviceAPI>
    <availableInstances>2</availableInstances>
   </type>
   <type id='nvidia-439'>
    <name>GRID RTX6000-3A</name>
    <deviceAPI>vfio-pci</deviceAPI>
    <availableInstances>8</availableInstances>
   </type>
   [...]
   <type id='nvidia-440'>
    <name>GRID RTX6000-4A</name>
```

```
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>6</availableInstances>
     </type>
     <type id='nvidia-261'>
      <name>GRID RTX6000-8Q</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>3</availableInstances>
     </type>
    </capability>
    <iommuGroup number='216'>
     <address domain='0x0000' bus='0x9b' slot='0x00' function='0x3'/>
     <address domain='0x0000' bus='0x9b' slot='0x00' function='0x1'/>
     <address domain='0x0000' bus='0x9b' slot='0x00' function='0x2'/>
     <address domain='0x0000' bus='0x9b' slot='0x00' function='0x0'/>
    </iommuGroup>
    <numa node='2'/>
    <pci-express>
     <link validity='cap' port='0' speed='8' width='16'/>
     <link validity='sta' speed='2.5' width='8'/>
    </pci-express>
   </capability>
</device>
```

其它资源

- **man virsh** 命令

16.2.4. 用于 NVIDIA vGPU 的远程桌面流服务

在启用了 NVIDIA vGPU 或 NVIDIA GPU passthrough 的 RHEL 9 管理程序中支持以下远程桌面流服务：

- HP ZCentral Remote Boost/Teradici
- NICE DCV
- Mechdyne TGX

有关支持详情请查看适当的供应商支持列表。

16.2.5. 其它资源

- NVIDIA vGPU 软件文档

第 17 章 配置虚拟机网络连接

要使虚拟机(VM)通过网络连接到主机、主机上的其他虚拟机以及外部网络中的位置，必须相应地配置虚拟机网络。为了提供虚拟机网络，RHEL 9 管理程序和新创建的虚拟机具有默认网络配置，也可以进一步修改。例如：

- 您可以使主机上的虚拟机通过主机外的位置被发现和连接，就如同虚拟机与主机位于同一网络中一样。
- 您可以部分或完全将虚拟机与入站网络流量隔离，以提高其安全性，并将虚拟机出现任何问题而影响主机的风险降至最低。

以下章节解释了各种类型的虚拟机网络配置，并提供了设置所选虚拟机网络配置的说明。

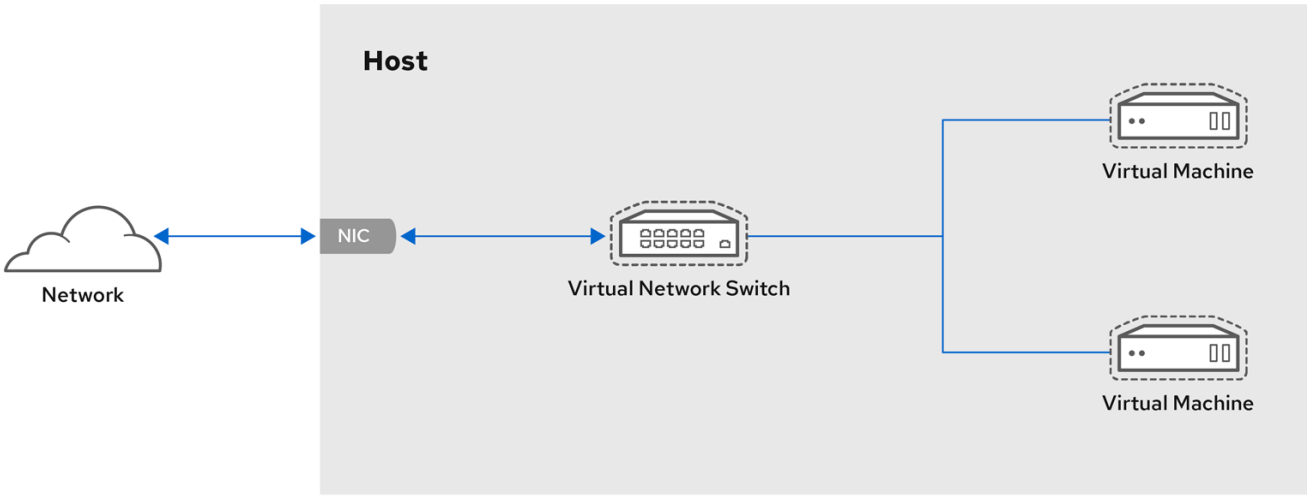
17.1. 了解虚拟网络

虚拟机(VM)连接到网络上的其他设备和位置必须由主机硬件来提供。以下小节解释了虚拟机网络连接的机制，并描述了默认的虚拟机网络设置。

17.1.1. 虚拟网络的工作方式

虚拟网络使用了虚拟网络交换机的概念。虚拟网络交换机是在主机机器中运行的软件构造。VM 通过虚拟网络交换机连接到网络。根据虚拟交换机的配置，虚拟机可以使用由 hypervisor 管理的现有虚拟网络或不同的网络连接方法。

下图展示了将两个虚拟机连接到网络的虚拟网络交换机：



从客户机操作系统的角度来看，虚拟网络连接与物理网络连接是一样的。主机虚拟机将虚拟网络交换机视为网络接口。当 **virtnetworkd** 服务首次安装并启动时，它会创建 **virbr0**，即虚拟机的默认网络接口。

要查看有关此接口的信息，请使用主机上的 **ip** 工具。

```
$ ip addr show virbr0
3: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UNKNOWN link/ether 1b:c4:94:cf:fd:17 brd ff:ff:ff:ff:ff:ff
inet 192.0.2.1/24 brd 192.0.2.255 scope global virbr0
```

默认情况下，单个主机上的所有虚拟机都连接到名为 **default** 的同一 NAT 类型 虚拟网络，该网络使用 **virbr0** 接口。详情请参阅虚拟网络默认配置。

对于从虚拟机进行基础出站网络访问，通常不需要额外的网络设置，因为默认网络会与 **libvirt-daemon-config-network** 软件包一起安装，并在启动 **virtnetworkd** 服务时自动启动。

如果需要不同的虚拟机网络功能，您可以创建额外的虚拟网络和网络接口，并将虚拟机配置为使用它们。除了默认的 NAT 外，这些网络和接口也可以配置为使用以下一种模式：

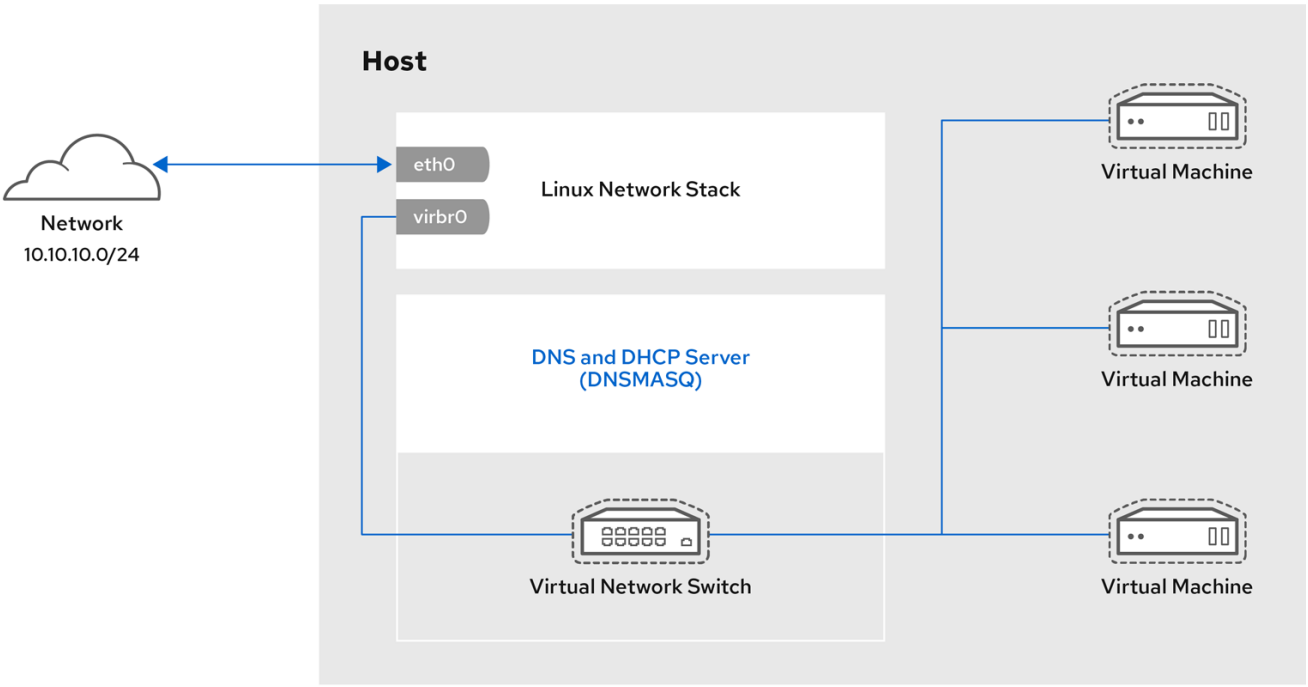
- 路由模式
- 网桥模式
- 隔离模式
- 开放模式

17.1.2. 虚拟网络默认配置

当在虚拟化主机上安装 **virtnetworkd** 服务时，它会在网络地址转换(NAT)模式中包含初始虚拟网络配置。默认情况下，主机上的所有虚拟机都连接到同一 **libvirt** 虚拟网络，名为 **default**。这个网络上的虚拟机可以连接到主机上和主机之外的网络位置，但有以下限制：

- 网络上的虚拟机对主机和主机上的其他虚拟机可见，但网络流量会受到客户机操作系统网络堆栈中的防火墙以及附加到客户机接口的 **libvirt** 网络过滤规则的影响。
- 网络上的虚拟机可以连接到主机之外但对它们不可见的位置。出站流量受 NAT 规则以及主机系统的防火墙影响。

下图演示了默认虚拟机网络配置：



17.2. 使用 WEB 控制台管理虚拟机网络接口

使用 RHEL 9 web 控制台，您可以管理连接到 web 控制台的虚拟机的虚拟网络接口。您可以：

- 查看网络接口的相关信息并进行编辑。
- 向虚拟机添加网络接口，以及 断开或删除接口。

17.2.1. 在 web 控制台中查看和编辑虚拟网络接口信息

通过使用 RHEL 9 web 控制台，您可以在所选虚拟机(VM)上查看和修改虚拟网络接口：

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 在 **Virtual Machines** 界面中，点击您要查看其信息的虚拟机。
   此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟机的图形界面的控制台部分。

2. 滚动 **到网络接口** 。
   网络接口部分显示为虚拟机配置的虚拟网络接口的信息，以及 **添加**、**删除**、**编辑** 或 **拔掉** 网络接口的选项。

   该信息包括：

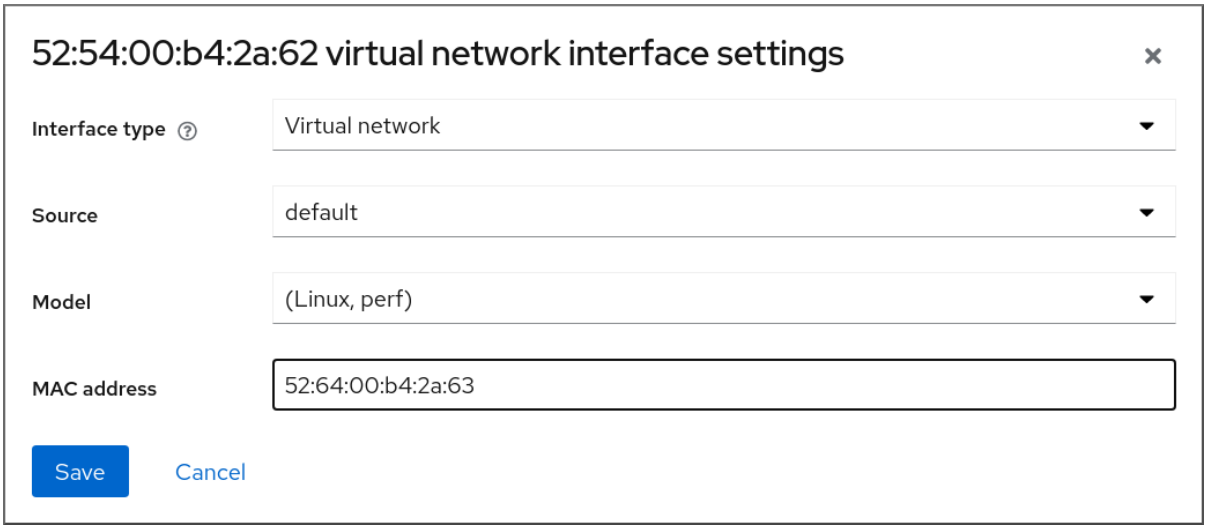
   - **类型** - 虚拟机的网络接口类型。类型包括虚拟网络、网桥到 LAN 以及直接附加。

注意

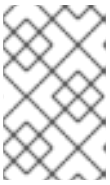
RHEL 9 及更高版本不支持通用以太网连接。

   - **型号类型** - 虚拟网络接口的型号。
   - **MAC 地址** - 虚拟网络接口的 MAC 地址。
   - **IP 地址** - 虚拟网络接口的 IP 地址。
   - **Source** - 网络接口源。这取决于网络类型。
   - **State** - 虚拟网络接口的状态。

3. 要编辑虚拟网络接口设置，请点 **Edit**。此时会打开「虚拟网络接口设置」对话框。



4. 更改接口类型、源、型号或 MAC 地址。
5. 点 **Save**。已修改网络接口。



注意

对虚拟网络接口设置的更改仅在重启虚拟机后生效。

此外，MAC 地址只能在虚拟机关闭时修改。

其它资源

- 使用 web 控制台查看虚拟机信息

17.2.2. 在 web 控制台中添加和连接虚拟网络接口

通过使用 RHEL 9 web 控制台，您可以创建一个虚拟网络接口，并将虚拟机(VM)连接到它。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 在 **Virtual Machines** 界面中，点击您要查看其信息的虚拟机。
   此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟机的图形界面的控制台部分。
2. 滚动 **到网络接口** 。
   网络接口部分显示为虚拟机配置的虚拟网络接口的信息，以及 **Add**、**Edit** 或 **Plug** 网络接口的选项。
3. 点击您要连接的虚拟网络接口所在行中的 **Plug**。
   所选虚拟网络接口连接至虚拟机。

17.2.3. 在 web 控制台中断开和删除虚拟网络接口

通过使用 RHEL 9 web 控制台，您可以断开连接到所选虚拟机(VM)的虚拟网络接口。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 在 **Virtual Machines** 界面中，点击您要查看其信息的虚拟机。
   此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟机的图形界面的控制台部分。

2. 滚动 **到网络接口** 。
   网络接口部分显示为虚拟机配置的虚拟网络接口的信息，以及 **添加**、**删除**、**编辑** 或 **拔掉** 网络接口的选项。

3. 在您要断开连接的虚拟网络接口行中点 **Unplug**。
   所选虚拟网络接口断开与虚拟机的连接。

17.3. 推荐的虚拟机网络配置

在很多情况下，默认的 VM 网络配置已经足够了。但是，如果需要调整配置，您可以使用命令行界面 (CLI) 或 RHEL 9 web 控制台进行此操作。以下章节描述了这些情况下所选择的虚拟机网络设置。

17.3.1. 使用命令行界面配置外部可见的虚拟机

默认情况下，新创建的虚拟机连接到使用 **virbr0** （主机上默认的虚拟网桥）的 NAT 类型网络。这可确保虚拟机可以使用主机的网络接口控制器(NIC)来连接到外部网络，但无法从外部系统访问虚拟机。

如果需要虚拟机出现在与 hypervisor 相同的外部网络上，您必须使用 桥接模式。为此，请将虚拟机连接到连接到 hypervisor 物理网络设备的网桥设备。要使用命令行界面，请遵循以下步骤。

先决条件

- 带有默认 NAT 设置的现有虚拟机。

- 管理程序的 IP 配置。这随着主机的网络连接而变化。例如，这个流程使用这样一种场景，其中主机使用以太网线连接到网络，主机的物理 NIC MAC 地址被分配给 DHCP 服务器上的一个静态 IP。因此，以太网接口被视为 hypervisor IP。
  要获取以太网接口的 IP 配置，请使用 **ip addr** 工具：

```
# 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 192.0.2.1/24 brd 192.0.2.255 scope global dynamic noprefixroute enp0s25
```

流程

1. 为主机上的物理接口创建和设置网桥连接。具体步骤请参阅 配置网络桥接。

请注意，在使用静态 IP 分配的场景中，您必须将物理以太网接口的 IPv4 设置移到网桥接口。

2. 修改虚拟机的网络，以使用创建的桥接接口。例如,以下设置 *testguest* 使用 *bridge0*。

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

3. 启动虚拟机。

```
# virsh start testguest
```

4. 在客户机操作系统中，调整系统网络接口的 IP 和 DHCP 设置，就好像虚拟机是与 hypervisor 位于同一网络中的另一台物理系统一样。
具体步骤将因虚拟机使用的客户端操作系统而异。例如，如果客户机操作系统是 RHEL 9，请参阅配置以太网连接。

验证

1. 确保新创建的网桥正在运行，并且包含主机的物理接口和虚拟机的接口。

```
# 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. 确保虚拟机出现在与 hypervisor 相同的外部网络上：

   a. 在客户机操作系统中，获取系统的网络 ID。例如，如果它是 Linux 客户机：

```
# 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 192.0.2.1/24 brd 192.0.2.255 scope global dynamic noprefixroute enp0s0
```

   b. 从连接到本地网络的外部系统，使用获取的 ID 连接到虚拟机。

```
# ssh root@192.0.2.1
root@192.0.2.1's password:
Last login: Mon Sep 24 12:05:36 2019
root~#*
```

   如果连接正常工作，则网络已配置成功。

故障排除

- 在某些情况下，比如当虚拟机托管在客户端上时使用客户端到站点的 VPN，使用桥接模式使您的虚拟机可用于外部位置是不可能的。
要临时解决这个问题，您可以 使用 **nftables** 为虚拟机设置目标 NAT。

其它资源

- 使用 web 控制台配置外部可见的虚拟机
- 网桥模式中的虚拟网络

17.3.2. 使用 web 控制台配置外部可见的虚拟机

默认情况下，新创建的虚拟机连接到使用 **virbr0** （主机上默认的虚拟网桥）的 NAT 类型网络。这可确保虚拟机可以使用主机的网络接口控制器(NIC)来连接到外部网络，但无法从外部系统访问虚拟机。

如果需要虚拟机出现在与 hypervisor 相同的外部网络上，您必须使用 桥接模式。为此，请将虚拟机连接到连接到 hypervisor 物理网络设备的网桥设备。要使用 RHEL 9 web 控制台，请遵循以下步骤。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。
- 带有默认 NAT 设置的现有虚拟机。
- 管理程序的 IP 配置。这随着主机的网络连接而变化。例如，这个流程使用这样一种场景，其中主机使用以太网线连接到网络，主机的物理 NIC MAC 地址被分配给 DHCP 服务器上的一个静态 IP。因此，以太网接口被视为 hypervisor IP。
  要获取以太网接口的 IP 配置，请转至 web 控制台中的 **Networking** 选项卡，并查看 **接口** 部分。

  步骤

  1. 为主机上的物理接口创建和设置网桥连接。具体步骤请参阅在 web 控制台中配置网络桥接。请注意，在使用静态 IP 分配的场景中，您必须将物理以太网接口的 IPv4 设置移到网桥接口。
  2. 修改虚拟机的网络，以使用桥接接口。在虚拟机的 网络接口 选项卡中：
     a. 点击 **Add Network Interface**
     b. 在 **Add Virtual Network Interface** 对话框中：
        - 将 **Interface Type** 设成 **Bridge to LAN**
        - Source 到新创建的网桥，比如 **bridge0**
     c. 点 **Add**
     d. **可选** ：对连接到虚拟机的所有其他接口，点击 **Unplug** 。
  3. 点击 **Run** 来启动虚拟机。
  4. 在客户机操作系统中，调整系统网络接口的 IP 和 DHCP 设置，就好像虚拟机是与 hypervisor 位于同一网络中的另一台物理系统一样。
     具体步骤将因虚拟机使用的客户端操作系统而异。例如，如果客户机操作系统是 RHEL 9，请参阅配置以太网连接。

验证

1. 在主机 Web 控制台的 **Networking** 选项卡中，单击新创建的网桥所在的行，以确保它正在运行，并且包含主机的物理接口和虚拟机接口。

2. 确保虚拟机出现在与虚拟机监控程序相同的外部网络中。

   a. 在客户机操作系统中，获取系统的网络 ID。例如，如果它是 Linux 客户机：

```
# 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 192.0.2.1/24 brd 192.0.2.255 scope global dynamic noprefixroute enp0s0
```

   b. 从连接到本地网络的外部系统，使用获取的 ID 连接到虚拟机。

```
# ssh root@192.0.2.1
root@192.0.2.1's password:
Last login: Mon Sep 24 12:05:36 2019
root~#*
```

   如果连接正常工作，则网络已配置成功。

故障排除

- 在某些情况下，比如当虚拟机托管在客户端上时使用客户端到站点的 VPN，使用桥接模式使您的虚拟机可用于外部位置是不可能的。

其他资源

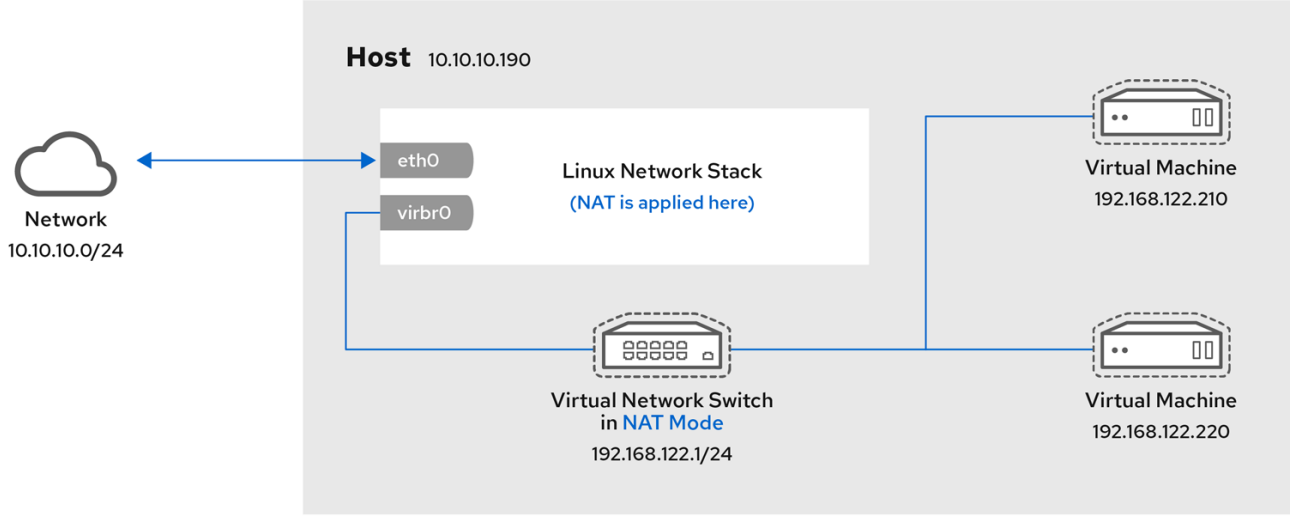
- 使用命令行界面配置外部可见的虚拟机
- 网桥模式中的虚拟网络

17.4. 虚拟机网络连接的类型

要修改虚拟机的网络属性和行为，更改虚拟机使用的虚拟网络或接口类型。以下小节描述了 RHEL 9 中虚拟机可用的连接类型。

17.4.1. 使用网络地址转换进行虚拟联网

默认情况下，虚拟网络交换机运行在网络地址转换(NAT)模式下。它们使用 IP 伪装而不是源-NAT(SNAT)或目的-NAT(DNAT)。IP 伪装可让连接的虚拟机使用主机机器的 IP 地址与任何外部网络通信。当虚拟网络交换机运行在 NAT 模式时，主机外的计算机无法与主机内的虚拟机进行通信。



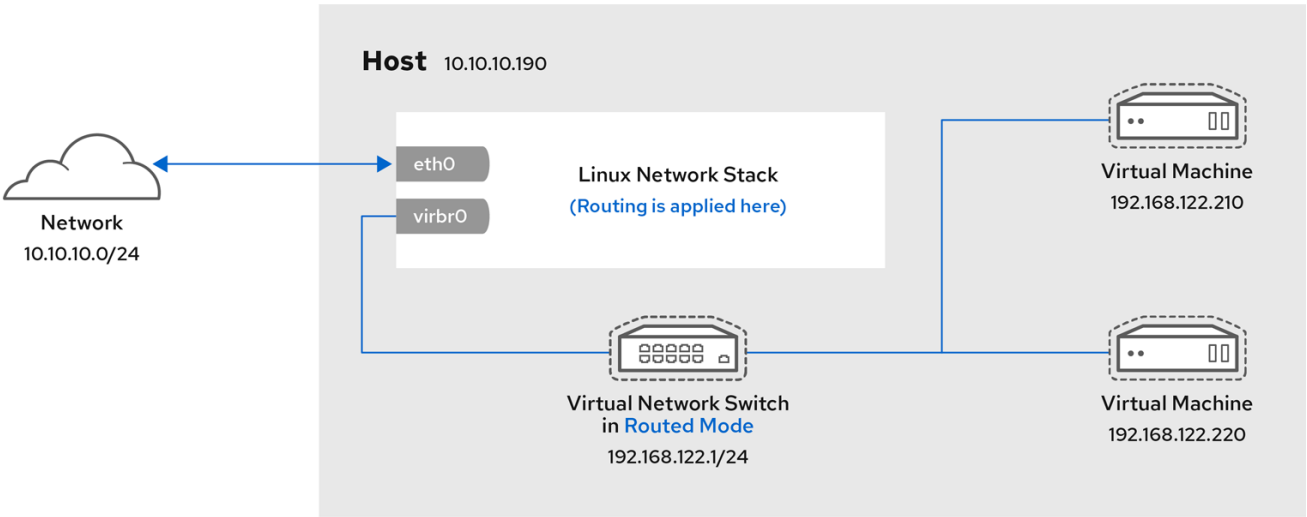
警告

虚拟网络交换机使用防火墙规则配置的 NAT。不建议在交换机运行时编辑这些规则，因为不正确的规则可能会导致交换机无法进行通信。

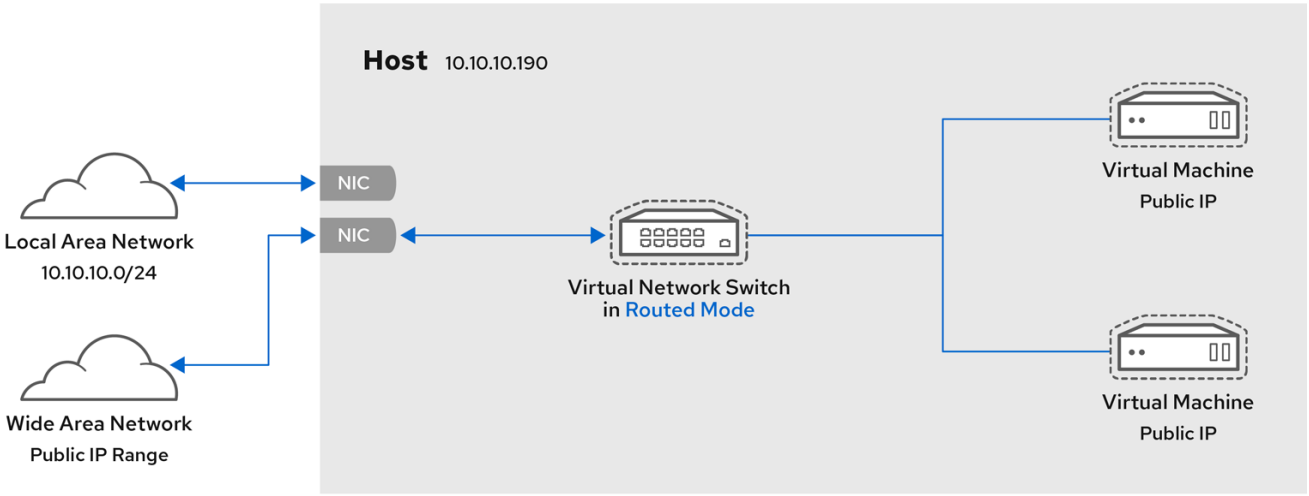
17.4.2. 路由模式中的虚拟网络

当使用 *Routed* 模式时，虚拟交换机会连接到连接到主机的物理 LAN 中，并在不使用 NAT 的情况下传输数据。虚拟交换机可以检查所有流量，并使用网络数据包中包含的信息来做出路由决策。使用此模式时，虚拟机(VM)都位于一个子网中，与主机是分开的。虚拟机子网通过虚拟交换机进行路由，该交换机位于主机上。这会启用传入连接，但对外部网络上的系统需要额外的路由表条目。

路由模式使用基于 IP 地址的路由：



使用路由模式的通用拓扑是虚拟服务器托管 (VSH)。VSH 提供程序可能有多个主机计算机，每个都有两个物理网络连接。一个接口用于管理和记帐，另一个用于虚拟机连接。每个虚拟机都有自己的公共 IP 地址，但主机使用私有的 IP 地址，以便只有内部管理员可以管理虚拟机。

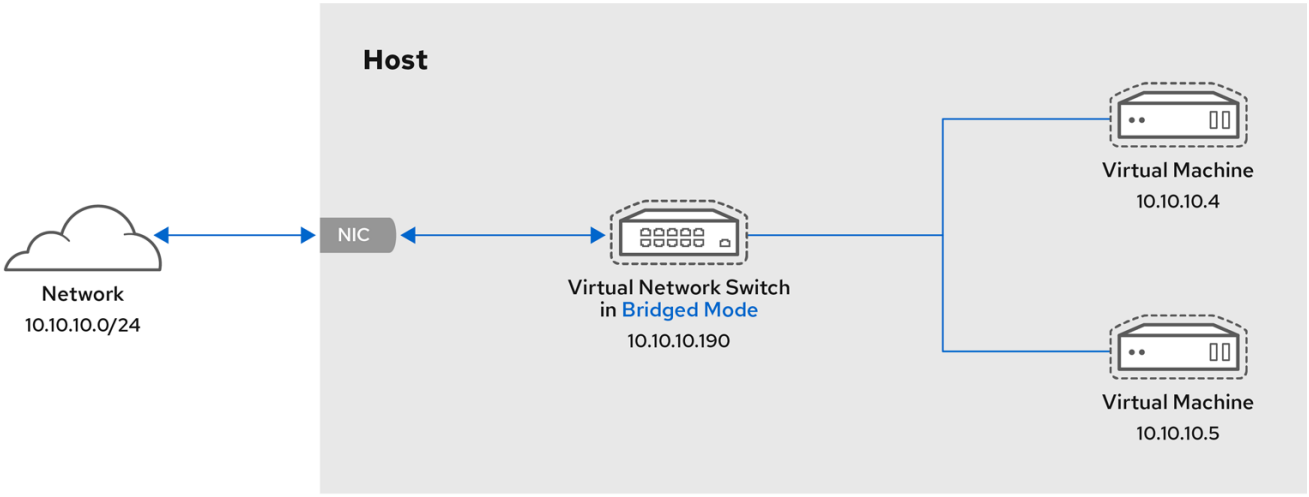


93_RHEL_0520

17.4.3. 网桥模式中的虚拟网络

在大多数虚拟机网络模式中，虚拟机会自动创建并连接到 **virbr0** 虚拟网桥。相反，在*桥接*模式中，VM 会连接到主机上的一个已存在的 Linux 网桥。因此，虚拟机可以在物理网络中直接看到。这就可以允许进入的连接，但不需要任何额外的路由表条目。

网桥模式使用基于 MAC 地址的连接切换：



RHEL_52_1219

在桥接模式中，虚拟机出现在与主机相同的子网中。同一物理网络中的所有其他物理机器都可以检测虚拟机并访问它。

网桥网络绑定

通过使用绑定将多个物理网桥接口连在一起，可以在 hypervisor 上使用多个物理网桥接口。然后，可以将绑定添加到网桥，之后也可以将虚拟机添加到网桥。但是，绑定驱动程序有多种操作模式，而且并非所有这些模式都能与虚拟机正在使用的桥接一起工作。

以下 绑定模式 很有用：

- 模式 1
- 模式 2
- 模式 4

相反，模式 0、3、5 或 6 可能会导致连接失败。另请注意，应使用独立于介质的接口(MII)监控来监控绑定模式，因为地址解析协议(ARP)监控无法正常工作。

有关绑定模式的详情，请参考 红帽知识库。

常见情况

使用桥接模式的最常见用例包括：

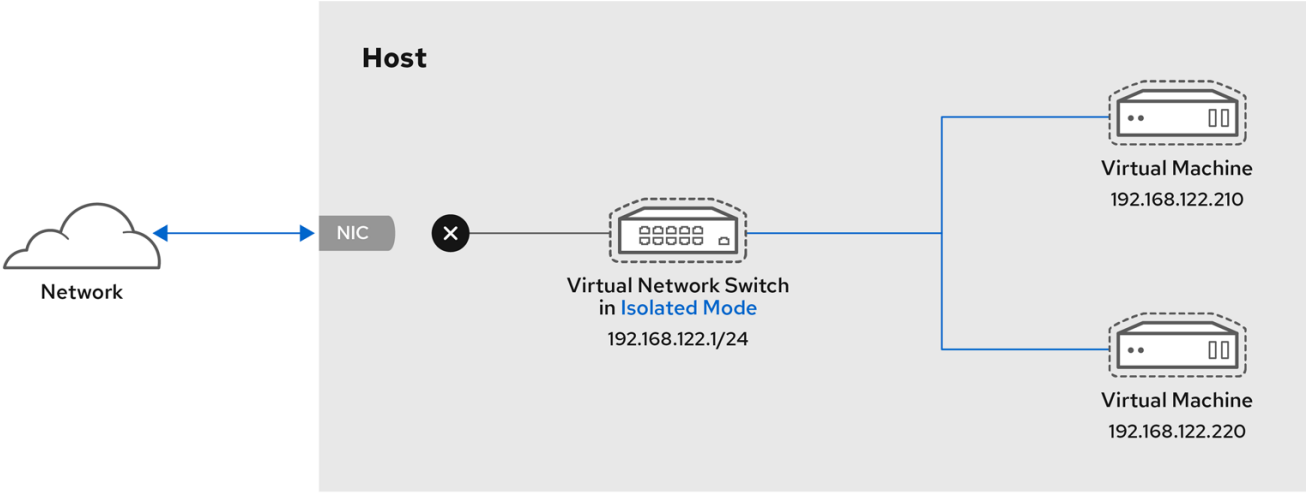
- 主机机器和虚拟机一起出现在现有网络中，最终用户看不到虚拟机和物理机器之间的不同。
- 在不更改现有物理网络配置设置的情况下部署虚拟机。
- 部署需要被现有物理网络轻松访问的虚拟机。将虚拟机放置到必须访问 DHCP 服务的物理网络中。
- 将虚拟机连接到使用虚拟 LAN(VLAN)的现有网络。
- 一个 demilitarized zone (DMZ) 网络。对于带有虚拟机的 DMZ 部署，红帽建议在物理网络路由器和交换机上设置 DMZ，并使用桥接模式将虚拟机连接到物理网络。

其它资源

- 使用命令行界面配置外部可见的虚拟机
- 使用 web 控制台配置外部可见的虚拟机
- **bridge_opts** 参数的说明

17.4.4. 隔离模式中的虚拟网络

通过使用 *isolated* 模式，连接到虚拟交换机的虚拟机可以相互通信，并与主机机器进行通信，但其流量不会通过主机外部，且不能接收来自主机外部的流量。对于基本功能，需要在这个模式中使用 **dnsmasq**，如 DHCP。



17.4.5. 开放模式中的虚拟网络

当对网络使用 *开放* 模式时，**libvirt**不会为网络生成任何防火墙规则。因此，**libvirt** 不会覆盖主机提供的防火墙规则，因此用户可以手动管理虚拟机的防火墙规则。

17.4.6. 虚拟机连接类型的比较

下表提供了有关所选的虚拟机(VM)网络配置可以连接到的位置的信息，以及它们对哪些位置可见。

表 17.1. 虚拟机连接类型

	连接到主机	连接到主机上的其他虚拟机	连接到外部位置	可查看外部位置
网桥模式	是	是	是	是
NAT	是	是	是	*否*
路由模式	是	是	是	是
隔离模式	是	是	*否*	*否*
开放模式	*取决于主机的防火墙规则*			

17.5. 从 PXE 服务器启动虚拟机

使用预引导执行环境(PXE)的虚拟机可以从网络引导并加载它们的配置。本章描述了如何使用 **libvirt** 在虚拟或桥接网络上从 PXE 服务器引导虚拟机。

警告

这些流程仅作为示例提供。在操作之前，请确保您有足够的备份。

17.5.1. 在虚拟网络中设置 PXE 引导服务器

这个流程描述了如何配置 **libvirt** 虚拟网络以提供预启动执行环境(PXE)。这可让主机上的虚拟机被配置为从虚拟网络上提供的引导镜像引导。

先决条件

- 本地 PXE 服务器（DHCP 和 TFTP），例如：
  - libvirt 内部服务器
  - 手动配置 dhcpd 和 tftpd
  - dnsmasq

- Cobbler 服务器

- PXE 引导映像，如 Cobbler 配置的 **PXELINUX** 或手工进行配置。

流程

1. 将 PXE 引导镜像和配置放在 **/var/lib/tftpboot** 文件夹中。
2. 设置文件夹权限：

```
# chmod -R a+r /var/lib/tftpboot
```

3. 设置文件夹所有权：

```
# chown -R nobody: /var/lib/tftpboot
```

4. 更新 SELinux 上下文：

```
# chcon -R --reference /usr/sbin/dnsmasq /var/lib/tftpboot
# chcon -R --reference /usr/libexec/libvirt_leaseshelper /var/lib/tftpboot
```

5. 关闭虚拟网络：

```
# virsh net-destroy default
```

6. 在默认编辑器中打开虚拟网络配置文件：

```
# virsh net-edit default
```

7. 编辑 **<ip>** 元素，来包含合适的地址、网络掩码、DHCP 地址范围和引导文件，其中 *example-pxelinux* 是引导镜像文件的名称。

```
<ip address='192.0.2.1' netmask='255.255.255.0'>
  <tftp root='/var/lib/tftpboot'/>
  <dhcp>
    <range start='192.0.2.2' end='192.0.2.254' />
    <bootp file='example-pxelinux'/>
  </dhcp>
</ip>
```

8. 启动虚拟网络：

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

验证

- 验证 **default** 虚拟网络是否处于活跃状态：

```
# virsh net-list
Name           State    Autostart   Persistent
----------------------------------------------------
default        active   no          no
```

其他资源

- 使用 PXE 准备从网络安装

17.5.2. 使用 PXE 和虚拟网络引导虚拟机

要从虚拟网络上的预引导执行环境(PXE)服务器引导虚拟机(VM)，您必须启用 PXE 引导。

先决条件

- 在虚拟网络上设置 PXE 引导服务器，如在虚拟网络中设置 PXE 引导服务器所述。

流程

- 创建启用了 PXE 引导的新虚拟机。例如，若要从 **default** 虚拟网络上提供的 PXE 安装一个新的 10 GB qcow2 镜像文件：

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

  - 另外，您可以手动编辑现有虚拟机的 XML 配置文件：

    i. 确保 **<os>** 元素中有一个 **<boot dev='network'/>** 元素：

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

    ii. 确定客户端网络已配置为使用您的虚拟网络：

```
<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>
```

验证

- 使用 **virsh start** 命令启动虚拟机。如果正确配置了 PXE，则虚拟机可以从 PXE 服务器上提供的引导镜像启动。

17.5.3. 使用 PXE 和桥接网络启动虚拟机

要从桥接网络上提供的预引导执行环境(PXE)服务器启动虚拟机(VM)，您必须启用 PXE 引导。

先决条件

- 启用了网络桥接。
- 网桥网络上提供了 PXE 引导服务器。

流程

- 创建启用了 PXE 引导的新虚拟机。例如，若要在 **breth0** 桥接网络上从 PXE 安装一个新的 10 GB qcow2 镜像文件：

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

  - 另外，您可以手动编辑现有虚拟机的 XML 配置文件：

    i. 确保 **<os>** 元素中有一个 **<boot dev='network'/>** 元素：

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

    ii. 确保虚拟机配置为使用桥接网络：

```
<interface type='bridge'>
  <mac address='52:54:00:5a:ad:cb'/>
  <source bridge='breth0'/>
  <target dev='vnet0'/>
  <alias name='net0'/>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
</interface>
```

验证

- 使用 **virsh start** 命令启动虚拟机。如果正确配置了 PXE，则虚拟机可以从 PXE 服务器上提供的引导镜像启动。

其它资源

- 配置网络桥接

17.6. 配置 PASST 用户空间连接

如果您需要对虚拟网络的非特权访问，例如当使用 **libvirt** 的 **会话** 连接时，您可以将虚拟机(VM)配置为使用 **passt** 网络后端。

先决条件

- **passt** 软件包已安装在您的系统上。

```
# dnf install passt
```

流程

1. 打开您要在其上使用 **passt** 连接的虚拟机的 XML 配置。例如：

```
# virsh edit <testguest1>
```

2. 在 **<devices>** 部分中，添加一个使用 **passt** 作为其后端类型的 **<interface type='user'>** 元素。例如，以下配置设置一个 **passt** 连接，它使用从与第一个默认路由关联的主机接口复制的地址和路由：

```
<devices>
 [...]
 <interface type='user'>
  <backend type='passt'/>
 </interface>
</devices>
```

另外，在使用 **passt** 时，您可以指定多个 **<portForward>** 元素，来将主机的传入网络流量转发到此虚拟机接口。您还可以自定义接口 IP 地址。例如：

```
<devices>
 [...]
 <interface type='user'>
  <backend type='passt'/>
  <mac address="52:54:00:98:d8:b7"/>
  <source dev='eth0'/>
  <ip family='ipv4' address='192.0.2.1' prefix='24'/>
  <ip family='ipv6' address='::ffff:c000:201'/>
  <portForward proto='tcp'>
   <range start='2022' to='22'/>
  </portForward>
  <portForward proto='udp' address='1.2.3.4'>
    <range start='5000' end='5020' to='6000'/>
    <range start='5010' end='5015' exclude='yes'/>
  </portForward>
  <portForward proto='tcp' address='2001:db8:ac10:fd01::1:10' dev='eth0'>
   <range start='8080'/>
   <range start='4433' to='3444'/>
  </portForward>
 </interface>
</devices>
```

这个示例配置建立了一个带有以下参数的 **passt** 连接：

- 虚拟机为转发来自 **eth0** 主机接口的流量复制网络路由。
- 接口 MAC 设置为 **52:54:00:98:d8:b7**。如果未设置，将生成一个随机的。
- IPv4 地址设为 **192.0.2.1/24**，IPv6 地址设为 **::ffff:c000:201**。
- 主机上的 TCP 端口 **2022** 将网络流量转发到虚拟机上的端口 **22**。
- 主机接口 **eth0** 上的 TCP 地址 **2001:db8:ac10:fd01::1:10**和端口 **8080** 将其网络流量转发到虚拟机上的端口 **8080**。端口 **4433** 转发到虚拟机上的端口 **3444**。
- 主机上的 UDP 地址 **1.2.3.4** 和端口 **5000 - 5009** 以及 **5016 - 5020** 将其网络流量转发到虚拟机上的端口 **6000 - 6009** 和 **6016 - 6020**。

3. 保存 XML 配置。

验证

- 启动或重启使用 **passt** 配置的虚拟机：

```
# virsh reboot <vm-name>
# virsh start <vm-name>
```

如果虚拟机成功引导，则现在使用 **passt** 网络后端。

其它资源

- **libvirt**中的系统和会话连接

17.7. 其它资源

- 配置和管理网络
- 将特定的网络接口卡作为 SR-IOV 设备 来提高虚拟机性能。

第 18 章 优化虚拟机性能

与主机相比，虚拟机的性能总会有所降低。以下小节解释了这个冲突的原因，并提供了有关如何在 RHEL 9 中最小化虚拟化性能影响的说明，以便您的硬件基础架构资源可以尽可能高效地使用。

18.1. 影响虚拟机性能的因素

虚拟机作为用户空间进程在主机上运行。因此管理程序需要转换主机的系统资源，以便虚拟机可使用它们。因此，部分资源会被转换消耗，因此虚拟机无法获得与主机相同的性能效率。

虚拟化对系统性能的影响
体虚拟机性能损失的原因包括：

- 虚拟 CPU（vCPU）是主机上的线，,由 Linux 调度程序处理。
- VM 不会自动继承主机内核的优化功能，比如 NUMA 或巨页。
- 主机的磁盘和网络 I/O 设置可能会对虚拟机有显著的性能影响。
- 网络流量通常通过基于软件的网桥到达虚拟机。
- 根据主机设备及其型号，特定硬件的模拟可能会产生大量的开销。

虚拟化对虚拟机性能影响的严重程度受到各种因素的影响，具体包括：

- 并行运行的虚拟机数量。
- 每个虚拟机使用的虚拟设备数量。
- 虚拟机使用的设备类型。

降低虚拟机性能损失
RHEL 9 提供了很多功能，可用于降低虚拟化的负面影响。值得注意的是：

- **TuneD** 服务 可以自动优化虚拟机的资源分布和性能。
- 块 I/O 调优 可以提高虚拟机块设备（如磁盘）的性能。
- NUMA 调优 可以提高 vCPU 的性能。
- 可以通过多种方式优化 虚拟网络。

重要

调整虚拟机性能会对其他虚拟化功能造成负面影响。例如，它可以使迁移修改过的虚拟机更为困难。

18.2. 使用 TUNED 优化虚拟机性能

TuneD 工具是一种调优配置文件交付机制，能够让 RHEL 适应某些工作负载特性，如 CPU 密集型任务或存储网络吞吐量响应的需求。它提供很多预先配置的调优配置文件，以便在多个特定用例中增强性能并降低功耗。您可以编辑这些配置集，或创建新配置集来创建适合您的环境的性能解决方案，包括虚拟环境。

要针对虚拟化优化 RHEL 9，请使用以下配置集：

- 对于 RHEL 9 虚拟机，请使用 **virtual-guest** 配置集。它基于普遍适用的 ***吞吐量性能*** 配置文件，但也减少了虚拟内存的交换。
- 对于 RHEL 9 虚拟化主机，请使用 **virtual-host** 配置集。这可提高脏内存页面的主动回写，这有助于主机性能。

先决条件

- **TuneD** 服务已安装并启用。

流程

启用特定的 **TuneD** 配置集：

1. 列出可用的 **TuneD** 配置集。

```
# 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. **可选：**创建一个新的 **TuneD** 配置集或编辑现有的 **TuneD** 配置集。
如需更多信息，请参阅自定义 TuneD 配置集。

3. 激活 **TuneD** 配置集。

```
# tuned-adm profile selected-profile
```

- 要优化虚拟化主机，请使用 *virtual-host* 配置集。

```
# tuned-adm profile virtual-host
```

- 在 RHEL 虚拟机操作系统中，使用 *virtual-guest* 配置集。

```
# tuned-adm profile virtual-guest
```

其它资源

- 监控和管理系统状态和性能

18.3. 优化 LIBVIRT 守护进程

libvirt 虚拟化套件作为 RHEL hypervisor 的管理层， **libvirt** 配置对您的虚拟化主机有很大影响。值得注意的是，RHEL 9 包含两种不同类型的 **libvirt** 守护进程，即单体或模块化，您使用的守护进程类型会影响您可以配置单独的虚拟化驱动程序。

18.3.1. libvirt 守护进程的类型

RHEL 9 支持以下 **libvirt** 守护进程类型：

单体 libvirt

传统的 **libvirt** 守护进程 **libvirtd** 通过使用单个配置文件 - **/etc/libvirt/libvirtd.conf** 控制各种虚拟化驱动程序。
因此，**libvirtd** 允许使用集中的虚拟机监控程序配置，但可能会导致对系统资源的使用效率低。因此，**libvirtd** 在以后的 RHEL 主发行版本中将不被支持。

但是，如果您从 RHEL 8 更新至 RHEL 9，您的主机仍然默认使用 **libvirtd**。

模块 libvirt

RHEL 9 中新引入的模块 **libvirt** 为各个虚拟化驱动程序提供一个特定的守护进程。其中包括：

- **virtqemud** - hypervisor 管理的主要守护进程
- **virtinterfaced** - 用于主机 NIC 管理的辅助守护进程
- **virtnetworkd** - 用于虚拟网络管理的辅助守护进程
- **virtnodedevd** - 主机物理设备管理的辅助守护进程
- **virtnwfilterd** - 主机防火墙管理的辅助守护进程
- **virtsecretd** - 用于主机 secret 管理的辅助守护进程
- **virtstoraged** - 用于存储管理的辅助守护进程

每个守护进程都有单独的配置文件 - 例如 **/etc/libvirt/virtqemud.conf**。因此，模块化的 **libvirt** 守护进程可以为调优 **libvirt** 资源管理提供更好的选项。

如果您执行了全新的 RHEL 9 安装，则会默认配置模块化的 **libvirt**。

后续步骤

- 如果您的 RHEL 9 使用 **libvirtd**，红帽建议切换到模块化守护进程。具体步骤请参阅启用模块化 libvirt 守护进程。

18.3.2. 启用模块化 libvirt 守护进程

在 RHEL 9 中，**libvirt** 库使用 modular 守护进程来处理您主机上的单个虚拟化驱动程序集。例如，**virtqemud** 守护进程处理 QEMU 驱动程序。

如果您执行了 RHEL 9 主机的全新安装，您的虚拟机监控程序默认使用模块化 **libvirt** 守护进程。但是，如果您将主机从 RHEL 8 升级到 RHEL 9，您的管理程序将使用单体 **libvirtd** 守护进程，这是 RHEL 8 中的默认设置。

如果是这种情况，红帽建议改为启用模块 **libvirt** 守护进程，因为它们为微调 **libvirt** 资源管理提供了更好的选项。另外，**libvirtd** 在以后的 RHEL 主发行版本中将不被支持。

先决条件

- 您的管理程序使用单体的 **libvirtd** 服务。

```
# systemctl is-active libvirtd.service
active
```

如果这个命令显示 **active**，则代表在使用 **libvirtd**。

- 您的虚拟机已关闭。

流程

1. 停止 **libvirtd** 及其套接字。

```
$ systemctl stop libvirtd.service
$ systemctl stop libvirtd{,-ro,-admin,-tcp,-tls}.socket
```

2. 禁用 **libvirtd** 以防止它在引导时启动。

```
$ systemctl disable libvirtd.service
$ systemctl disable libvirtd{,-ro,-admin,-tcp,-tls}.socket
```

3. 启用模块 **libvirt** 守护进程。

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

4. 启动模块守护进程的套接字。

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

5. **可选：** 如果您需要从远程主机连接到主机，请启用并启动虚拟化代理守护进程。

   a. 检查系统上是否启用了 **libvirtd-tls.socket** 服务。

```
# grep listen_tls /etc/libvirt/libvirtd.conf

listen_tls = 0
```

   b. 如果没有启用 **libvirtd-tls.socket** (**listen_tls = 0**)，请激活 **virtproxyd**，如下所示：

```
# systemctl unmask virtproxyd.service
# systemctl unmask virtproxyd{,-ro,-admin}.socket
# systemctl enable virtproxyd.service
# systemctl enable virtproxyd{,-ro,-admin}.socket
# systemctl start virtproxyd{,-ro,-admin}.socket
```

   c. 如果启用了 **libvirtd-tls.socket** (**listen_tls =**1)，请激活 **virtproxyd**，如下所示：

```
# systemctl unmask virtproxyd.service
# systemctl unmask virtproxyd{,-ro,-admin,-tls}.socket
# systemctl enable virtproxyd.service
# systemctl enable virtproxyd{,-ro,-admin,-tls}.socket
# systemctl start virtproxyd{,-ro,-admin,-tls}.socket
```

   要启用 **virtproxyd** 的 TLS 套接字，您的主机必须有配置的 TLS 证书才能使用 **libvirt**。如需更多信息，请参阅 上游 libvirt 文档。

验证

1. 激活已启用的虚拟化守护进程。

```
# virsh uri
qemu:///system
```

2. 验证您的主机是否使用 **virtqemud** 模块守护进程。

```
# systemctl is-active virtqemud.service
active
```

如果状态为 **active**，则代表您已成功启用了模块 **libvirt** 守护进程。

18.4. 配置虚拟机内存

要提高虚拟机(VM)的性能，您可以将额外的主机 RAM 分配给虚拟机。类似地，您可以减少分配给虚拟机的内存量，从而使主机内存可以分配给其他虚拟机或任务。

要执行这些操作，您可以使用 Web 控制台 或 命令行界面。

18.4.1. 使用 web 控制台添加和删除虚拟机内存

要提高虚拟机(VM)的性能或释放它使用的主机资源，您可以使用 Web 控制台来调整分配给虚拟机的内存量。

先决条件

- 客户端操作系统正在运行内存 balloon 驱动程序。请执行以下命令校验：

  1. 确保虚拟机的配置包含 **memballoon** 设备：

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

  如果此命令显示任何输出，并且型号未设置为 **none**，则存在 **memballoon** 设备。

  2. 确保 balloon 驱动程序在客户机操作系统中运行。

     - 在 Windows 客户机中，驱动程序作为 **virtio-win** 驱动程序软件包的一部分安装。具体步骤请查看为 Windows 虚拟机安装 KVM 半虚拟驱动程序。
     - 在 Linux 客户机中，通常默认包含驱动程序，并在存在 **memballoon** 设备时激活。

- Web 控制台 VM 插件 已安装在您的系统上。

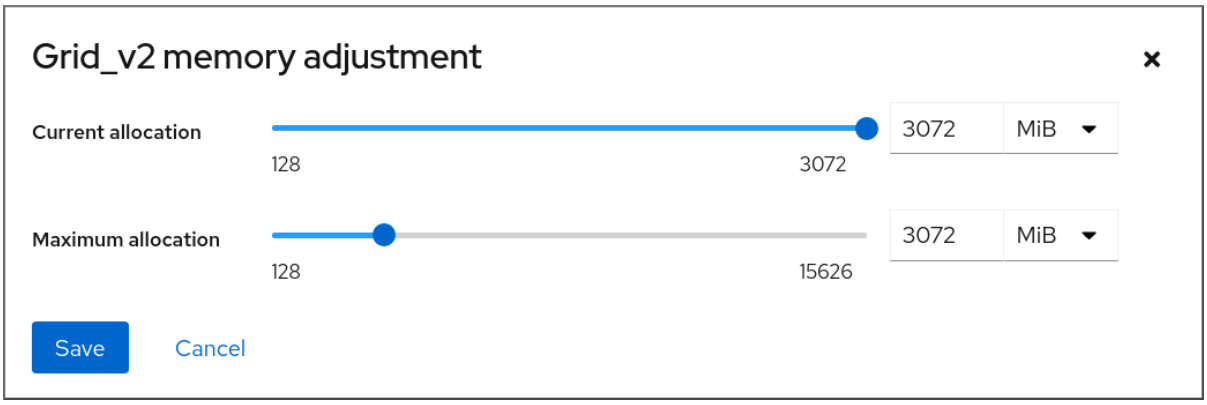
流程

1. **可选：**包含有关虚拟机最大内存和当前使用的内存的信息。这将作为您更改的基准，并进行验证。

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

2. 在 Virtual Machines 界面中，点击您要查看其信息的虚拟机。
此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟机的图形界面的控制台部分。

3. 单击概述窗格中 **Memory** 行旁边的 **edit**。
此时将显示 **Memory Adjustment** 对话框。



4. 为所选的虚拟机配置虚拟内存。

- **最大分配** - 设置虚拟机可用于其进程的最大主机内存量。您可以在创建虚拟机时指定最大内存，或可以在以后增大。您可以将内存指定为 MiB 或 GiB 的倍数。
只有在关闭虚拟机上才能调整最大内存分配。

- **当前分配** - 设置分配给虚拟机的实际内存量。这个值可以小于最大分配量，但不能超过它。您可以调整值，来控制虚拟机的进程可使用的内存。您可以将内存指定为 MiB 或 GiB 的倍数。
如果没有指定这个值，则默认分配是 **Maximum allocation** 值。

5. 点击 **Save**。
调整了虚拟机的内存分配。

其它资源

- 使用命令行界面添加和删除虚拟机内存
- 优化虚拟机 CPU 性能

18.4.2. 使用命令行界面添加和删除虚拟机内存

若要提高虚拟机(VM)的性能或释放其使用的主机资源，您可以使用 CLI 来调整分配给虚拟机的内存量。

先决条件

- 客户端操作系统正在运行内存 balloon 驱动程序。请执行以下命令校验：

1. 确保虚拟机的配置包含 **memballoon** 设备：

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

如果此命令显示任何输出，并且型号未设置为 **none**，则存在 **memballoon** 设备。

2. 确定 ballon 驱动程序正在客户端操作系统中运行。

   - 在 Windows 客户机中，驱动程序作为 **virtio-win** 驱动程序软件包的一部分安装。具体步骤请查看为 Windows 虚拟机安装 KVM 半虚拟驱动程序。
   - 在 Linux 客户机中，通常默认包含驱动程序，并在存在 **memballoon** 设备时激活。

流程

1. **可选：**包含有关虚拟机最大内存和当前使用的内存的信息。这将作为您更改的基准，并进行验证。

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

2. 调整分配给虚拟机的最大内存。增加这个值可以提高虚拟机的性能风险，降低这个值会降低虚拟机在主机上的性能占用空间。请注意，此更改只能在关闭的虚拟机上执行，因此调整正在运行的虚拟机需要重新启动才能生效。
例如，将 *testguest* 虚拟机可以使用的最大内存更改为 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.
```

要增加正在运行的虚拟机的最大内存，您可以将内存设备附加到虚拟机。这也被称为**内存热插拔**。详情请参阅将内存设备附加到虚拟机。

警告

不支持从正在运行的虚拟机中删除内存设备（也称为内存热插拔），因此红帽强烈不鼓励这样做。

3. **可选：**您还可以调整虚拟机当前使用的内存，最多不超过最大分配数。这调整了虚拟机在主机上的内存负载，直到下一次重启为止，而不需要更改最大的虚拟机分配。

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

验证

1. 确认虚拟机使用的内存已更新：

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

2. **可选：**如果您调整了当前虚拟机内存，您可以获取虚拟机的内存 balloon 统计，以评估它如何有效地控制其内存使用量。

```
 # 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
```

其它资源

- 使用 web 控制台添加和删除虚拟机内存
- 优化虚拟机 CPU 性能

18.4.3. 使用 virtio-mem 添加和删除虚拟机内存

RHEL 9 提供 **virtio-mem** 半虚拟化内存设备。此设备使得可以在虚拟机(VM)中动态添加或删除主机内存。例如，您可以使用 **virtio-mem** 在正在运行的虚拟机之间移动内存资源，或者根据您的当前要求在云设置中调整虚拟机内存的大小。

18.4.3.1. virtio-mem 概述

virtio-mem 是一个半虚拟化内存设备，可用于在虚拟机中动态添加或删除主机内存。例如，您可以使用这个设备在运行的虚拟机之间移动内存资源，或者根据您当前的要求在云设置中调整虚拟机内存大小。

通过使用 **virtio-mem**，您可以将虚拟机的内存增加到超过其初始大小，并将其缩小回其原始大小，大小可以是 4 到几百兆(MiB)。但请注意，**virtio-mem** 也依赖于特定的客户机操作系统配置，特别是为了可靠地拔出内存。

virtio-mem 功能限制

virtio-mem 目前与以下功能不兼容：

- 对主机上的实时应用程序使用内存锁定
- 在主机上使用加密的虚拟化
- 在主机上将 **virtio-mem** 与 **memballoon** 膨胀和收缩合并
- 在虚拟机中卸载或重新载入 **virtio_mem** 驱动程序
- 使用 vhost-user 设备，但 **virtiofs** 除外

其它资源

- 在虚拟机中配置内存在线

- 将 virtio-mem 设备附加到虚拟机

18.4.3.2. 在虚拟机中配置内存在线

在使用 **virtio-mem** 将内存附加到正在运行的虚拟机（也称为内存热插拔）之前，您必须配置虚拟机(VM)操作系统，以便可将热插拔内存自动设置为在线状态。否则，客户端操作系统无法使用额外的内存。您可以从以下内存在线的配置中选择：

- **online_movable**
- **online_kernel**
- **auto-movable**

要了解这些配置之间的区别，请参阅：内存在线配置比较

RHEL 中默认内存在线是使用 udev 规则配置的。但是，在使用 **virtio-mem** 时，建议直接在内核中配置内存在线。

先决条件

- 主机具有 Intel 64 或 AMD64 CPU 架构。
- 主机使用 RHEL 9.4 或更高版本作为操作系统。
- 主机上运行的虚拟机使用以下操作系统版本之一：
  - RHEL 8.10

    重要

    从正在运行的虚拟机中拔出内存默认在 RHEL 8.10 虚拟机中被禁用。
  - RHEL 9

流程

- 要在虚拟机中设置内存在线，以使用 **online_movable** 配置：
  1. 将 **memhp_default_state** 内核命令行参数设置为 **online_movable** ：

     ```
     # grubby --update-kernel=ALL --remove-args=memhp_default_state --
     args=memhp_default_state=online_movable
     ```

  2. 重启虚拟机。
- 要在虚拟机中设置内存在线，以使用 **online_kernel** 配置：
  1. 将 **memhp_default_state** 内核命令行参数设置为 **online_kernel** ：

     ```
     # grubby --update-kernel=ALL --remove-args=memhp_default_state --
     args=memhp_default_state=online_kernel
     ```

  2. 重启虚拟机。

- 要在虚拟机中使用 **auto-movable** 内存在线策略：

  1. 将 **memhp_default_state** 内核命令行参数设置为 **online** ：

     ```
     # grubby --update-kernel=ALL --remove-args=memhp_default_state --
     args=memhp_default_state=online
     ```

  2. 将 **memory_hotplug.online_policy** 内核命令行参数设置为 **auto-movable** ：

     ```
     # grubby --update-kernel=ALL --remove-args="memory_hotplug.online_policy" --
     args=memory_hotplug.online_policy=auto-movable
     ```

  3. **可选：** 要进一步调整 **auto-movable** 在线策略，请更改 **memory_hotplug.auto_movable_ratio** 和 **memory_hotplug.auto_movable_numa_aware** 参数：

     ```
     # grubby --update-kernel=ALL --remove-args="memory_hotplug.auto_movable_ratio" --
     args=memory_hotplug.auto_movable_ratio=<percentage>

     # grubby --update-kernel=ALL --remove-
     args="memory_hotplug.memory_auto_movable_numa_aware" --
     args=memory_hotplug.auto_movable_numa_aware=<y/n>
     ```

     - 与可用于任何分配的内存相比，**memory_hotplug.auto_movable_ratio 参数** 设置仅可用于可移动分配的最大内存比率。比率以百分比表示，默认值为 *301* (%)，它是 3:1 的比率。

     - **memory_hotplug.auto_movable_numa_aware** 参数控制 **memory_hotplug.auto_movable_ratio** 参数是否应用到跨所有可用 NUMA 节点的内存，或仅应用到单个 NUMA 节点上的内存。默认值为：*y* (yes)
       例如，如果最大比率设置为 301%，并且 **memory_hotplug.auto_movable_numa_aware** 设置为 *y* (yes)，即使在附加了 **virtio-mem** 设备的 NUMA 节点内也应用 3:1 比率。如果参数设置为 *n* (no)，则最大 3:1 比率仅应用于整个 NUMA 节点。

       另外，如果没有超过比率，则新热插拔内存将只可用于可移动分配。否则，新热拔插内存将同时可用于可移动和不可移动分配。

  4. 重启虚拟机。

验证

- 要查看 **online_movable** 配置是否已正确设置，请检查 **memhp_default_state** 内核参数的当前值：

  ```
  # cat /sys/devices/system/memory/auto_online_blocks

  online_movable
  ```

- 要查看 **online_kernel** 配置是否已正确设置，请检查 **memhp_default_state** 内核参数的当前值：

  ```
  # cat /sys/devices/system/memory/auto_online_blocks

  online_kernel
  ```

- 要查看 **auto-movable** 配置是否已正确设置，请检查以下内核参数：
  - **memhp_default_state** ：

    ```
    # cat /sys/devices/system/memory/auto_online_blocks
    online
    ```

  - **memory_hotplug.online_policy**:

    ```
    # cat /sys/module/memory_hotplug/parameters/online_policy
    auto-movable
    ```

  - **memory_hotplug.auto_movable_ratio**:

    ```
    # cat /sys/module/memory_hotplug/parameters/auto_movable_ratio
    301
    ```

  - **memory_hotplug.auto_movable_numa_aware**:

    ```
    # cat /sys/module/memory_hotplug/parameters/auto_movable_numa_aware
    y
    ```

其它资源

- virtio-mem 概述
- 将 virtio-mem 设备附加到虚拟机
- 配置内存热插拔

18.4.3.3. 将 virtio-mem 设备附加到虚拟机

要将额外内存附加到正在运行的虚拟机（也称为内存热插拔），之后能够调整热插拔内存的大小，您可以使用 **virtio-mem** 设备。特别是，您可以使用 libvirt XML 配置文件和 **virsh** 命令来定义并将 **virtio-mem** 设备附加到虚拟机(VM)。

先决条件

- 主机具有 Intel 64 或 AMD64 CPU 架构。
- 主机使用 RHEL 9.4 或更高版本作为操作系统。
- 主机上运行的虚拟机使用以下操作系统版本之一：
  - RHEL 8.10

重要

从正在运行的虚拟机中拔出内存默认在 RHEL 8.10 虚拟机中被禁用。

- RHEL 9

- 虚拟机配置了内存在线。具体说明，请参阅：在虚拟机中配置内存在线

流程

1. 确保目标虚拟机的 XML 配置包含 **maxMemory** 参数：

```
# virsh edit testguest1

<domain type='kvm'>
  <name>testguest1</name>
  ...
  <maxMemory unit='GiB'>128</maxMemory>
  ...
</domain>
```

在本例中，**testguest1** 虚拟机的 XML 配置定义了一个 128 gibibyte (GiB)大小的 **maxMemory** 参数。**maxMemory** 大小指定虚拟机可以使用的最大内存，其包括初始内存和热插拔内存。

2. 创建并打开 XML 文件，来在主机上定义 **virtio-mem** 设备，例如：

```
# vim virtio-mem-device.xml
```

3. 在文件中添加 **virtio-mem** 设备的 XML 定义并保存：

```
<memory model='virtio-mem'>
      <target>
            <size unit='GiB'>48</size>
            <node>0</node>
            <block unit='MiB'>2</block>
            <requested unit='GiB'>16</requested>
            <current unit='GiB'>16</current>
      </target>
      <alias name='ua-virtiomem0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
</memory>
<memory model='virtio-mem'>
      <target>
            <size unit='GiB'>48</size>
            <node>1</node>
            <block unit='MiB'>2</block>
            <requested unit='GiB'>0</requested>
            <current unit='GiB'>0</current>
      </target>
      <alias name='ua-virtiomem1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/>
</memory>
```

在本例中，使用以下参数定义了两个 **virtio-mem** 设备：

- **size** ：这是设备的最大大小。在示例中，它是 48 GiB。 **size** 必须是 **block** 大小的倍数。
- **node** ：这是为 **virtio-mem** 设备分配的 vNUMA 节点。

- **block** ：这是设备的块大小。它必须至少是 Transparent Huge Page (THP)的大小，在 Intel 64 或 AMD64 CPU 架构上它是 2 MiB。Intel 64 或 AMD64 架构上的 2 MiB 块大小通常是好的默认选择。当使用带有 *虚拟功能 I/O(VFIO)* 或 *中介设备(mdev)* 的 **virtio-mem** 时，跨所有 **virtio-mem** 设备的总块数不能大于 32768，否则 RAM 的插入可能会失败。

- **requested** ：这是您附加到具有 **virtio-mem** 设备的虚拟机的内存量。但是，它只对虚拟机的请求，可能无法成功解决，例如如果虚拟机没有正确配置。**requested** 大小必须是 **block** 大小的倍数，且不能超过定义的 **size**最大值。

- **current**：这代表提供给虚拟机的当前 **virtio-mem** 设备的大小。**current** 大小可能与 **requested** 不同，例如当请求无法完成或重启虚拟机时。

- **alias** ：这是一个可选d 用户定义的别名，您可用来指定预期的 **virtio-mem** 设备，例如使用 libvirt 命令编辑设备时。libvirt 中所有用户定义的别名必须以 *"ua-"* 前缀开头。
  除了这些特定的参数外，**libvirt** 像处理任何其他 PCI 设备一样处理 **virtio-mem** 设备。有关管理附加到虚拟机的 PCI 设备的更多信息，请参阅 管理虚拟设备

4. 使用 XML 文件将定义的 **virtio-mem** 设备附加到虚拟机。例如，要将 **virtio-mem-device.xml** 中定义的两个设备永久附加到正在运行的 **testguest1** 虚拟机：

```
# virsh attach-device testguest1 virtio-mem-device.xml --live --config
```

   --live 选项仅将设备附加到正在运行的虚拟机，不会在引导之间保持持久性。**--config** 选项使配置更改持久化。您还可以将设备附加到没有 **--live** 选项的关闭的虚拟机。

5. **可选：** 要动态更改附加到正在运行的虚拟机的 **virtio-mem** 设备的 **requested** 大小，请使用 **virsh update-memory-device** 命令：

```
# virsh update-memory-device testguest1 --alias ua-virtiomem0 --requested-size 4GiB
```

   在本例中：

   - **testguest1** 是您要更新的虚拟机。
   - **--alias ua-virtiomem0** 是之前由定义的别名指定的 **virtio-mem** 设备。
   - **--requested-size 4GiB** 将 **virtio-mem** 设备的 **requested** 大小更改为 4 GiB。

警告

通过减少 **所需的** 大小，从正在运行的虚拟机中拔下内存可能不可靠。此过程是否成功取决于各种因素，如所使用的内存在线策略。

在某些情况下，客户机操作系统无法成功完成请求，因为那时可能无法更改热插拔内存量。

另外，从正在运行的虚拟机中拔出内存默认在 RHEL 8.10 虚拟机中被禁用。

6. **可选** ：要从关闭的虚拟机中拔出 **virtio-mem** 设备，请使用 **virsh detach-device** 命令：

```
# virsh detach-device testguest1 virtio-mem-device.xml
```

7. **可选** ：要从正在运行的虚拟机中拔出 **virtio-mem** 设备：

   a. 将 **virtio-mem** 设备的 **请求的** 大小改为 0，否则尝试从正在运行的虚拟机中拔出 **virtio-mem** 设备将失败。

   ```
   # virsh update-memory-device testguest1 --alias ua-virtiomem0 --requested-size 0
   ```

   b. 从正在运行的虚拟机中拔出 **virtio-mem** 设备：

   ```
   # virsh detach-device testguest1 virtio-mem-device.xml
   ```

验证

- 在虚拟机中，检查可用的 RAM ,并查看总内存现在是否包含热插拔内存：

```
# free -h

               total        used        free      shared  buff/cache   available
Mem:            31Gi       5.5Gi        14Gi       1.3Gi        11Gi        23Gi
Swap:          8.0Gi          0B       8.0Gi
```

```
# numactl -H

available: 1 nodes (0)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 29564 MB
node 0 free: 13351 MB
node distances:
node   0
  0:  10
```

- 也可以通过显示正在运行的虚拟机的 XML 配置来查看主机上当前的插入的 RAM 量：

```
# virsh dumpxml testguest1

<domain type='kvm'>
  <name>testguest1</name>
  ...
  <currentMemory unit='GiB'>31</currentMemory>
  ...
  <memory model='virtio-mem'>
     <target>
       <size unit='GiB'>48</size>
       <node>0</node>
       <block unit='MiB'>2</block>
       <requested unit='GiB'>16</requested>
       <current unit='GiB'>16</current>
     </target>
     <alias name='ua-virtiomem0'/>
     <address type='pci' domain='0x0000' bus='0x08' slot='0x00' function='0x0'/>
  ...
</domain>
```

在本例中：

- **<currentMemory unit='GiB'>31</currentMemory>** 代表虚拟机中所有源的可用 RAM 总量。
- **<current unit='GiB'>16</current>** 代表 **virtio-mem** 设备提供的插入的 RAM 的当前大小。

其他资源

- virtio-mem 概述
- 在虚拟机中配置内存在线

18.4.3.4. 内存在线配置的比较

将内存附加到正在运行的 RHEL 虚拟机（也称为内存热插拔）时，您必须在虚拟机(VM)操作系统将热插内存设置为在线状态。否则，系统将无法使用内存。

下表总结了在可用内存在线配置间选择时的主要注意事项。

表 18.1. 内存在线配置的比较

配置名称	从虚拟机中拔出内存	创建内存区不平衡的风险	一个潜在的用例	预期工作负载的内存要求
online_movable	热插内存可以可靠地拔出。	是	热插拔相对较少的内存	主要是用户空间内存
auto-movable	热插内存的可移动部分可以可靠地拔出。	最小	热插拔大量内存	主要是用户空间内存
online_kernel	热插拔内存无法可靠地拔出。	否	不可靠内存拔出是可以接受的。	用户空间或内核空间内存

区域不平衡 是某个 Linux 内存区域中缺少可用内存页。 *区域不平衡* 会对系统性能造成负面影响。例如，如果内核用完了不可移动分配的可用内存，则内核可能会崩溃。通常，可移动分配主要包含用户空间内存页，不可移动分配主要包含内核空间内存页。

其他资源

- 在线和离线内存块
- 区域不平衡
- 在虚拟机中配置内存在线

18.4.4. 其它资源

- 将设备附加到虚拟机.

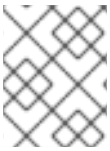
18.5. 优化虚拟机 I/O 性能

虚拟机(VM)的输入和输出(I/O)能力可能会显著限制虚拟机的整体效率。要解决这个问题，您可以通过配置块 I/O 参数来优化虚拟机的 I/O。

18.5.1. 在虚拟机中调整块 I/O

当一个或多个虚拟机正在使用多个块设备时,可能需要通过修改虚拟设备的 I/O 优先级来调整虚拟设备的 *I/O 权重*。

增加设备的 I/O 权重会增加设备的 I/O 带宽的优先级，从而为它提供更多主机资源。同样的，降低设备的权重可使其消耗较少的主机资源。



注意

每个设备的 **weight** 值必须在 **100** 到 **1000** 之间。或者，该值可以是 **0**，它会从每个设备列表中删除该设备。

流程

显示和设置虚拟机的块 I/O 参数：

1. 显示虚拟机当前的 **<blkio>** 参数：
   # virsh dumpxml *VM-name*

```
<domain>
 [...]
 <blkiotune>
  <weight>800</weight>
  <device>
   <path>/dev/sda</path>
   <weight>1000</weight>
  </device>
  <device>
   <path>/dev/sdb</path>
   <weight>500</weight>
  </device>
 </blkiotune>
 [...]
</domain>
```

2. 编辑指定设备的 I/O 加权：

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

例如，以下命令将 *testguest1* 虚拟机中 */dev/sda* 设备的权重改为 500。

```
# virsh blkiotune testguest1 --device-weights /dev/sda, 500
```

18.5.2. 虚拟机中的磁盘 I/O 节流

当多个虚拟机同时运行时，它们可能会使用过量的磁盘 I/O 而干扰系统性能。KVM 虚拟化中的磁盘 I/O 节流使得能够对从虚拟机发送到主机的磁盘 I/O 请求设定限制。这可以防止虚拟机过度使用共享资源并影响其他虚拟机的性能。

要启用磁盘 I/O 节流，请对从附加到虚拟机的每个块设备发送给主机的磁盘 I/O 请求设置限制。

流程

1. 使用 **virsh domblklist** 命令列出指定虚拟机上所有磁盘设备的名称。

```
# virsh domblklist rollin-coal
Target     Source
------------------------------------------------
vda        /var/lib/libvirt/images/rollin-coal.qcow2
sda        -
sdb        /home/horridly-demanding-processes.iso
```

2. 找到您要节流的虚拟磁盘挂载的主机块设备。
例如，如果您想要从上一步中节流 **sdb** 虚拟磁盘，以下输出显示该磁盘挂载在 **/dev/nvme0n1p3** 分区上。

```
$ 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-fxzy1ac12fea 253:0    0 236.9G  0 crypt /home
```

3. 使用 **virsh blkiotune** 命令为块设备设置 I/O 限制。

```
# virsh blkiotune VM-name --parameter device,limit
```

以下示例将 **rollin-coal** 上的 **sdb** 磁盘节流为每秒 1000 个读写 I/O 操作，每秒的读写 I/O 操作吞吐量 50 MB。

```
# 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
```

附加信息

- 磁盘 I/O 节流可用于各种情况，例如，当属于不同客户的虚拟机在同一台主机上运行时，或者为不同的虚拟机提供服务质量保障时。磁盘 I/O 节流还可用来模拟较慢的磁盘。
- I/O 节流可以独立应用于附加到虚拟机的每个块设备，并支持对吞吐量和 I/O 操作的限制。
- 红帽不支持使用 **virsh blkdeviotune**命令来在 VM 中配置 I/O 节流。有关使用 RHEL 9 作为虚拟机主机时不支持的功能的更多信息，请参阅 RHEL 9 虚拟化中不支持的功能。

18.5.3. 启用多队列 virtio-scsi

在虚拟机(VM)中使用 **virtio-scsi** 存储设备时，*multi-queue virtio-scsi* 特性可提高存储性能和可扩展性。它允许每个虚拟 CPU(vCPU)使用单独的队列和中断，而不影响其他 vCPU。

流程

- 要为特定虚拟机启用 multi-queue virtio-scsi 支持，请在虚拟机的 XML 配置中添加以下内容，其中 *N* 是 vCPU 队列的总数：

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

18.6. 优化虚拟机 CPU 性能

与主机中的物理 CPU 非常相似，vCPU 对虚拟机(VM)性能至关重要。因此，优化 vCPU 会对虚拟机的资源效率产生重大影响。优化 vCPU：

1. 调整分配给虚拟机的主机 CPU 数。您可以使用 CLI 或 Web 控制台进行此操作。
2. 确保 vCPU 模型与主机的 CPU 型号一致。例如，将 *testguest1* 虚拟机设置为使用主机的 CPU 型号：

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

   在 ARM 64 系统上，请使用 **--cpu host-passthrough**。
3. 管理内核相同的页面合并 (KSM) 。
4. 如果您的主机使用非统一内存访问 (NUMA)，您也可以为其虚拟机 **配置 NUMA**。这会尽可能将主机的 CPU 和内存进程映射到虚拟机的 CPU 和内存进程上。实际上，NUMA 调优为 vCPU 提供了对分配给虚拟机的系统内存更精简的访问，这可以提高 vCPU 的处理效率。
   详情请参阅 在虚拟机中配置 NUMA 和 vCPU 性能调整场景示例 。

18.6.1. 使用命令行界面添加和删除虚拟 CPU

要提高或优化虚拟机(VM)的 CPU 性能，您可以添加或删除分配给虚拟机的虚拟 CPU(vCPU)。

当在运行的虚拟机上执行时，这也被称为 vCPU 热插和热拔。但请注意，RHEL 9 不支持 vCPU 热拔，红帽不建议使用它。

先决条件

- **可选：**查看目标虚拟机中的 vCPU 的当前状态。例如，显示 *testguest* 虚拟机上的 vCPU 数量：

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

  此输出显示 *testguest* 目前使用 1 个 vCPU，另外 1 个 vCPu 可以热插入以提高虚拟机性能。但是，重新引导后，vCPU *testguest* 使用的数量会改为 2，而且能够热插 2 个 vCPU。

流程

1. 调整可以附加到虚拟机的最大 vCPU 数量，其在虚拟机下次启动时生效。
   例如，要将 *testguest* 虚拟机的最大 vCPU 数量增加到 8:

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

   请注意，最大值可能会受 CPU 拓扑、主机硬件、hypervisor 和其他因素的限制。

2. 将当前附加到虚拟机的 vCPU 数量调整到上一步中配置的最大值。例如：

   - 将附加到正在运行的 *testguest* 虚拟机的 vCPU 数量增加到 4:

     ```
     # virsh setvcpus testguest 4 --live
     ```

     这会增加虚拟机的性能和主机的 *testguest* 负载占用，直到虚拟机下次引导为止。

   - 将附加到 *testguest* 虚拟机的 vCPU 数量永久减少至 1：

     ```
     # virsh setvcpus testguest 1 --config
     ```

     这会降低虚拟机的性能和 *testguest* 的主机负载占用。但是，如果需要可热插入虚拟机以暂时提高性能。

验证

- 确认虚拟机的 vCPU 的当前状态反映了您的更改。

```
# virsh vcpucount testguest
maximum      config         8
maximum      live           4
current      config         1
current      live           4
```

其它资源

- 使用 Web 控制台管理虚拟 CPU

18.6.2. 使用 Web 控制台管理虚拟 CPU

通过使用 RHEL 9 web 控制台，您可以查看并配置 web 控制台连接的虚拟机所使用的虚拟 CPU。

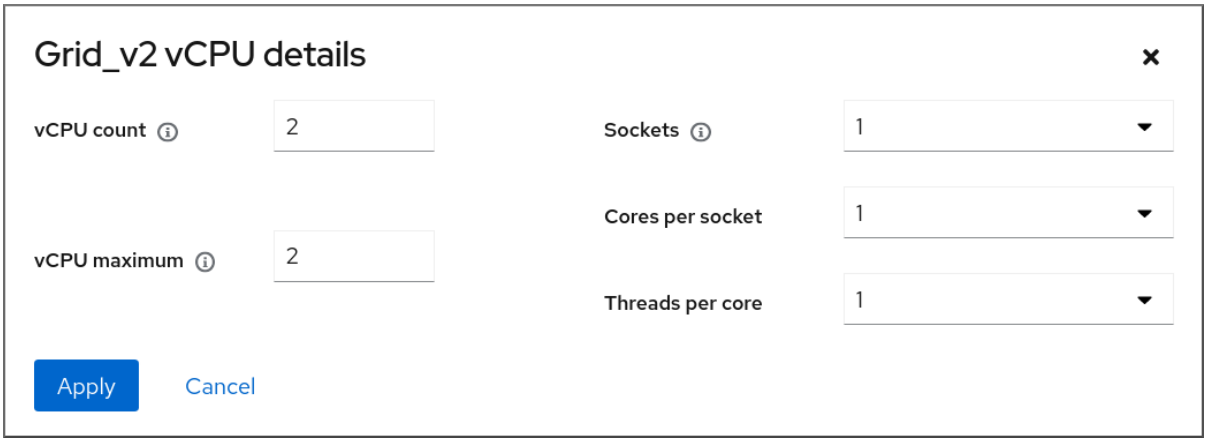
先决条件

- Web 控制台 VM 插件 已安装在您的系统上。

流程

1. 在 **Virtual Machines** 界面中，点击您要查看其信息的虚拟机。
   此时将打开一个新页面，其中包含关于所选虚拟机基本信息的概述部分，以及用于访问虚拟机的图形界面的控制台部分。

2. 单击概述窗格中 vCPU 数旁边的 **edit**。
   此时会出现 vCPU 详情对话框。



1. 为所选虚拟机配置虚拟 CPU。

   - **vCPU 数量** - 当前正在使用的 vCPU 数量。

     注意

     vCPU 数量不能超过 vCPU 的最大值。

   - **vCPU 最大** - 可为虚拟机配置的最大虚拟 CPU 数。如果这个值大于 **vCPU Count**，可以为虚拟机附加额外的 vCPU。
   - **插槽** - 向虚拟机公开的插槽数量。
   - **每个插槽的内核数** - 向虚拟机公开的每个插槽的内核数。
   - **每个内核的线程数** - 向虚拟机公开的每个内核的线程数。
     请注意，**插槽**、**每个插槽的内核数**和**每个内核的线程数**选项调整了虚拟机的 CPU 拓扑。这可能对 vCPU 性能有好处，但可能会影响客户机操作系统中某些软件的功能。如果您的部署不需要不同的设置，请保留默认值。

2. 点**应用**。
   配置了虚拟机的虚拟 CPU。

   注意

   对虚拟 CPU 设置的更改仅在重启虚拟机后生效。

其它资源

- 使用命令行界面添加和删除虚拟 CPU

18.6.3. 在虚拟机中配置 NUMA

以下方法可用于在 RHEL 9 主机上配置虚拟机 (VM) 的非一致性内存访问 (NUMA) 设置。

先决条件

- 主机是一个与 NUMA 兼容的机器。要检测是否是这种情况，请使用 **virsh nodeinfo** 命令，并查看 **NUMA cell(s)** 行：

```
# 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
```

如果行的值为 2 或更高，则主机与 NUMA 兼容。

流程

为便于使用，您可以使用自动化工具和服务设置虚拟机的 NUMA 配置。但是，手动 NUMA 设置可能会显著提高性能。

自动方法

- 将虚拟机的 NUMA 策略设为 **Preferred**。例如，对于 *testguest5* 虚拟机要这样做：

```
# virt-xml testguest5 --edit --vcpus placement=auto
# virt-xml testguest5 --edit --numatune mode=preferred
```

- 在主机上启用自动 NUMA 均衡：

```
# echo 1 > /proc/sys/kernel/numa_balancing
```

- 启动 **numad** 服务，来自动将 VM CPU 与内存资源对齐。

```
# systemctl start numad
```

手动方法

1. 将特定 vCPU 线程固定到特定主机 CPU 或者 CPU 范围。在非 NUMA 主机和虚拟机上也可以这样做，我们推荐您使用一种安全的方法来提高 vCPU 性能。
例如，以下命令将 *testguest6* 虚拟机的 vCPU 线程 0 到 5 分别固定到主机 CPU 1、3、5、7、9 和 11：

```
# 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
```

之后，您可以验证操作是否成功：

```
# virsh vcpupin testguest6
VCPU   CPU Affinity
----------------------
0      1
1      3
2      5
```

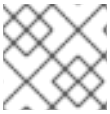
```
3     7
4     9
5     11
```

2. 固定 vCPU 线程后，您还可以将与指定虚拟机关联的 QEMU 进程线程固定到特定的主机 CPU 或 CPU 范围。例如：以下命令将 *testguest6* 的 QEMU 进程线程 固定到 CPU 13 和 15，确认成功：

```
# virsh emulatorpin testguest6 13,15
# virsh emulatorpin testguest6
emulator: CPU Affinity
----------------------------------
       *: 13,15
```

3. 最后，您也可以指定将哪些主机 NUMA 节点专门分配给某个虚拟机。这可提高虚拟机 vCPU 的主机内存用量。例如，以下命令将 *testguest6* 设置为使用主机 NUMA 节点 3 到 5，确认成功：

```
# virsh numatune testguest6 --nodeset 3-5
# virsh numatune testguest6
```



注意

为了获得最佳性能，建议使用以上列出的所有手动调优方法

已知问题

- 目前无法在 IBM Z 主机上执行 NUMA 调整 。

其他资源

- vCPU 性能调整场景示例
- 使用 **numastat** 工具 查看系统的当前 NUMA 配置

18.6.4. vCPU 性能调整场景示例

要获得最佳 vCPU 性能，红帽建议同时使用手动 **vcpupin**、**emulatorpin** 和 **numatune** 设置，例如在以下场景中。

起始场景

- 您的主机有以下与硬件相关的信息：
  - 2 个 NUMA 节点
  - 每个节点上的 3 个 CPU 内核
  - 每个内核有 2 个线程

  此类机器的 **virsh nodeinfo** 输出类似于：

```
# 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
```

- 您打算将现有的虚拟机修改为有 8 个 vCPU，这意味着单个 NUMA 节点无法容纳它。
  因此，您应该在每个 NUMA 节点上分发 4 个 vCPU，并使 vCPU 拓扑尽可能接近主机拓扑。这意味着，作为给定物理 CPU 的同级线程运行的 vCPU 应该固定到同一核上的主机线程。详情请查看以下*解决方案*:

解决方案

1. 获取有关主机拓扑的信息：

   ```
   # virsh capabilities
   ```

   输出应包含类似如下的部分：

```
<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. **可选：** 使用适用的工具和实用程序测试虚拟机的性能。

3. 在主机上设置并挂载 1 GiB 巨页：

注意

1 GiB 巨页可能在某些架构和配置上不提供，如 ARM 64 主机。

a. 在主机的内核命令行中添加以下行：

```
default_hugepagesz=1G hugepagesz=1G
```

b. 使用以下内容创建 **/etc/systemd/system/hugetlb-gigantic-pages.service** 文件：

```
[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. 使用以下内容创建 **/etc/systemd/hugetlb-reserve-pages.sh** 文件：

```
#!/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
```

这会从 *node1* 保留 4 个 1GiB 巨页，并在 *node2* 中保留 4 个 1GiB 巨页。

d. 使在上一步中创建的脚本可执行：

```
# chmod +x /etc/systemd/hugetlb-reserve-pages.sh
```

e. 在引导时启用巨页保留：

```
# systemctl enable hugetlb-gigantic-pages
```

4. 使用 **virsh edit** 命令编辑您要优化的虚拟机的 XML 配置，在本例中为 *super-VM* ：

```
# virsh edit super-vm
```

5. 用以下方法调整虚拟机的 XML 配置：

a. 将虚拟机设置为使用 8 个静态 vCPU。使用 **<vcpu/>** 元素来执行此操作。

b. 将每个 vCPU 线程固定到拓扑中镜像的对应主机 CPU 线程。为此，请在 **<cputune>** 部分中使用 **<vcpupin/>** 元素。
请注意，如上面 **virsh capabilities** 工具所示，主机 CPU 线程在各自的内核中不是按顺序排序的。此外，vCPU 线程应固定到同一 NUMA 节点上最多可用的主机核集合。有关表图，请查看以下 **示例拓扑** 部分。

步骤 a. 和 b. 的 XML 配置类似：

```
<cputune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='4'/>
  <vcpupin vcpu='2' cpuset='2'/>
  <vcpupin vcpu='3' cpuset='5'/>
  <vcpupin vcpu='4' cpuset='7'/>
  <vcpupin vcpu='5' cpuset='10'/>
  <vcpupin vcpu='6' cpuset='8'/>
  <vcpupin vcpu='7' cpuset='11'/>
  <emulatorpin cpuset='6,9'/>
</cputune>
```

c. 将虚拟机设置为使用 1 GiB 巨页：

```
<memoryBacking>
  <hugepages>
    <page size='1' unit='GiB'/>
  </hugepages>
</memoryBacking>
```

d. 配置虚拟机的 NUMA 节点，使其使用主机上对应的 NUMA 节点的内存。要做到这一点，请在 **<numatune/>** 部分中使用 **<memnode/>** 元素：

```
<numatune>
  <memory mode="preferred" nodeset="1"/>
  <memnode cellid="0" mode="strict" nodeset="0"/>
  <memnode cellid="1" mode="strict" nodeset="1"/>
</numatune>
```

e. 确保 CPU 模式设为 **host-passthrough**，且 CPU 在 **passthrough** 模式下使用缓存：

```
  <cpu mode="host-passthrough">
    <topology sockets="2" cores="2" threads="2"/>
    <cache mode="passthrough"/>
```

在 ARM 64 系统上，省略 **<cache mode="passthrough"/>** 行。

验证

1. 确认生成的虚拟机 XML 配置包含类似如下内容：

```
[...]
  <memoryBacking>
    <hugepages>
      <page size='1' unit='GiB'/>
    </hugepages>
  </memoryBacking>
  <vcpu placement='static'>8</vcpu>
  <cputune>
    <vcpupin vcpu='0' cpuset='1'/>
    <vcpupin vcpu='1' cpuset='4'/>
    <vcpupin vcpu='2' cpuset='2'/>
    <vcpupin vcpu='3' cpuset='5'/>
    <vcpupin vcpu='4' cpuset='7'/>
    <vcpupin vcpu='5' cpuset='10'/>
    <vcpupin vcpu='6' cpuset='8'/>
    <vcpupin vcpu='7' cpuset='11'/>
    <emulatorpin cpuset='6,9'/>
  </cputune>
  <numatune>
    <memory mode="preferred" nodeset="1"/>
    <memnode cellid="0" mode="strict" nodeset="0"/>
    <memnode cellid="1" mode="strict" nodeset="1"/>
  </numatune>
  <cpu mode="host-passthrough">
    <topology sockets="2" cores="2" threads="2"/>
    <cache mode="passthrough"/>
    <numa>
      <cell id="0" cpus="0-3" memory="2" unit="GiB">
        <distances>
          <sibling id="0" value="10"/>
          <sibling id="1" value="21"/>
        </distances>
      </cell>
      <cell id="1" cpus="4-7" memory="2" unit="GiB">
        <distances>
          <sibling id="0" value="21"/>
          <sibling id="1" value="10"/>
        </distances>
      </cell>
    </numa>
  </cpu>
</domain>
```

2. **可选：** 使用 适用的工具和实用程序测试虚拟机的性能，以评估虚拟机优化的影响。

拓扑示例

- 下表演示了 vCPU 和主机 CPU 之间的连接：

表 18.2. 主机拓扑

CPU 线程	0	3	1	4	2	5	6	9	7	10	8	11
内核	0		1		2		3		4		5	
插槽	0						1					
NUMA 节点	0						1					

表 18.3. VM 拓扑

vCPU 线程	0	1	2	3	4	5	6	7
内核	0		1		2		3	
插槽	0				1			
NUMA 节点	0				1			

表 18.4. 合并主机和虚拟机拓扑

vCPU 线程			0	1	2	3			4	5	6	7
主机 CPU 线程	0	3	1	4	2	5	6	9	7	10	8	11
内核	0		1		2		3		4		5	
插槽	0						1					
NUMA 节点	0						1					

在这种情况下，有 2 个 NUMA 节点和 8 个 vCPU。因此，应该为每个节点固定 4 个 vCPU 线程。

另外，红帽建议在每个节点上至少保留一个 CPU 线程用于主机系统操作。

因为在这个示例中，每个 NUMA 节点都有 3 个核，每个核都有 2 个主机 CPU 线程，节点 0 的设置转换如下：

```
<vcpupin vcpu='0' cpuset='1'/>
<vcpupin vcpu='1' cpuset='4'/>
<vcpupin vcpu='2' cpuset='2'/>
<vcpupin vcpu='3' cpuset='5'/>
```

18.6.5. 管理内核相同的页面合并

内核同页合并 (KSM) 通过在虚拟机 (VM) 间共享相同的内存页面来提高内存密度。但是，启用 KSM 会增加 CPU 使用率，并可能会根据工作负载对整体性能造成负面影响。

根据具体要求，您可以为单个会话启用或禁用 KSM，或者永久禁用 KSM。

注意

在 RHEL 9 及更高版本中，默认禁用 KSM。

先决条件

- 对主机系统的 root 访问权限。

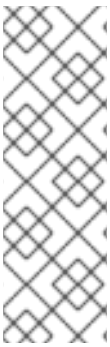
流程

- 禁用 KSM：
  - 要对单个会话停用 KSM，请使用 **systemctl** 工具停止 **ksm** 和 **ksmtuned** 服务。

```
# systemctl stop ksm

# systemctl stop ksmtuned
```

  - 要永久停用 KSM，请使用 **systemctl** 工具来禁用 **ksm** 和 **ksmtuned** 服务。

```
# 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.
```



注意

取消激活 KSM 前在虚拟机间共享的内存页将保持共享。要停止共享，请使用以下命令删除系统中的所有 **PageKSM** 页：

```
# echo 2 > /sys/kernel/mm/ksm/run
```

匿名页面替换了 KSM 页面后，**khugepaged** 内核服务将在虚拟机物理内存中重建透明的大页面。

- 启用 KSM：

警告

启用 KSM 会增加 CPU 使用率并影响整体 CPU 性能。

1. 安装 **ksmtuned** 服务：
   # dnf install ksmtuned

2. 启动服务：

   - 要为单个会话启用 KSM，请使用 **systemctl** 程序启动 **ksm** 和 **ksmtuned** 服务。

     ```
     # systemctl start ksm
     # systemctl start ksmtuned
     ```

   - 要永久启用 KSM，请使用 **systemctl** 程序启用 **ksm** 和 **ksmtuned** 服务。

     ```
     # systemctl enable ksm
     Created symlink /etc/systemd/system/multi-user.target.wants/ksm.service →
     /usr/lib/systemd/system/ksm.service

     # systemctl enable ksmtuned
     Created symlink /etc/systemd/system/multi-user.target.wants/ksmtuned.service →
     /usr/lib/systemd/system/ksmtuned.service
     ```

18.7. 优化虚拟机网络性能

由于虚拟机的网络接口卡(NIC)的虚拟性质，虚拟机会丢失其分配的主机网络带宽的一部分，这可以降低虚拟机的整体工作负载效率。以下提示可最大程度降低虚拟化对虚拟 NIC(vNIC)吞吐量的负面影响。

流程

使用以下任一方法并观察它是否对虚拟机网络性能有帮助：

启用 vhost_net 模块

在主机上，确保启用了 **vhost_net** 内核特性：

```
# lsmod | grep vhost
vhost_net           32768  1
vhost               53248  1 vhost_net
tap                 24576  1 vhost_net
tun                 57344  6 vhost_net
```

如果这个命令的输出为空，请启用 **vhost_net** 内核模块：

```
# modprobe vhost_net
```

设置 multi-queue virtio-net

要为虚拟机设置 *multi-queue virtio-net* 特性，请使用 **virsh edit** 命令来编辑虚拟机的 XML 配置。在 XML 中，将以下内容添加到 **<devices>** 部分，并将 **N** 替换为虚拟机中的 vCPU 个数，最多 16 个:

```
<interface type='network'>
    <source network='default'/>
    <model type='virtio'/>
    <driver name='vhost' queues='N'/>
</interface>
```

如果虚拟机正在运行，重启它以使更改生效。

批量网络数据包

在具有长传输路径的 Linux VM 配置中，在将数据包提交给内核之前对其进行批处理可以提高缓存利用率。要设置数据包批处理，请在主机上使用以下命令，并将 *tap0* 替换为虚拟机使用的网络接口的名称：

```
# ethtool -C tap0 rx-frames 64
```

SR-IOV

如果您的主机 NIC 支持 SR-IOV，请为您的 vNIC 使用 SR-IOV 设备分配。如需更多信息，请参阅管理 SR-IOV 设备。

其它资源

- 了解虚拟网络

18.8. 虚拟机性能监控工具

要确定什么消耗了最多的 VM 资源，以及虚拟机性能的哪方面需要优化，可以使用性能诊断工具，包括通用的和特定于虚拟机的。

默认操作系统性能监控工具

要进行标准性能评估，您可以使用主机和客户机操作系统默认提供的工具：

- 在 RHEL 9 主机上，以 root 用户身份使用 **top** 实用程序或 **系统监控** 应用程序，并在输出中查找 **qemu** 和 **virt**。这显示了您的虚拟机消耗的主机系统资源量。
  - 如果监控工具显示任何 **qemu** 或 **virt** 进程消耗了大量的主机 CPU 或内存量，请使用 **perf** 工具进行调查。详情请查看以下信息。
  - 另外，如果 **vhost_net** 线程进程（如 *vhost_net-1234*）被显示为消耗大量主机 CPU 容量，请考虑使用 虚拟网络优化功能，如 **multi-queue virtio-net**。
- 在客户机操作系统上，使用系统上可用的性能工具和应用程序来评估哪些进程消耗了最多的系统资源。
  - 在 Linux 系统上，您可以使用 **top** 工具。
  - 在 Windows 系统中，您可以使用 **Task Manager** 应用程序。

perf kvm

您可以使用 **perf** 实用程序收集有关 RHEL 9 主机性能的特定虚拟化统计。要做到这一点：

1. 在主机上安装 *perf* 软件包：

   ```
   # dnf install perf
   ```

2. 使用 **perf kvm stat** 命令之一来显示您的虚拟化主机的 perf 统计信息：
   - 若要实时监控 hypervisor，请使用 **perf kvm stat live** 命令。
   - 要记录一段时间内 hypervisor 的 perf 数据，请使用 **perf kvm stat record** 命令激活日志记录。在命令被取消或中断后，数据保存在 **perf.data.guest** 文件中，可以使用 **perf kvm stat report** 命令进行分析。

3. 分析 **VM-EXIT** 事件类型及其分发的 **perf** 输出。例如，**PAUSE_INSTRUCTION** 事件应当不常发生，但在以下输出中，此事件的频繁发生表明主机 CPU 没有很好地处理正在运行的 vCPU。在这种场景下，请考虑关闭某些处于活动状态的虚拟机，从这些虚拟机中删除 vCPU，或者 调优 vCPU 的性能。

```
# perf kvm stat report

Analyze events for all VMs, all VCPUs:


             VM-EXIT    Samples  Samples%     Time%    Min Time    Max Time         Avg time

  EXTERNAL_INTERRUPT     365634    31.59%    18.04%      0.42us  58780.59us    204.08us ( +-   0.99% )
            MSR_WRITE     293428    25.35%     0.13%      0.59us  17873.02us      1.80us ( +-   4.63% )
     PREEMPTION_TIMER     276162    23.86%     0.23%      0.51us  21396.03us      3.38us ( +-   5.19% )
   PAUSE_INSTRUCTION     189375    16.36%    11.75%      0.72us  29655.25us    256.77us ( +-   0.70% )
                 HLT      20440     1.77%    69.83%      0.62us  79319.41us  14134.56us ( +-   0.79% )
              VMCALL      12426     1.07%     0.03%      1.02us   5416.25us      8.77us ( +-   7.36% )
       EXCEPTION_NMI         27     0.00%     0.00%      0.69us      1.34us      0.98us ( +-   3.50% )
       EPT_MISCONFIG          5     0.00%     0.00%      5.15us     10.85us      7.88us ( +-  11.67% )

Total Samples:1157497, Total events handled time:413728274.66us.
```

perf kvm stat 输出中表明有问题的其它事件类型包括：

- **INSN_EMULATION** - 建议次优化的 虚拟机 I/O 配置 。

有关使用 **perf** 监控虚拟化性能的更多信息，请参阅 **perf-kvm** 手册页。

numastat

要查看系统当前的 NUMA 配置，您可以使用 **numastat** 工具，该工具通过安装 **numactl** 软件包来提供。

以下显示了一个有 4 个运行虚拟机的主机，各自从多个 NUMA 节点获取内存。这不是 vCPU 性能的最佳方案，并保证调整：

```
# 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   4851    136     22    445  8116
53773 (qemu-kvm)   1393      3      1      2     12      0      0   6702  8114
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total              1769    463   2024   7462  10037   2672    169   7837 32434
```

相反，以下显示单个节点为每个虚拟机提供内存，这效率显著提高。

```
# 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      0   8113      0  8120
53773 (qemu-kvm)      0      0      7      0      0      0      1   8110  8118
59065 (qemu-kvm)      0      0   8050      0      0      0      0      0  8051
---------------  ------ ------ ------ ------ ------ ------ ------ ------ -----
Total                 0      0   8072      0   8072      0   8114   8110 32368
```

18.9. 其它资源

- 优化 Windows 虚拟机

第 19 章 保护虚拟机

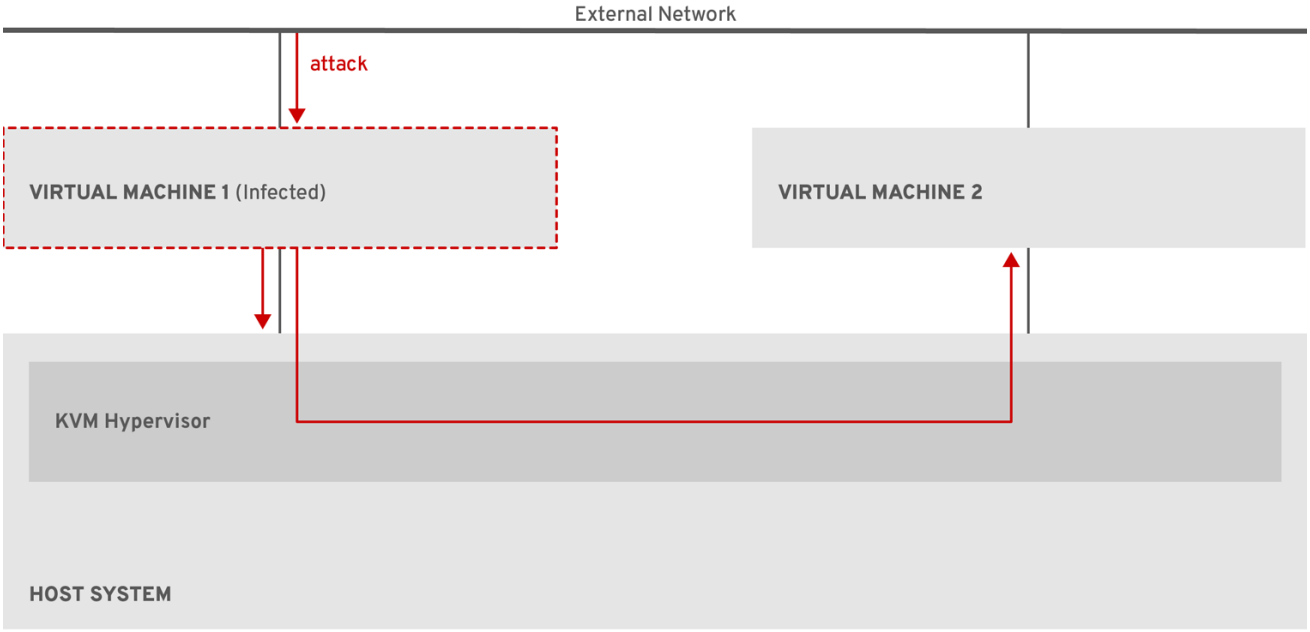
作为使用虚拟机(VM)的 RHEL 9 系统管理员，保护虚拟机的安全可尽可能降低您的客户端和主机操作系统被恶意软件破坏的风险。

本文档概述了在 RHEL 9 主机上保护虚拟机的机制，并提供提高虚拟机安全性的方法列表。

19.1. 虚拟机中的安全性是如何工作的

通过使用虚拟机，可在单一主机机器中托管多个操作系统。这些系统通过 hypervisor 与主机连接，通常也通过虚拟网络连接。因此，每个虚拟机都可以用作使用恶意软件攻击主机的向量，主机可用作攻击任何虚拟机的向量。

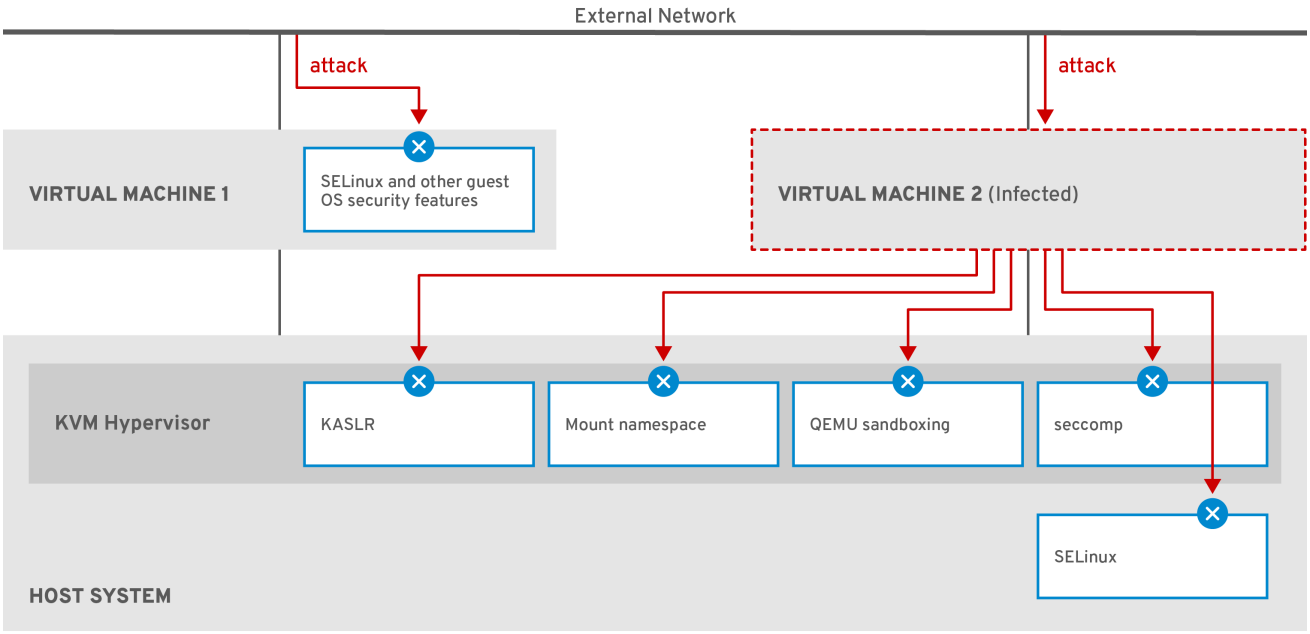
图 19.1. 虚拟化主机上潜在的恶意攻击向量



因为虚拟机监控程序使用主机内核来管理虚拟机，所以在虚拟机操作系统中运行的服务通常会被利用来将恶意代码注入主机系统。但是，您可以通过使用主机和您客户机系统上的 一些安全特性 来保护您的系统免受此类安全威胁。

这些特性（如 SELinux 或 QEMU 沙盒）提供了各种措施，使恶意代码难于攻击 hypervisor，并在您的主机和虚拟机之间进行传输。

图 19.2. 防止对虚拟化主机进行恶意软件攻击



RHEL 9 为虚拟机安全性提供的许多功能始终处于活动状态，且不必启用或配置。详情请查看虚拟机安全自动功能。

此外，您可以遵循各种最佳实践来最大程度降低虚拟机和 hypervisor 的漏洞。如需更多信息，请参阅保护虚拟机的最佳实践。

19.2. 保护虚拟机的最佳实践

根据以下步骤，您的虚拟机被恶意代码利用，并用作攻击向量攻击您的主机系统的风险会大幅降低。

在客户端中：

- 象保护物理机器一样保护虚拟机的安全。增强安全性的具体方法取决于客户端操作系统。如果您的虚拟机正在运行 RHEL 9，请参阅保护 Red Hat Enterprise Linux 9 的安全以改进客户端系统的安全性。

在主机端：

- 当远程管理虚拟机时，请使用加密的工具（如 **SSH**）和网络协议（如 **SSL**）连接到虚拟机。
- 确定 SELinux 处于 Enforcing 模式：

```
# getenforce
Enforcing
```

如果 SELinux 被禁用或者处于 *Permissive* 模式，请参阅使用 SELinux 文档来激活 Enforcing 模式。

注意

SELinux Enforcing 模式还启用 sVirt RHEL 9 功能。这是用于虚拟化的一组特殊的 SELinux 布尔值，可手动调整，以便进行细致的虚拟机安全管理。

- 使用带有 *SecureBoot* 的虚拟机：
  SecureBoot 是一种特性，可确保虚拟机运行加密签名的操作系统。这可防止因为恶意软件攻击而更改了操作系统的虚拟机引导。

  SecureBoot 只能在 AMD64 或 Intel 64 主机上安装使用 OVMF 固件的 Linux 虚拟机时应用。具体步骤请参阅创建 SecureBoot 虚拟机。

- 不要使用 **qemu-*** 命令，如 **qemu-kvm**。
  QEMU 是 RHEL 9 中虚拟化架构的基本组件，但难以手动管理，而且不正确的 QEMU 配置可能会导致安全漏洞。因此，红帽不支持使用大多数 **qemu-*** 命令。反之，使用 *libvirt* 工具，如**virsh**、**virt-install** 和 **virt-xml**，根据最佳实践来编排 QEMU。

  但请注意，对于 虚拟磁盘镜像的管理，支持 **qemu-img** 工具。

其它资源

- RHEL 中虚拟化的 SELinux 布尔值

19.3. 创建 SECUREBOOT 虚拟机

您可以创建一个使用 *SecureBoot* 特性的 Linux 虚拟机(VM)，这将确保您的虚拟机运行加密签名的操作系统。如果虚拟机的客户机操作系统已被恶意软件更改，这将非常有用。在这种场景下，SecureBoot 会阻止虚拟机启动，从而停止可能将恶意软件传播给您的主机。

先决条件

- 虚拟机是 Q35 机器类型。
- 您的主机系统使用 AMD64 或 Intel 64 架构。
- **edk2-OVMF** 软件包已安装：

```
# dnf install edk2-ovmf
```

- 操作系统（OS）安装源可存在于本地或者网络中。可以是以下格式之一：
  - 安装介质的 ISO 镜像
  - 现有虚拟机安装的磁盘镜像

警告

在 RHEL 9 中无法从主机 CD-ROM 或者 DVD-ROM 设备安装。当使用 RHEL 9 中的任何虚拟机安装方法时，如果选择了 CD-ROM 或者 DVD-ROM 作为安装源，则安装将失败。如需更多信息，请参阅红帽知识库。

- 可选：对于快速、简单的配置安装，可以使用 Kickstart 文件。

流程

1. 使用 **virt-install** 命令创建虚拟机，如 使用命令行界面创建虚拟机 中所述。对于 **--boot** 选项，请使用 **uefi,nvram_template=/usr/share/OVMF/OVMF_VARS.secboot.fd** 值。这会使用 **OVMF_VARS.secboot.fd** 和 **OVMF_CODE.secboot.fd** 文件作为虚拟机非易失性 RAM(NVRAM)设置的模板，该设置启用了 SecureBoot 特性。
   例如：

```
# virt-install --name rhel8sb --memory 4096 --vcpus 4 --os-variant rhel9.0 --boot
uefi,nvram_template=/usr/share/OVMF/OVMF_VARS.secboot.fd --disk
boot_order=2,size=10 --disk boot_order=1,device=cdrom,bus=scsi,path=/images/RHEL-9.0-
installation.iso
```

2. 根据屏幕上的说明，按照操作系统安装过程进行操作。

验证

1. 客户机操作系统安装后，通过打开 图形客户机控制台 中的终端或 使用 SSH 连接到客户机 OS，以访问虚拟机的命令行。

2. 要确认是否在虚拟机上启用了 SecureBoot，请使用 **mokutil --sb-state** 命令：

```
# mokutil --sb-state
SecureBoot enabled
```

其他资源

- 在 AMD64、Intel 64 和 64 位 ARM 上安装 RHEL 9

19.4. 限制虚拟机用户可以使用哪些操作

在某些情况下，在 RHEL 9 中托管的虚拟机(VM)用户可以默认执行。如果是这种情况，您可以通过将 **libvirt** 守护进程配置为在主机上使用 **polkit** 策略工具包来限制虚拟机用户可用的操作。

流程

1. **可选：** 确保与 **libvirt** 有关的系统 **polkit** 控制策略根据您的偏好而设置。

   a. 在 **/usr/share/polkit-1/actions/** 和 **/usr/share/polkit-1/rules.d/** 目录中找到所有与 libvirt 相关的文件。

```
# ls /usr/share/polkit-1/actions | grep libvirt
# ls /usr/share/polkit-1/rules.d | grep libvirt
```

   b. 打开文件并检查规则设置。
   有关读取 **polkit** 控制策略语法的详情，请使用 **man polkit**。

   c. 修改 **libvirt** 控制策略。要做到这一点：

      i. 在 **/etc/polkit-1/rules.d/** 目录中创建一个新的 **.rules** 文件。

      ii. 将自定义策略添加到此文件中，并保存。
      有关 **libvirt** 控制策略的更多信息和示例，请参阅 **libvirt** 上游文档。

2. 配置您的虚拟机，以使用由 **polkit** 确定的访问策略。

为此，请在 **/etc/libvirt/** 目录中找到虚拟化驱动程序的所有配置文件，并取消其中注释了 **access_drivers = ["polkit"]**的行。

```
# find /etc/libvirt/ -name virt*d.conf -exec sed -i 's/#access_drivers = \[ "polkit"
\]/access_drivers = \[ "polkit" \]/g' {} +
```

3. 对于您在上一步中修改的每个文件，请重新启动对应的服务。
例如，如果您修改了 **/etc/libvirt/virtqemud.conf**，请重新启动 **virtqemud** 服务。

```
# systemctl try-restart virtqemud
```

验证

- 作为您要限制其虚拟机操作的用户，请执行一个受限操作。
例如，如果非特权用户被限制查看在系统会话中创建的虚拟机：

```
$ virsh -c qemu:///system list --all
 Id   Name           State
--------------------------------
```

如果这个命令没有列出任何虚拟机，即使系统上有一个或多个虚拟机，则 **polkit** 成功地限制了非特权用户的操作。

故障排除

- 目前，将 **libvirt** 配置为使用 **polkit** 便能够使用 RHEL 9 web 控制台连接到虚拟机，因为与 **libvirt-dbus** 服务不兼容。
如果您需要在 web 控制台中对虚拟机进行精细访问控制，请创建一个自定义 D-Bus 策略。具体说明请参阅红帽知识库中的 如何在 Cockpit 中配置虚拟机的精细控制。

其它资源

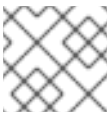
- **man polkit** 命令
- 有关 polkit 访问控制策略 的 **libvirt** 上游信息

19.5. 虚拟机安全性的自动功能

除了在保护虚拟机的最佳实践中列出的手工提高虚拟机的安全性外，**libvirt** 还提供了一组安全功能，它们会在 RHEL 9 的虚拟化环境中自动启用。它们是：

系统和会话连接

要访问 RHEL 9 中虚拟机管理的所有可用实用程序，您需要使用 libvirt 的*系统连接*（**qemu:///system**）。要做到这一点，必须在系统中具有 root 权限，或者作为 *libvirt* 用户组的一部分。不属于 *libvirt* 组中的非 root 用户只能访问 libvirt 的 *会话连接*（**qemu:///session**），后者必须在访问资源时遵守本地用户的访问权限。例如，使用会话连接，您无法检测或访问系统连接中或由其他用户创建的虚拟机。另外，可用的 VM 网络配置选项也有很大限制。



注意

RHEL 9 文档假定您有系统连接权限。

虚拟机分离

单个虚拟机作为隔离进程在主机上运行，并依赖于主机内核强制的安全性。因此，虚拟机无法读取或访问同一主机上其他虚拟机的内存或存储。

QEMU 沙盒

此特性可防止 QEMU 代码执行可能会破坏主机安全性的系统调用。

内核地址空间随机化(KASLR)

启用对内核镜像解压缩的物理和虚拟地址进行随机化。因此，KASLR 会根据内核对象的位置防止客户机安全漏洞。

19.6. 用于虚拟化的 SELINUX 布尔值

RHEL 9 提供了 **sVirt** 功能，它是一组专用的 SELinux 布尔值，它们在 SELinux 处于 Enforcing 模式下的主机上自动启用。

要在 RHEL 9 系统中对虚拟机安全性进行精细配置，您可以在主机上配置 SELinux 布尔值，以确保管理程序以特定方式执行。

要列出所有与虚拟化相关的布尔值及其状态，请使用 **getsebool -a | grep virt** 命令：

```
$ getsebool -a | grep virt
[...]
virt_sandbox_use_netlink --> off
virt_sandbox_use_sys_admin --> off
virt_transition_userdomain --> off
virt_use_comm --> off
virt_use_execmem --> off
virt_use_fusefs --> off
[...]
```

要启用特定的布尔值，请以 root 用户身份使用 **setsebool -P *boolean_name* on** 命令。要禁用布尔值，请使用 **setsebool -P *boolean_name* off**。

下表列出了 RHEL 9 中可用的与虚拟化相关的布尔值以及启用后的功能：

表 19.1. SELinux 虚拟化布尔值

SELinux 布尔值	描述
staff_use_svirt	启用非 root 用户创建并转换虚拟机至 sVirt。
unprivuser_use_svirt	启用非特权用户创建虚拟机并将其转换至 sVirt。
virt_sandbox_use_audit	启用沙盒容器来发送审核信息。
virt_sandbox_use_netlink	启用沙盒容器使用 netlink 系统调用。
virt_sandbox_use_sys_admin	启用沙盒容器使用 sys_admin 系统调用，如 mount。
virt_transition_userdomain	启用虚拟进程作为用户域运行。

SELinux 布尔值	描述
virt_use_comm	启用 virt 使用串行/并行通信端口。
virt_use_execmem	使受限的虚拟客户机可以使用可执行的内存和可执行的堆栈.
virt_use_fusefs	启用 virt 读取 FUSE 挂载的文件。
virt_use_nfs	启用 virt 管理 NFS 挂载的文件。
virt_use_rawip	启用 virt 与 rawip 套接字交互。
virt_use_samba	启用 virt 管理 CIFS 挂载的文件。
virt_use_sanlock	启用受限制的虚拟客户机与 sanlock 交互。
virt_use_usb	启用 virt 使用 USB 设备。
virt_use_xserver	启用虚拟机与 X 窗口系统交互。

19.7. 在 IBM Z 中设置 IBM SECURE EXECUTION

当使用 IBM Z 硬件运行 RHEL 9 主机时，您可以通过为虚拟机配置 IBM Secure Execution 来提高虚拟机(VM)的安全性。

IBM Secure Execution（也称 Protected Virtualization）可防止主机系统访问虚拟机的状态和内存内容。因此，即使主机被攻击，也无法用作攻击客户端操作系统的向量。另外，安全执行也可以用来防止不可信主机从虚拟机获取敏感信息。

以下流程描述了如何将 IBM Z 主机上的现有虚拟机转换为安全虚拟机。

先决条件

- 系统硬件是以下之一：
  - IBM z15 或更高版本
  - IBM LinuxONE III 或更高版本
- 为您的系统启用安全执行功能。要验证，请使用：

```
# grep facilities /proc/cpuinfo | grep 158
```

  如果这个命令显示任何输出，代表您的 CPU 与安全执行兼容。

- 内核包含对安全执行的支持。要确认，请使用：

```
# ls /sys/firmware | grep uv
```

  如果命令生成任何输出，那么内核支持安全执行。

- 主机 CPU 型号包含 **解包** 功能。要确认，请使用：

```
# virsh domcapabilities | grep unpack
<feature policy='require' name='unpack'/>
```

  如果命令生成上述输出，则您的 CPU 主机型号与安全执行兼容。

- 虚拟机的 CPU 模式设置为 **host-model**。若要对此进行确认，请使用以下命令，并将 **vm-name** 替换为您的虚拟机的名称：

```
# virsh dumpxml vm-name | grep "<cpu mode='host-model'/>"
```

  如果命令生成任何输出，则会正确设置虚拟机的 CPU 模式。

- 主机上必须安装 *genprotimg* 软件包。

```
# dnf install genprotimg
```

- 您已获取并验证了 IBM Z 主机密钥文档。有关此操作的说明，请参阅 IBM 文档中的 验证主机密钥文档。

流程

在主机上 执行以下步骤：

1. 将 **prot_virt=1** 内核参数添加到主机的引导配置。

```
# grubby --update-kernel=ALL --args="prot_virt=1"
```

2. 更新引导菜单：
   # zipl

3. 使用 **virsh edit** 修改您要保护的虚拟机的 XML 配置。

4. 将 **<launchSecurity type="s390-pv"/>** 添加到 **</devices>** 行下。例如：

```
[...]
    </memballoon>
  </devices>
  <launchSecurity type="s390-pv"/>
</domain>
```

5. 如果配置的 **<devices>** 部分包含了一个 **virtio-rng** 设备 (**<rng model="virtio">**)，请删除 **<rng>** **</rng>** 块的所有行。

6. **可选：** 如果要保护的虚拟机使用 32 GiB 或更多 RAM，请将 **<async-teardown enabled='yes'/>** 行添加到 XML 配置中的 **<features></features>** 部分。
   这提高了重启或停止此类安全执行客户机的性能。

在您要保护的虚拟机的 **客户机操作系统中** 执行以下步骤。

1. 创建一个参数文件。例如：

```
# touch ~/secure-parameters
```

2. 在 **/boot/loader/entries** 目录中，识别最新版本的引导加载程序条目：

```
# ls /boot/loader/entries -l
[...]
-rw-r--r--. 1 root root  281 Oct  9 15:51 3ab27a195c2849429927b00679db15c1-4.18.0-
240.el8.s390x.conf
```

3. 检索引导装载程序条目的内核选项行：

```
# cat /boot/loader/entries/3ab27a195c2849429927b00679db15c1-4.18.0-240.el8.s390x.conf
| grep options
options root=/dev/mapper/rhel-root
rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap
```

4. 将 options 行的内容和 **swiotlb=262144** 添加到创建的参数文件中。

```
# echo "root=/dev/mapper/rhel-root rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap swiotlb=262144" >
~/secure-parameters
```

5. 生成一个 IBM 安全执行镜像。
例如，以下命令会根据 **/boot/vmlinuz-4.18.0-240.el8.s390x** 映像，使用 **secure-parameters** 文件创建 **/boot/secure-image** 安全镜像、**/boot/initramfs-4.18.0-240.el8.s390x.img** 初始 RAM 磁盘文件和 **HKD-8651-000201C048.crt** 主机密钥文档。

```
# genprotimg -i /boot/vmlinuz-4.18.0-240.el8.s390x -r /boot/initramfs-4.18.0-
240.el8.s390x.img -p ~/secure-parameters -k HKD-8651-00020089A8.crt -o /boot/secure-
image
```

通过使用 **genprotimg** 工具创建安全镜像，其包含内核参数、初始 RAM 磁盘和引导镜像。

6. 更新虚拟机的引导菜单，以从安全镜像引导。此外，删除以 **initrd** 和 **options** 开头的行，因为不需要它们。
例如，在 RHEL 8.3 虚拟机中，可以在 **/boot/loader/entries/** 目录中编辑引导菜单：

```
# cat /boot/loader/entries/3ab27a195c2849429927b00679db15c1-4.18.0-240.el8.s390x.conf
title Red Hat Enterprise Linux 8.3
version 4.18.0-240.el8.s390x
linux /boot/secure-image
[...]
```

7. 创建可引导磁盘镜像：

```
# zipl -V
```

8. 安全地删除原始的未保护的文件。例如：

```
# shred /boot/vmlinuz-4.18.0-240.el8.s390x
# shred /boot/initramfs-4.18.0-240.el8.s390x.img
# shred secure-parameters
```

原始引导镜像、初始 RAM 镜像和内核参数文件未受保护，如果未删除它们，则启用了安全执行的虚拟机仍然容易受到黑客攻击或敏感数据挖掘的攻击。

验证

- 在主机上，使用 **virsh dumpxml** 工具确认受保护虚拟机的 XML 配置。配置必须包含 **<launchSecurity type="s390-pv"/>** 元素，且没有 <rng model="virtio"> 行。

```
# virsh dumpxml vm-name
[...]
  <cpu mode='host-model'/>
  <devices>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2' cache='none' io='native'>
      <source file='/var/lib/libvirt/images/secure-guest.qcow2'/>
      <target dev='vda' bus='virtio'/>
    </disk>
    <interface type='network'>
      <source network='default'/>
      <model type='virtio'/>
    </interface>
    <console type='pty'/>
    <memballoon model='none'/>
  </devices>
  <launchSecurity type="s390-pv"/>
</domain>
```

其它资源

- 有关启动安全虚拟服务器的 IBM 文档
- IBM 关于 **genprotimg** 的文档
- 配置内核命令行参数

19.8. 将加密 COPROCESSORS 附加到 IBM Z 上的虚拟机

要在 IBM Z 主机上的虚拟机中使用硬件加密，请从加密的 coprocessor 设备创建介质设备并将其分配给预期的虚拟机。具体步骤请查看以下说明。

先决条件

- 您的主机运行在 IBM Z 硬件上。
- 加密 coprocessor 与设备分配兼容。要进行确认，请确保协处理器的 **类型** 列为 **CEX4** 或更高版本。

```
# lszcrypt -V

CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS  PENDING HWTYPE QDEPTH
FUNCTIONS  DRIVER
--------------------------------------------------------------------------------
05          CEX5C CCA-Coproc  online       1       0    11     08 S--D--N--  cex4card
05.0004     CEX5C CCA-Coproc  online       1       0    11     08 S--D--N--  cex4queue
05.00ab     CEX5C CCA-Coproc  online       1       0    11     08 S--D--N--  cex4queue
```

- **vfio_ap** 内核模块已加载 。要验证，请使用：

```
# lsmod | grep vfio_ap
vfio_ap        24576  0
[...]
```

要载入模块，请使用：

```
# modprobe vfio_ap
```

- **s390utils** 版本支持 **ap** 处理：

```
# lszdev --list-types
...
ap          Cryptographic Adjunct Processor (AP) device
...
```

流程

1. 获取您要分配给虚拟机的设备的十进制值。例如，对于设备 **05.0004** 和 **05.00ab** ：

```
# echo "obase=10; ibase=16; 04" | bc
4
# echo "obase=10; ibase=16; AB" | bc
171
```

2. 在主机上，将设备重新分配给 **vfio-ap** 驱动程序：

```
# chzdev -t ap apmask=-5 aqmask=-4,-171
```

注意

要永久分配设备，请使用 **-p** 标志。

3. 验证是否正确重新分配了加密设备。

```
# lszcrypt -V

CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS  PENDING HWTYPE QDEPTH
FUNCTIONS  DRIVER
--------------------------------------------------------------------------------
05          CEX5C CCA-Coproc  -            1       0     11     08 S--D--N--  cex4card
05.0004     CEX5C CCA-Coproc  -            1       0     11     08 S--D--N--  vfio_ap
05.00ab     CEX5C CCA-Coproc  -            1       0     11     08 S--D--N--  vfio_ap
```

如果域队列的 DRIVER 值变为 **vfio_ap**，则重新分配成功。

4. 创建一个定义新介质设备的 XML 片段。
以下示例演示了定义一个永久的介质设备，并为它分配队列。具体来说，本例中的 **vfio_ap.xml** XML 片断向到介质设备分配一个域适配器 **0x05**、域队列 **0x0004** 和 **0x00ab**，以及一个控制域 **0x00ab** 。

```
# vim vfio_ap.xml
```

```
<device>
  <parent>ap_matrix</parent>
  <capability type="mdev">
    <type id="vfio_ap-passthrough"/>
    <attr name='assign_adapter' value='0x05'/>
    <attr name='assign_domain' value='0x0004'/>
    <attr name='assign_domain' value='0x00ab'/>
    <attr name='assign_control_domain' value='0x00ab'/>
  </capability>
</device>
```

5. 从 **vfio_ap.xml** XML 片断创建一个新的介质设备。

```
# virsh nodedev-define vfio_ap.xml
Node device 'mdev_8f9c4a73_1411_48d2_895d_34db9ac18f85_matrix' defined from
'vfio_ap.xml'
```

6. 启动您在上一步中创建的介质设备，在本例中为 **mdev_8f9c4a73_1411_48d2_895d_34db9ac18f85_matrix**。

```
# virsh nodedev-start mdev_8f9c4a73_1411_48d2_895d_34db9ac18f85_matrix
Device mdev_8f9c4a73_1411_48d2_895d_34db9ac18f85_matrix started
```

7. 检查配置是否已正确应用

```
# cat /sys/devices/vfio_ap/matrix/mdev_supported_types/vfio_ap-
passthrough/devices/669d9b23-fe1b-4ecb-be08-a2fabca99b71/matrix
05.0004
05.00ab
```

如果输出中包含您之前分配给 **vfio-ap** 的队列的数字值，则该过程成功。

8. 将介质设备附加到虚拟机。

   a. 显示您创建的介质设备的 UUID，并保存它，以供下一步使用。

```
# virsh nodedev-dumpxml mdev_8f9c4a73_1411_48d2_895d_34db9ac18f85_matrix

<device>
  <name>mdev_8f9c4a73_1411_48d2_895d_34db9ac18f85_matrix</name>
  <parent>ap_matrix</parent>
  <capability type='mdev'>
    <type id='vfio_ap-passthrough'/>
    <uuid>8f9c4a73-1411-48d2-895d-34db9ac18f85</uuid>
    <iommuGroup number='0'/>
    <attr name='assign_adapter' value='0x05'/>
    <attr name='assign_domain' value='0x0004'/>
    <attr name='assign_domain' value='0x00ab'/>
    <attr name='assign_control_domain' value='0x00ab'/>
  </capability>
</device>
```

   b. 为加密策略中介设备创建并打开一个 XML 文件。例如：

```
# vim crypto-dev.xml
```

c. 将以下行添加到文件中并保存。将 **uuid** 值替换为您在步骤 *a* 中获取的 UUID。

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-ap'>
  <source>
    <address uuid='8f9c4a73-1411-48d2-895d-34db9ac18f85'/>
  </source>
</hostdev>
```

d. 使用 XML 文件将介质设备附加到虚拟机。例如，要将 **crypto-dev.xml** 文件中定义的设备永久附加到正在运行的 **testguest1** 虚拟机：

```
# virsh attach-device testguest1 crypto-dev.xml --live --config
```

--live 选项仅将设备附加到正在运行的虚拟机，不会在引导之间保持持久性。**--config** 选项使配置更改持久化。您可以只使用 **--config** 选项将设备附加到关闭的虚拟机。

请注意，每个 UUID 每次只能分配给一个虚拟机。

验证

1. 确保客户端操作系统检测到了分配的加密设备。

```
# lszcrypt -V

CARD.DOMAIN TYPE  MODE        STATUS  REQUESTS  PENDING HWTYPE QDEPTH
FUNCTIONS  DRIVER
-----------------------------------------------------------------------------
05          CEX5C CCA-Coproc  online        1        0    11     08 S--D--N--  cex4card
05.0004     CEX5C CCA-Coproc  online        1        0    11     08 S--D--N--  cex4queue
05.00ab     CEX5C CCA-Coproc  online        1        0    11     08 S--D--N--  cex4queue
```

客户端操作系统中这个命令的输出将与具有同样加密协处理器设备的主机逻辑分区上的输出相同。

2. 在客户端操作系统中，确认控制域已成功分配给加密设备。

```
# lszcrypt -d C

DOMAIN 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
------------------------------------------------------
  00 . . . . U . . . . . . . . . .
  10 . . . . . . . . . . . . . . .
  20 . . . . . . . . . . . . . . .
  30 . . . . . . . . . . . . . . .
  40 . . . . . . . . . . . . . . .
  50 . . . . . . . . . . . . . . .
  60 . . . . . . . . . . . . . . .
  70 . . . . . . . . . . . . . . .
  80 . . . . . . . . . . . . . . .
  90 . . . . . . . . . . . . . . .
  a0 . . . . . . . . . . . B . . . .
  b0 . . . . . . . . . . . . . . .
```

```
   c0 . . . . . . . . . . . . . . . .
   d0 . . . . . . . . . . . . . . . .
   e0 . . . . . . . . . . . . . . . .
   f0 . . . . . . . . . . . . . . . .
--------------------------------------------------
C: Control domain
U: Usage domain
B: Both (Control + Usage domain)
```

如果 **lszcrypt -d C** 在加密设备矩阵中显示 **U** 和 **B** 相交，则控制域分配是成功的。

19.9. 在 WINDOWS 虚拟机中启用标准硬件安全性

要保护 Windows 虚拟机，您可以使用 Windows 设备的标准硬件功能启用基本级别安全性。

先决条件

- 请确定您安装了最新的 WHQL 认证的 VirtIO 驱动程序。
- 确保虚拟机固件支持 UEFI 引导。
- 在您的主机上安装 **edk2-OVMF** 软件包。

```
# {PackageManagerCommand} install edk2-ovmf
```

- 在您的主机上安装 **vTPM** 软件包。

```
# {PackageManagerCommand} install swtpm libtpms
```

- 确保虚拟机使用 Q35 机器架构。
- 请确定您有 Windows 安装介质。

流程

1. 通过在虚拟机 XML 配置中的 **<devices>** 部分中添加以下参数来启用 TPM 2.0。

```
<devices>
[...]
  <tpm model='tpm-crb'>
    <backend type='emulator' version='2.0'/>
  </tpm>
[...]
</devices>
```

2. 在 UEFI 模式中安装 Windows。有关如何操作的更多信息，请参阅 创建 SecureBoot 虚拟机。
3. 在 Windows 虚拟机上安装 VirtIO 驱动程序。有关如何操作的更多信息，请参阅 在 Windows 客户端上安装 virtio 驱动程序。
4. 在 UEFI 中，启用安全引导。有关如何操作的更多信息，请参阅 安全引导。

验证

- 确定 Windows 机器中的设备**安全性**页面显示以下信息：
  settings > Update & Security > Windows Security > device Security

> **Your device meets the requirements for standard hardware security.**

19.10. 在 WINDOWS 虚拟机上启用增强的硬件安全

为进一步保护 Windows 虚拟机(VM)，您可以启用基于虚拟化的代码完整性保护，也称为 hypervisor 保护的代码完整性(HVCI)。

先决条件

- 确保启用了标准硬件安全。如需更多信息，请参阅在 Windows 虚拟机上启用标准硬件安全性。
- 确保启用了 Hyper-V enlightenments。如需更多信息，请参阅启用 Hyper-Vlightenments。

流程

1. 打开 Windows 虚拟机的 XML 配置。以下示例打开 *Example-L1* 虚拟机的配置：

> **# virsh edit *Example-L1***

2. 在 **<cpu>** 部分下，指定 CPU 模式并添加策略标记。

重要

- 对于 Intel CPU，启用 **vmx** 策略标记。
- 对于 AMD CPU，启用 **svm** 策略标记。
- 如果您不想指定一个自定义 CPU，您可以将 **<cpu mode>** 设置为 **host-passthrough**。

```
<cpu mode='custom' match='exact' check='partial'>
  <model fallback='allow'>Skylake-Client-IBRS</model>
  <topology sockets='1' dies='1' cores='4' threads='1'/>
  <feature policy='require' name='vmx'/>
</cpu>
```

3. 保存 XML 配置并重启虚拟机。
4. 在虚拟机操作系统中，进入到 **Core 隔离详情**页 ：
   settings > Update & Security > Windows Security > Device Security > Core isolated details
5. 切换开关以启用 **内存完整性**。
6. 重启虚拟机。

注意

有关启用 HVCI 的其他方法，请查看相关的 Microsoft 文档。

验证

- 确保 Windows 虚拟机上的 **Device Security** 页面显示以下信息：
  settings > Update & Security > Windows Security > device Security

  > **Your device meets the requirements for enhanced hardware security.**

- 或者，检查有关 Windows 虚拟机的系统信息：

  a. 在命令提示符下运行 **msinfo32.exe**。

  b. 检查 **Credential Guard, Hypervisor enforced Code Integrity** 是否在 **Virtualization-based security Services Running** 下列出。

第 20 章 在主机及其虚拟机间共享文件

您可能需要在主机系统和其运行的虚拟机(VM)之间共享数据。要快速高效地完成此操作，您可以在系统上建立 NFS 文件共享。或者，您也可以使用 **virtiofs** 与 Linux 和 Windows 虚拟机共享数据。

20.1. 使用 NFS 在主机和其虚拟机之间共享文件

为了 RHEL 9 主机系统和虚拟机(VM)之间高效的文件共享，您可以导出虚拟机可以挂载和访问的 NFS 共享。

但是，对于 Linux 虚拟机，使用 **virtiofs** 特性通常更为方便。

先决条件

- **nfs-utils** 软件包已安装在主机上。

```
# dnf install nfs-utils -y
```

- **NAT** 的虚拟网络或 **网桥** 类型被配置为将主机连接到虚拟机。
- **可选：** 为了提高安全性，请确保您的虚拟机与 NFS 版本 4 或更高版本兼容。

流程

1. 在主机上，将一个带有文件的目录导出为网络文件系统(NFS)：
   a. 与虚拟机共享一个现有目录。如果您不想共享任何现有目录，请创建一个新目录：

```
# mkdir shared-files
```

   b. 获取每个虚拟机的 IP 地址，以便从主机（如 *testguest1* 和 *testguest2*）共享文件：

```
# virsh domifaddr testguest1
Name       MAC address          Protocol     Address
-------------------------------------------------------------------------------
vnet0      52:53:00:84:57:90    ipv4         192.0.2.2/24

# virsh domifaddr testguest2
Name       MAC address          Protocol     Address
-------------------------------------------------------------------------------
vnet1      52:53:00:65:29:21    ipv4         192.0.2.3/24
```

   c. 编辑主机上的 **/etc/exports** 文件，并添加一行，其中包含您要共享的目录、要共享的虚拟机的 IP 以及其它选项：

```
/home/<username>/Downloads/<shared_directory>/ <VM1-IP(options)> <VM2-IP(options)>
...
```

   以下示例将主机上的 **/usr/local/shared-files** 目录与 *testguest1* 和 *testguest2* 共享，并允许虚拟机编辑目录的内容：

```
/usr/local/shared-files/ 192.0.2.2(rw,sync) 192.0.2.3(rw,sync)
```

注意

要与 Windows 虚拟机共享目录，您需要确保 Windows NFS 客户端在共享目录中有写权限。您可以在 **/etc/exports** 文件中使用 **all_squash**、**anonuid** 和 **anongid** 选项。

/usr/local/shared-files/ 192.0.2.2(rw,sync,all_squash,anonuid=*<directory-owner-UID>*,anongid=*<directory-owner-GID>*)

<directory-owner-UID> 和 *<directory-owner-GID>* 是拥有主机上共享目录的本地用户的 UID 和 GID。

对于管理 NFS 客户端权限的其他选项，请按照 确保 NFS 服务安全 指南进行操作。

d. 导出更新的文件系统：

```
# exportfs -a
```

e. 启动 **nfs-server** 服务：

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

f. 获取主机系统的 IP 地址，来在虚拟机上挂载共享目录：

```
# 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 192.0.2.1/24 brd 192.0.2.255 scope global virbr0
valid_lft forever preferred_lft forever
...
```

请注意，相关网络将主机与虚拟机连接，以共享文件。通常，这是 **virbr0**。

2. 在 Linux 虚拟机上挂载共享目录在 **/etc/exports** 文件中指定：

```
# mount 192.0.2.1:/usr/local/shared-files /mnt/host-share
```

- **192.0.2.1** ：主机的 IP 地址。
- **/usr/local/shared-files** ：主机上导出的目录的文件系统路径。
- **/mnt/host-share** ：虚拟机上的挂载点

注意

挂载点必须是一个空目录。

3. 在 Windows 虚拟机上挂载共享目录，如 **/etc/exports** 文件中所述：

a. 以管理员身份打开 PowerShell shell 提示符。

b. 在 Windows 上安装 **NFS-Client** 软件包。

i. 要在服务器版本上安装，请输入：

```
# Install-WindowsFeature NFS-Client
```

ii. 要在桌面版本上安装，请输入：

```
# Enable-WindowsOptionalFeature -FeatureName ServicesForNFS-ClientOnly,
ClientForNFS-Infrastructure -Online -NoRestart
```

c. 在 Windows 虚拟机上挂载主机导出的目录：

```
# C:\Windows\system32\mount.exe -o anon \\192.0.2.1\usr\local\shared-files Z:
```

在本例中：

- **192.0.2.1** ：主机的 IP 地址。
- **/usr/local/shared-files** ：主机上导出的目录的文件系统路径。
- **Z:**: 挂载点的盘符。

注意

您必须选择一个系统上不使用的驱动器符。

验证

- 列出虚拟机上共享目录的内容，以便您可以在主机和虚拟机之间共享文件：

```
$ ls <mount_point>
shared-file1  shared-file2  shared-file3
```

在这个示例中，将 *<mount_point>* 替换为挂载的共享目录的文件系统路径。

其它资源

- 部署 NFS 服务器

20.2. 使用 VIRTIOFS 在主机及其虚拟机间共享文件

使用 virtiofs，您可以在主机和虚拟机(VM)之间将文件共享为与本地文件系统结构相同的目录树。您可以使用 virtiofs 执行以下任务：

- 在主机和虚拟机间共享文件
- 在主机和 Windows 虚拟机间共享文件
- 使用 Web 控制台在主机和虚拟机间共享文件
- 使用 Web 控制台删除主机和虚拟机间的共享文件

20.2.1. 使用 virtiofs 在主机及其虚拟机间共享文件

当使用 RHEL 9 作为管理程序时，您可以使用 **virtiofs** 功能在主机系统及其虚拟机(VM)之间高效地共享文件。

先决条件

- 虚拟化已在 RHEL 9 主机 上安装并启用。
- 要与虚拟机共享的目录。如果您不想共享任何现有目录，请创建一个新目录，例如： *shared-files*。

```
# mkdir /root/shared-files
```

- 您要与之共享数据的虚拟机使用 Linux 发行版作为其客户机操作系统。

流程

1. 对于您要与虚拟机共享的主机的每个目录，请在虚拟机 XML 配置中将其设置为 virtiofs 文件系统。

   a. 打开预期虚拟机的 XML 配置。

   ```
   # virsh edit vm-name
   ```

   b. 在虚拟机 XML 配置的 **<devices>** 部分添加类似于以下内容的条目。

   ```
   <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>
   ```

   本例设置主机上的 **/root/shared-files** 目录，使其作为 **host-file-share** 呈现给虚拟机。

2. 为虚拟机设置共享内存。要做到这一点，将共享内存支持添加到 XML 配置的 & **lt;domain** > 部分：

   ```
   <domain>
    [...]
    <memoryBacking>
      <access mode='shared'/>
    </memoryBacking>
    [...]
   </domain>
   ```

3. 引导虚拟机。

   ```
   # virsh start vm-name
   ```

4. 在客户端操作系统中挂载文件系统。以下示例使用 Linux 客户机操作系统挂载之前配置的 **host-file-share** 目录。

```
# mount -t virtiofs host-file-share /mnt
```

验证

- 确保共享目录可在虚拟机上访问，且您现在可以打开文件存储在 目录中。

限制和已知问题

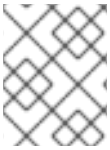
- 与访问时间相关的文件系统挂载选项（如 **noatime** 和 **strictatime** ）可能不适用于 virtiofs，红帽不建议使用它。

故障排除

- 如果 **virtiofs** 不适用于您的用例或系统不支持，您可以使用 NFS。

20.2.2. 使用 virtiofs 在主机和 Windows 虚拟机间共享文件

当使用 RHEL 9 作为 hypervisor 时，您可以使用 **virtiofs** 功能以及 **virtio-win** 软件包在主机系统和 Windows 虚拟机(VM)之间高效地共享文件。



注意

您可以使用 **virtiofs.exe** 命令和 **-i** 参数在 Windows 虚拟机上运行 **virtiofs** 服务，不区分大小写。

先决条件

- 您已将虚拟机的 XML 配置文件配置为使用 virtiofs。详情请查看 第 20.2.1 节 “使用 virtiofs 在主机及其虚拟机间共享文件”。
- 您已将 **virtio** 驱动程序安装介质附加到虚拟机。
- 您已在 Windows 虚拟机上安装了 **virtio-win** 软件包。如需更多信息，请参阅在 Windows 客户端中安装 virtio 驱动程序。

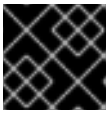
流程

1. 在 Windows 虚拟机上，安装 WinFsp。为此，请挂载 **virtio-win** ISO 镜像，启动 **winfsp** MSI 安装程序，并按照提示进行操作。
   在安装向导的 **Custom Setup** 窗口中，选择您要在虚拟机上安装的功能。
2. 启动 virtiofs 服务：

   ```
   # sc start VirtioFsSvc
   ```

3. 进入 **This PC**:
   File Explorer → This PC

   virtiofs 应作为从 **z:** 开始的第一个可用驱动器符在 Windows 虚拟机上提供，并向后移动。例如，**my_viofs (Z:)**。



重要

您必须在每次虚拟机重启后重启 virtiofs 服务来访问共享目录。

4. **可选：** 要设置额外的 virtiofs 实例：

   a. 停止 virtiofs 服务：

   ```
   # sc stop VirtioFsSvc
   # sc config VirtioFsSvc start=demand
   ```

   b. 配置 WinFSP.Launcher 服务以设置多个 virtiofs 实例：

   ```
   # "C:\Program Files (x86)\WinFsp\bin\fsreg.bat" virtiofs "<path to the binary>\virtiofs.exe"
   "-t %1 -m %2"
   ```

   c. 将 virtiofs 实例挂载到驱动器。
   例如，将带有标签 **mount_tag0** 的 virtiofs 挂载到 **Y:** 驱动器：

   ```
   "C:\Program Files (x86)\WinFsp\bin\launchctl-x64.exe" start virtiofs viofsY mount_tag0 Y:
   ```

   d. 重复上述步骤以挂载所有 virtiofs 实例。

   e. 要卸载 virtiofs 实例：

   ```
   "C:\Program Files (x86)\WinFsp\bin\launchctl-x64.exe" stop virtiofs viofsY
   ```

验证

1. 在 Windows 虚拟机上，导航到 **This PC**:
   File Explorer → This PC

   - 如果您在设置 virtiofs 服务时没有指定挂载点，它将使用以 **z:** 开始的第一个可用驱动器符，并往后移动。
   - 如果您设置了多个 virtiofs 实例，它们将显示为具有您分配给实例的符号的驱动器。

20.2.3. 在 web 控制台中使用 virtiofs 在主机及其虚拟机间共享文件

您可以通过 RHEL web 控制台使用 **virtiofs** 功能在主机系统及其虚拟机(VM)之间高效地共享文件。

先决条件

- Web 控制台 VM 插件 已安装在您的系统上。
- 要与虚拟机共享的目录。如果您不想共享任何现有的目录，请创建一个新目录，例如，名为 *centurion*。

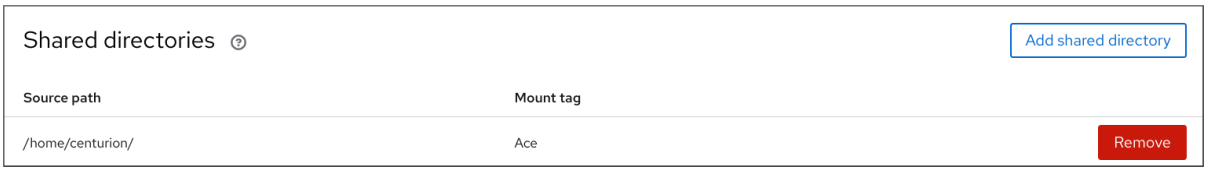
  ```
  # mkdir /home/centurion
  ```

- 您要与之共享数据的虚拟机使用 Linux 发行版作为其客户机操作系统。

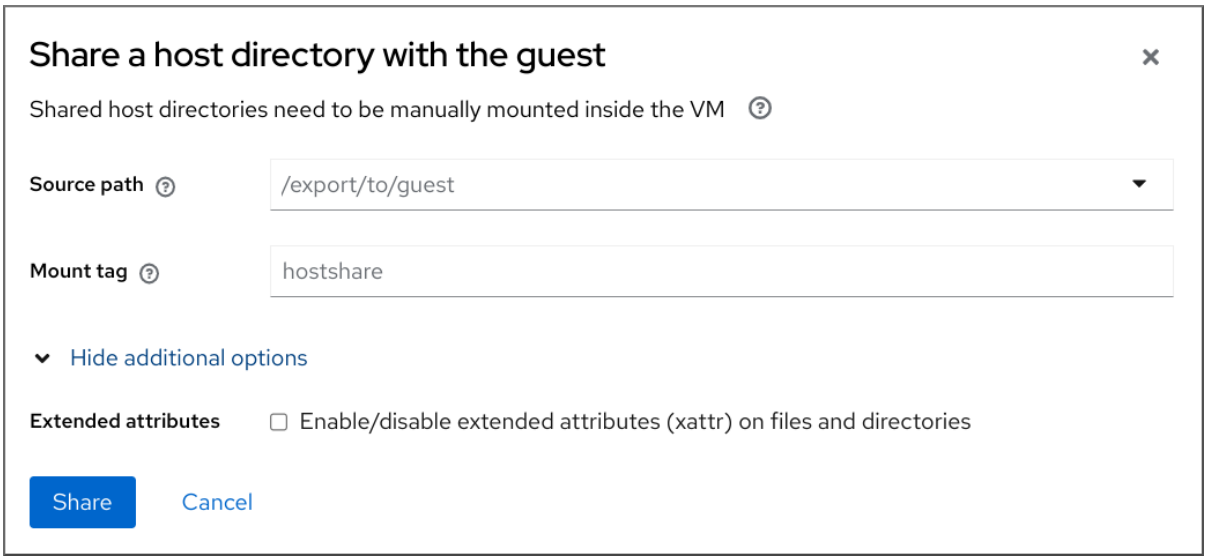
流程

1. 在 **Virtual Machines** 接口中，点击您要共享文件的虚拟机。
这时将打开一个新页面，其中有一个 **Overview** 部分，其中包含有关所选虚拟机和 **Console** 部分的基本信息。

2. 滚动到 **Shared directories**。
Shared directory 部分显示由该虚拟机共享的主机文件和目录的信息，以及用来 **添加** 或 **删除** 共享目录的选项。



3. 单击 **Add shared directory**。
此时会出现 **与客户机共享主机目录** 对话框。



4. 输入以下信息：
   - **Source path** - 您要共享的主机目录的路径。
   - **Mount tag** - 虚拟机用来挂载目录的标签。
5. 设置其它选项：
   - **Extended attributes** - 设置是否在共享文件和目录上启用扩展属性 **xattr**。
6. 单击 **Share**。
所选目录与虚拟机共享。

验证

- 确保共享目录可在虚拟机上访问，并且您现在可以打开存储在该目录中的文件。

20.2.4. 通过 web 控制台使用 virtiofs 删除主机及其虚拟机之间的共享文件

您可以通过 RHEL web 控制台使用 **virtiofs** 功能删除主机系统及其虚拟机(VM)之间共享的文件。

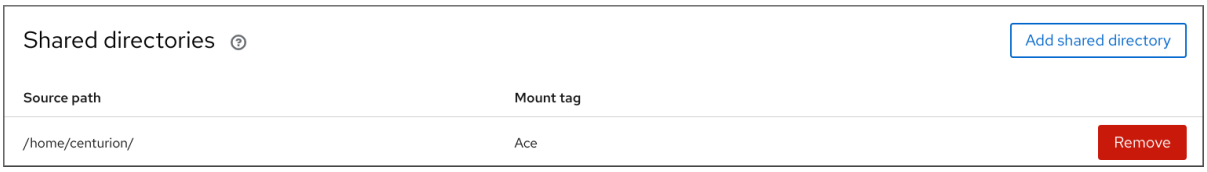
先决条件

- Web 控制台 VM 插件 已安装在您的系统上。
- 虚拟机不再使用该目录。

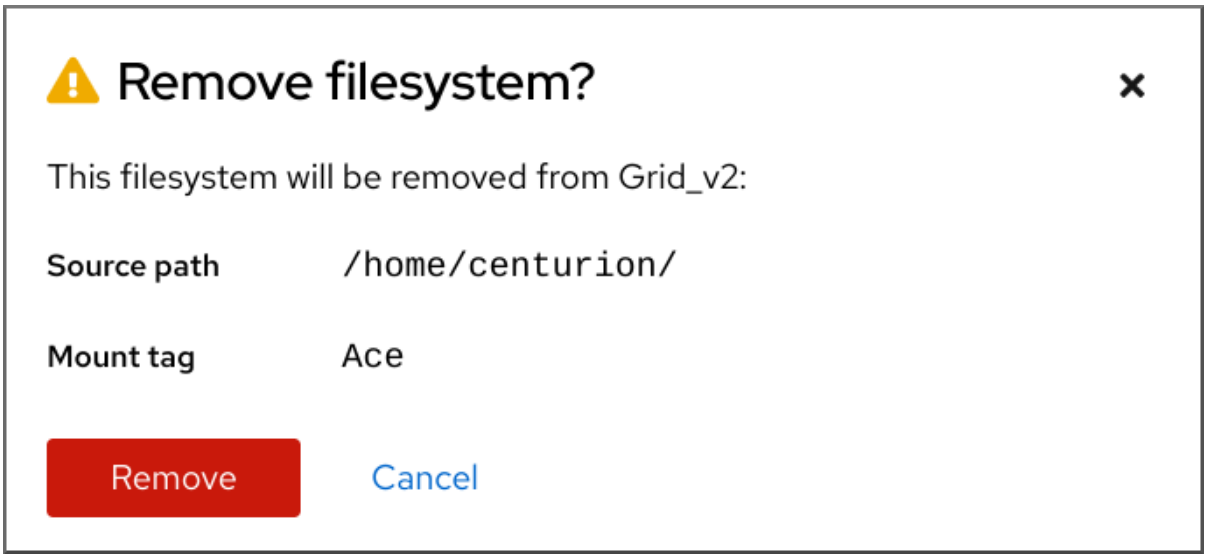
流程

1. 在 **Virtual Machines** 接口中，点击您要从中删除共享文件的虚拟机。
这时将打开一个新页面，其中有一个 **Overview** 部分，其中包含有关所选虚拟机和 **Console** 部分的基本信息。

2. 滚动到 **Shared directories**。
Shared directory 部分显示由该虚拟机共享的主机文件和目录的信息，以及用来 **添加** 或 **删除** 共享目录的选项。



3. 点击您要与虚拟机解除共享的目录旁边的 **Remove**。
此时会出现 **Remove filesystem** 对话框。



4. 点击 **Remove**。
所选目录与虚拟机解除共享。

验证

- 共享目录对虚拟机不再可用和可访问。

第 21 章 安装和管理 WINDOWS 虚拟机

要在 RHEL 9 主机上使用 Microsoft Windows 作为虚拟机(VM)中的客户机操作系统，红帽建议执行额外的步骤以确保这些虚拟机正确运行。

为此，以下章节提供了有关在主机上安装并优化 Windows 虚拟机，以及在这些虚拟机中安装和配置驱动程序的信息。

21.1. 安装 WINDOWS 虚拟机

您可以在 RHEL 9 主机上创建完全虚拟化的 Windows 机器，在虚拟机(VM)中启动图形 Windows 安装程序，并优化已安装的 Windows 虚拟机操作系统(OS)。

要创建虚拟机并安装 Windows 客户机操作系统，请使用 **virt-install** 命令或 RHEL 9 web 控制台。

先决条件

- Windows OS 安装源可以是以下之一，并在本地或网络上可用：
  - 安装介质的 ISO 镜像
  - 现有虚拟机安装的磁盘镜像
- 带有 KVM **virtio** 驱动程序的存储介质.
  要创建此介质，请参阅在主机中准备 virtio 驱动程序安装介质。
- 如果要安装 Windows 11，必须在主机上安装 **edk2-ovmf**、**swtpm** 和 **libtpms** 软件包。

流程

1. 创建虚拟机。具体步骤请参阅创建虚拟机，但请注意以下具体信息。
   - 如果使用 **virt-install** 工具来创建虚拟机，请在命令中添加以下选项：
     - 带有 KVM **virtio** 驱动程序的存储介质.例如：

       ```
       --disk path=/usr/share/virtio-win/virtio-win.iso,device=cdrom
       ```

     - 要安装的 Windows 版本。例如，对于 Windows 10 和 11：

       ```
       --os-variant win10
       ```

       要获得可用 Windows 版本列表以及相应的选项，请使用以下命令：

       ```
       # osinfo-query os
       ```

     - 如果您要安装 Windows 11，启用 *统一可扩展固件接口* (UEFI) 和 *虚拟信任平台模块* (vTPM)：

       ```
       --boot uefi
       ```

   - 如果使用 Web 控制台创建虚拟机，请在 **Create New Virtual Machine**窗口中的 **Operating System** 字段中指定您的 Windows 版本。

- 如果要在 Windows 11 和 Windows Server 2022 之前安装 Windows 版本，点**创建并运行**开始安装并运行。
- 如果您要安装 Windows 11，或者您想要使用额外的 Windows Server 2022 功能，点**创建并编辑**并使用 CLI 启用 UEFI 和 vTPM：

A. 打开虚拟机的 XML 配置：

```
# virsh edit windows-vm
```

B. 在 **os** 元素中添加 **firmware='efi'** 选项：

```
<os firmware='efi'>
  <type arch='x86_64' machine='pc-q35-6.2'>hvm</type>
  <boot dev='hd'/>
</os>
```

C. 在 **devices** 元素中添加 **tpm** 设备：

```
<devices>
  <tpm model='tpm-crb'>
    <backend type='emulator' version='2.0'/>
  </tpm>
</devices>
```

D. 点**虚拟机**表中的 **Install** 启动 Windows 安装。

2. 在虚拟机中安装 Windows OS。
有关如何安装 Windows 操作系统的详情，请参考相关的 Microsoft 安装文档。

3. 如果使用 Web 控制台创建虚拟机，请使用 **Disks** 接口将带有 virtio 驱动程序的存储介质附加到虚拟机。具体说明请参阅 使用 web 控制台将现有磁盘附加到虚拟机。

4. 在 Windows 客户机操作系统中配置 KVM **virtio** 驱动程序。详情请查看为 Windows 虚拟机安装 KVM 半虚拟驱动程序。

其它资源

- 优化 Windows 虚拟机
- 在 Windows 虚拟机中启用标准硬件安全性
- 在 Windows 虚拟机上启用增强的硬件安全
- 虚拟机 XML 配置示例

21.2. 优化 WINDOWS 虚拟机

当在 RHEL 9 中托管的虚拟机(VM)中使用 Microsoft Windows 作为客户机操作系统时，客户机的性能可能会受到负面影响。

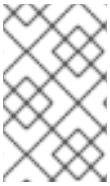
因此,红帽建议您使用以下组合来优化 Windows 虚拟机：

- 使用半虚拟驱动程序。如需更多信息，请参阅为 Windows 虚拟机安装 KVM 半虚拟驱动程序。

- 启用 Hyper-V enlightenments。如需更多信息，请参阅启用 Hyper-Vlightenments。
- 配置 NetKVM 驱动程序参数。如需更多信息，请参阅配置 NetKVM 驱动程序参数。
- 优化或禁用 Windows 后台进程。如需更多信息，请参阅优化 Windows 虚拟机上的后台进程。

21.2.1. 为 Windows 虚拟机安装 KVM 半虚拟驱动程序

提高 Windows 虚拟机性能的主要方法是在客户机操作系统上为 Windows 安装 KVM 半虚拟化(**virtio**)驱动程序。



注意

virtio-win 驱动程序是针对 Windows 10 和 11 的最新版本认证(WHQL)的，可在各自的 **virtio-win** 发布时提供。但是，**virtio-win** 驱动程序通常是经过测试的，并预期在之前 Windows 10 和 11 的构建上可以正常工作。

要在 Windows 虚拟机上安装驱动程序，请执行以下操作：

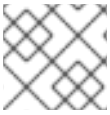
1. 准备主机机器上的安装介质。如需更多信息，请参阅在主机中准备 virtio 驱动程序安装介质。
2. 将安装介质附加到现有 Windows 虚拟机上，或者在创建新 Windows 虚拟机时附加该介质。如需更多信息，请参阅 在 RHEL 上安装 Windows 虚拟机 。
3. 在 Windows 客户机操作系统中安装 **virtio** 驱动程序。如需更多信息，请参阅在 Windows 客户端中安装 virtio 驱动程序。
4. 在 Windows 客户机操作系统上安装 **QEMU 客户机代理**。如需更多信息，请参阅 在 Windows 客户机上安装 QEMU 客户机代理。

21.2.1.1. Windows virtio 驱动程序如何工作

半虚拟化驱动程序通过降低 I/O 延迟并将吞吐量提高到几乎裸机水平来提高虚拟机(VM)的性能。红帽建议您对运行 I/O 密集型任务和应用程序的虚拟机使用半虚拟化驱动程序。

virtio 驱动程序是 KVM 的半虚拟化设备驱动程序，可用于在 KVM 主机上运行的 Windows 虚拟机。这些驱动程序由 **virtio-win** 软件包提供，其中包括以下设备的驱动程序：

- 块（存储）设备
- 网络接口控制器
- 视频控制器
- 内存 ballooning 设备
- 半虚拟串口设备
- 熵源设备
- 半虚拟 panic 设备
- 输入设备，如鼠标、键盘或平板电脑
- 一组小型模拟设备



注意

如需有关模拟、**virtio** 和分配设备的更多信息，请参阅管理虚拟设备。

通过使用 KVM virtio 驱动程序，预计以下 Microsoft Windows 版本的运行方式与物理系统类似：

- Windows Server 版本：请参阅红帽知识库中的 具有 KVM 的 Red Hat Enterprise Linux 的认证的客户机操作系统 。
- Windows Desktop （非服务器）版本：
  - Windows 10 (32 位和 64 位版本)
  - Windows 11 (64 位)

21.2.1.2. 在主机中准备 virtio 驱动程序安装介质

要在 Windows 虚拟机(VM)中安装或更新 KVM **virtio** 驱动程序，您必须首先在主机上准备 **virtio** 驱动程序安装介质。为此，请将 **virtio-win** 软件包提供的 **.iso** 文件作为存储设备附加到 Windows 虚拟机上。

先决条件

- 确定在 RHEL 9 主机系统中启用了虚拟化。如需更多信息，请参阅启用虚拟化。
- 确保具有虚拟机的 root 访问权限。

流程

1. 刷新您的订阅数据：

```
# subscription-manager refresh
All local data refreshed
```

2. 获取 **virtio-win** 软件包的最新版本。
   - 如果没有安装 **virtio-win** ：

```
# dnf install -y virtio-win
```

   - 如果安装了 **virtio-win** ：

```
# dnf upgrade -y virtio-win
```

   如果安装成功，则 **virtio-win** 驱动程序文件位于 **/usr/share/virtio-win/** 目录中。其中包括 **ISO** 文件和一个 **drivers** 目录，目录中包含驱动程序文件，每个架构及受支持的 Windows 版本各对应一个文件。

```
# ls /usr/share/virtio-win/
drivers/  guest-agent/  virtio-win-1.9.9.iso  virtio-win.iso
```

3. 将 **virtio-win.iso** 文件作为存储设备附加到 Windows 虚拟机上。
   - 当 创建新 Windows 虚拟机 时，请使用 **virt-install** 命令选项来附加文件。

- 当在现有 Windows 虚拟机上安装驱动程序时，请使用 **virt-xml** 工具将文件附加为 CD-ROM：

```
# virt-xml WindowsVM --add-device --disk virtio-win.iso,device=cdrom
Domain 'WindowsVM' defined successfully.
```

其它资源

- 在 Windows 客户机操作系统中安装 virtio 驱动程序 。

21.2.1.3. 在 Windows 客户端中安装 virtio 驱动程序

要在 Windows 客户机操作系统上安装 KVM **virtio** 驱动程序，您必须添加一个包含驱动程序的存储设备（创建虚拟机(VM)时或之后），并在 Windows 客户机操作系统中安装驱动程序。

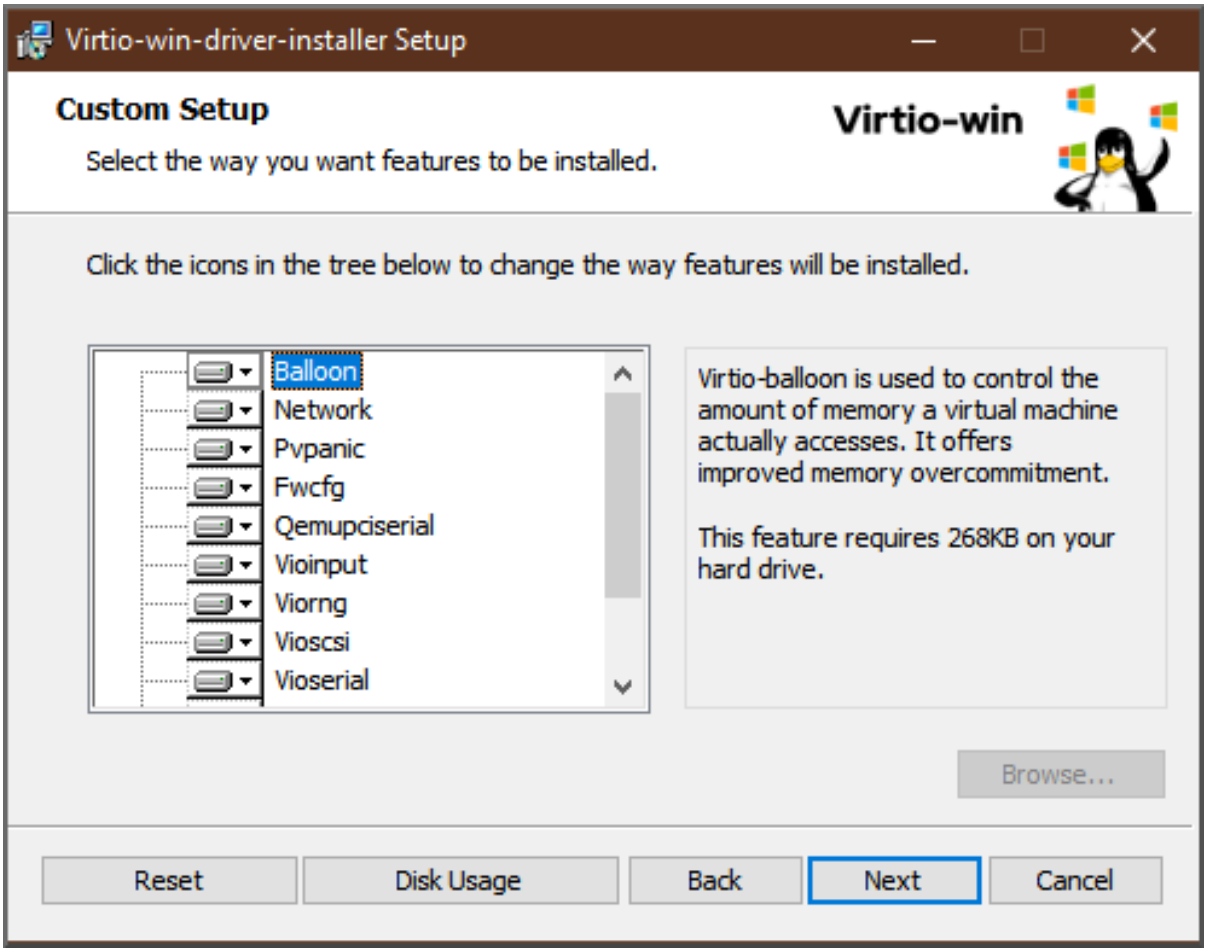
此流程提供了使用图形界面安装驱动程序的说明。您也可以使用 Microsoft Windows Installer(MSI) 命令行界面。

先决条件

- 带有 KVM **virtio** 驱动程序的安装介质必须附加到虚拟机。有关准备该介质的步骤，请参考在主机中准备 virtio 驱动程序安装介质。

流程

1. 在 Windows 客户机操作系统中，打开 **File Explorer** 应用程序。
2. 单击 **This PC**。
3. 在 **Devices and drives** 窗格中，打开 **virtio-win** 介质。
4. 根据在虚拟机上安装的操作系统，运行其中一个安装程序：
   - 如果使用 32 位操作系统，请运行 **virtio-win-gt-x86.msi** 安装程序。
   - 如果使用 64 位操作系统，请运行 **virtio-win-gt-x64.msi** 安装程序。
5. 在打开的 **Virtio-win-driver-installer** 设置向导中，按照显示的说明进行操作，直到您到达 **Custom Setup** 那步。



6. 在 Custom Setup 窗口中，选择您要安装的设备驱动程序。会自动选择建议的驱动程序集，驱动程序的描述显示在列表右侧。
7. 单击 **next**，然后单击 **Install**。
8. 安装完成后，点**完成**。
9. 重启虚拟机以完成驱动程序安装。

验证

1. 在 Windows 虚拟机上，进入 **Device Manager** ：
   a. 点 Start
   b. 搜索 **Device Manager**
2. 确保设备使用正确的驱动程序：
   a. 点击设备打开 **Driver Properties** 窗口。
   b. 导航到 **Driver** 选项卡。
   c. 单击 **Driver Details**。

后续步骤

- 如果安装了 NetKVM 驱动程序，您可能还需要配置 Windows 客户机的网络参数。如需更多信息，请参阅配置 NetKVM 驱动程序参数。

21.2.1.4. 更新 Windows 客户机上的 virtio 驱动程序

要更新 Windows 客户机操作系统(OS)上的 KVM **virtio** 驱动程序，如果 Windows OS 版本支持，您可以使用 **Windows Update** 服务。如果不支持，请从附加到 Windows 虚拟机(VM)的 **virtio** 驱动程序安装介质中重新安装驱动程序。

先决条件

- 安装了 virtio 驱动程序的 Windows 客户机操作系统。
- 如果没有使用 **Windows Update**，则必须将带有最新 KVM **virtio** 驱动程序的安装介质附加到 Windows 虚拟机上。有关准备该介质的步骤，请参考在主机中准备 virtio 驱动程序安装介质。

步骤 1：使用 Windows Update 更新驱动程序

在 Windows 10 上，Windows Server 2016 及更新版本的操作系统，使用 **Windows Update** 图形界面检查驱动程序更新是否可用：

1. 启动 Windows 虚拟机,并登录到其客户机操作系统。
2. 进入dao **Optional updates** 页面：
   Settings → Windows Update → Advanced options → Optional updates
3. 从 **Red Hat, Inc.** 安装所有更新。

流程 2：通过重新安装来更新驱动程序

在 Windows 10 和 Windows Server 2016 之前的操作系统上，或者操作系统无法访问 **Windows 更新**，请重新安装驱动程序。这会将 Windows 客户机操作系统的网络配置恢复到默认值(DHCP)。如果要保留自定义网络配置，还需要创建一个备份并使用 **netsh** 工具恢复它：

1. 启动 Windows 虚拟机,并登录到其客户机操作系统。
2. 打开 Windows 命令提示符：
   a. 使用 **Super+R** 键盘快捷键。
   b. 在显示的窗口中，输入 **cmd** 并按 **Ctrl+Shift+Enter** 以管理员身份运行。
3. 使用 Windows 命令提示备份 OS 网络配置：

   ```
   C:\WINDOWS\system32\netsh dump > backup.txt
   ```

4. 从附加的安装介质重新安装 KVM **virtio** 驱动程序。执行以下操作之一：
   - 使用 Windows 命令提示重新安装驱动程序，其中 *X* 是安装介质盘符。以下命令安装所有 **virtio** 驱动程序。
     - 如果使用 64 位 vCPU：

       ```
       C:\WINDOWS\system32\msiexec.exe /i X:\virtio-win-gt-x64.msi /passive /norestart
       ```

     - 如果使用 32 位 vCPU：

       ```
       C:\WINDOWS\system32\msiexec.exe /i X:\virtio-win-gt-x86.msi /passive /norestart
       ```

- 使用图形界面 重新安装驱动程序，而不重启虚拟机。

5. 使用 Windows 命令提示恢复 OS 网络配置：

```
C:\WINDOWS\system32\netsh -f backup.txt
```

6. 重启虚拟机以完成驱动程序安装。

其它资源

- 有关 Windows Update 的 Microsoft 文档

21.2.1.5. 在 Windows 客户机上启用 QEMU 客户机代理

要允许 RHEL 主机在 Windows 虚拟机(VM)上执行 某些操作的子集，您必须启用 QEMU 客户机代理(GA)。要做到这一点，将包含 QEMU 客户机代理安装程序的存储设备添加到现有虚拟机上或在创建新虚拟机时，在 Windows 客户机操作系统上安装驱动程序。

要使用图形界面安装客户机代理(GA)，请参阅以下流程 。要在命令行界面中安装 GA，请使用 Microsoft Windows 安装程序(MSI)。

先决条件

- 带有客户机代理的安装介质被附加到虚拟机。有关准备该介质的步骤，请参考在主机中准备 virtio 驱动程序安装介质。

流程

1. 在 Windows 客户机操作系统中，打开 **File Explorer** 应用程序。
2. 单击 **This PC**。
3. 在 **Devices and drives** 窗格中，打开 **virtio-win** 介质。
4. 打开 **guest-agent** 文件夹。
5. 根据虚拟机上安装的操作系统，运行以下安装程序之一：
   - 如果使用 32 位操作系统，请运行 **qemu-ga-i386.msi** 安装程序。
   - 如果使用 64 位操作系统，请运行 **qemu-ga-x86_64.msi** 安装程序。
6. **可选** ：如果您要使用半虚拟化串行驱动程序(**virtio-serial**)作为主机和 Windows 客户机之间的通信接口，请验证 **virtio-serial** 驱动程序是否已安装在 Windows 客户机上。有关安装 **virtio** 驱动程序的更多信息，请参阅 在 Windows 客户端上安装 virtio 驱动程序 。

验证

1. 在 Windows 虚拟机上，进入到 **Services** 窗口。
   computer Management > Services
2. 确保 **QEMU 客户机代理** 服务的状态为 **Running**。

其它资源

- 需要 QEMU 客户机代理的虚拟化功能

21.2.2. 启用 Hyper-V enlightenments

Hyper-V enlightenments 为 KVM 提供了一个模拟 Microsoft Hyper-V hypervisor的方法。这提高了 Windows 虚拟机的性能。

以下小节提供了有关支持的 Hyper-V enlightenments 以及如何启用它们的信息。

21.2.2.1. 在 Windows 虚拟机上启用 Hyper-V 激活

Hyper-Vlightenments 在 RHEL 9 主机中运行的 Windows 虚拟机(VM)中提供更好的性能。有关如何启用它们的步骤，请查看以下操作。

流程

1. 使用 **virsh edit** 命令打开虚拟机的 XML 配置。例如：

```
# virsh edit windows-vm
```

2. 将以下 **<hyperv>** 子部分添加到 XML 的 **<features>** 部分：

```
<features>
  [...]
  <hyperv>
    <relaxed state='on'/>
    <vapic state='on'/>
    <spinlocks state='on' retries='8191'/>
    <vpindex state='on'/>
    <runtime state='on' />
    <synic state='on'/>
    <stimer state='on'>
      <direct state='on'/>
    </stimer>
    <frequencies state='on'/>
    <reset state='on'/>
    <relaxed state='on'/>
    <time state='on'/>
    <tlbflush state='on'/>
    <reenlightenment state='on'/>
    <stimer state='on'>
      <direct state='on'/>
    </stimer>
    <ipi state='on'/>
    <crash state='on'/>
    <evmcs state='on'/>
  </hyperv>
  [...]
</features>
```

如果 XML 已包含 **<hyperv>** 子部分，请按上所示进行修改。

3. 按如下方式更改配置的 **clock** 部分：

```
<clock offset='localtime'>
```

```
  ...
  <timer name='hypervclock' present='yes'/>
</clock>
```

4. 保存并退出 XML 配置。

5. 如果虚拟机正在运行，请重新启动它。

验证

- 使用 **virsh dumpxml** 命令显示正在运行的虚拟机的 XML 配置。如果包括以下段，则虚拟机上启用了 Hyper-V enlightenments 。

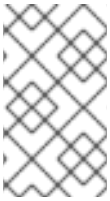
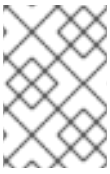
```
<hyperv>
  <relaxed state='on'/>
  <vapic state='on'/>
  <spinlocks state='on' retries='8191'/>
  <vpindex state='on'/>
  <runtime state='on' />
  <synic state='on'/>
  <stimer state='on'/>
  <frequencies state='on'/>
  <reset state='on'/>
  <relaxed state='on'/>
  <time state='on'/>
  <tlbflush state='on'/>
  <reenlightenment state='on'/>
  <stimer state='on'>
    <direct state='on'/>
  </stimer>
  <ipi state='on'/>
  <crash state='on'/>
  <evmcs state='on'/>
</hyperv>

<clock offset='localtime'>
  ...
  <timer name='hypervclock' present='yes'/>
</clock>
```

21.2.2.2. 可配置 Hyper-V enlightenments

您可以配置特定的 Hyper-V 功能来优化 Windows 虚拟机。下表提供了有关这些可配置 Hyper-V 功能及其值的信息。

表 21.1. 可配置 Hyper-V 功能

Enlightenment	描述	值

Enlightenment	描述	值
crash	向虚拟机提供 MSR，可用于在虚拟机崩溃时存储信息和日志。QEMU 日志中提供的信息。  **注意** 如果启用了 hv_crash，则不会创建 Windows 崩溃转储。	on, off
evmcs	在 L0(KVM)和 L1(Hyper-V) hypervisor 之间实施半虚拟化协议，从而使 L2 更快地退出到 hypervisor 。  **注意** 这个功能只供 Intel 处理器使用。	on, off
frequencies	启用 Hyper-V frequency Machine Specific Registers (MSR)。	on, off
ipi	启用半虚拟化处理器间中断(IPI)支持。	on, off
no-nonarch-coresharing	通知客户端操作系统：虚拟处理器永远不会共享物理核，除非它们被报告为同级 SMT 线程。Windows 和 Hyper-V 客户机需要这些信息才能适当地减轻与并发多线程(SMT)相关的 CPU 漏洞。	on, off, auto
reenlightenment	仅在迁移期间发生时间戳计数器(TSC)频率更改时通知。它还允许 guest 继续使用旧频率，直到准备好切换至新频率。	on, off
relaxed	禁用 Windows 完整性检查，当虚拟机在负载较重的主机上运行时，该检查通常会导致 BSOD 。这和 Linux 内核选项 no_timer_check 类似，它会在 Linux 在 KVM 中运行时自动启用。	on, off

Enlightenment	描述	值
runtime	设定运行客户机代码以及代表客户端代码的处理器时间。	on, off
spinlocks	• 虚拟机的操作系统用来通知 Hyper-V，调用虚拟处理器正在尝试获取一个可能由同一分区中其他虚拟处理器占有的资源。 • Hyper-V 用来向虚拟机的操作系统指明在向 Hyper-V 指示过度旋转情形之前应该尝试获取 spinlock 的次数。	on, off
stimer	为虚拟处理器启用合成计时器。请注意，某些 Windows 版本在未提供这种启示时将恢复使用 HPET（或在 HPET 不可用时使用 RTC），这可能导致大量 CPU 消耗，即使虚拟 CPU 处于空闲状态。	on, off
stimer-direct	当通过正常的中断发送过期事件时，启用合成计时器。	on, off.
synic	与 stimer 一起激活合成计时器。Windows 8 以周期性模式使用此功能。	on, off
time	启用以下虚拟机可用的特定于 Hyper-V 的时钟源, • 基于 MSR 的 82 个 Hyper-V 时钟源 (HV_X64_MSR_TIME_REF_COUNT, 0x40000020) • 参考通过 MSR 启用的 TSC 83 页面 (HV_X64_MSR_REFERENCE_TSC，0x40000021)	on, off
tlbflush	清除虚拟处理器的 TLB。	on, off
vapic	启用虚拟 APIC，其提供 MSR 对高使用率、内存映射高级可编程中断控制器(APIC)寄存器的加速访问。	on, off

Enlightenment	描述	值
vendor_id	设置 Hyper-V 厂商 id。	• on, off • id 值 - 最多 12 个字符的字符串
vpindex	启用虚拟处理器索引。	on, off

21.2.3. 配置 NetKVM 驱动程序参数

安装 NetKVM 驱动程序后，您可以对其进行配置以更好地适合您的环境。以下流程中列出的参数可以使用 Windows 设备管理器(**devmgmt.msc**)进行配置。

重要

修改驱动程序的参数会导致 Windows 重新加载该驱动程序。这会中断现有的网络活动。

先决条件

- NetKVM 驱动程序安装在虚拟机上。
  如需更多信息，请参阅为 Windows 虚拟机安装 KVM 半虚拟驱动程序。

流程

1. 打开 Windows Device Manager。
   有关打开设备管理器的详情，请参考 Windows 文档。
2. 找到 **Red Hat VirtIO Ethernet Adapter**。
   a. 在 Device Manager 窗口中，点 Network adapters 旁边的 **+**。
   b. 在网络适配器列表中，双击 **Red Hat VirtIO Ethernet Adapter**。
   该设备的 **Properties** 窗口将打开。
3. 查看设备参数。
   在 **Properties** 窗口中，点击 **Advanced** 选项卡。
4. 修改设备参数。
   a. 点击您要修改的参数。
   此时会显示那个参数的选项。
   b. 根据需要修改选项。
   有关 NetKVM 参数选项的详情，请参考 NetKVM 驱动程序参数。
   c. 点 **OK** 保存更改。

21.2.4. NetKVM 驱动程序参数

下表提供了有关可配置的 NetKVM 驱动程序日志记录参数的信息。

表 21.2. 日志参数

参数	描述 2
Logging.Enable	确定是否启用日志记录的布尔值。默认值为 Enabled。
Logging.Level	定义日志级别的整数。当整数增加时，日志的详细程度也会增加。 ● 默认值为 0（仅错误）。 ● 1-2 添加配置信息。 ● 3-4 添加数据包流信息。 ● 5-6 添加中断以及 DPC 级别追踪信息。  **注意** 高日志级别会减慢您的虚拟机速度。

下表提供了有关可配置的 NetKVM 驱动程序初始参数的信息。

表 21.3. 初始参数

参数	描述
分配 MAC	为半虚拟 NIC 定义本地管理的 MAC 地址的字符串。默认不设置。
Init.Do802.1PQ	启用 Priority/VLAN 标签填充和删除支持的布尔值。默认值为 Enabled。
Init.MaxTxBuffers	代表将被分配的 TX 环描述符数的整数。值受 QEMU 的 Tx 队列的大小的限制。 默认值为 1024。 有效值有：16、32、64、128、256、512 和 1024。
init.MaxRxBuffers	代表将要分配的 RX 环描述符数的整数。值受 QEMU 的 Tx 队列的大小的限制。 默认值为 1024。 有效值为：16、32、64、128、256、512、1024、2048 和 4096。

参数	描述
Offload.Tx.Checksum	指定 TX 校验和卸载功能。 在 Red Hat Enterprise Linux 9 中，这个参数的有效值为： ● All （默认），其为 IPv4 和 IPv6 启用 IP、TCP 和 UDP 校验和卸载 ● TCP/UDP (v4,v6)，其为 IPv4 和 IPv6 启用 TCP 和 UDP 校验和卸载 ● TCP/UDP (v4)，其仅为 IPv4 启用 TCP 和 UDP 校验和卸载 ● TCP(v4)，其仅为 IPv4 启用 TCP 检验和卸载
Offload.Rx.Checksum	指定 RX 校验和卸载功能。 在 Red Hat Enterprise Linux 9 中，这个参数的有效值为： ● All （默认），其为 IPv4 和 IPv6 启用 IP、TCP 和 UDP 校验和卸载 ● TCP/UDP (v4,v6)，其为 IPv4 和 IPv6 启用 TCP 和 UDP 校验和卸载 ● TCP/UDP (v4)，其仅为 IPv4 启用 TCP 和 UDP 校验和卸载 ● TCP(v4)，其仅为 IPv4 启用 TCP 检验和卸载
Offload.Tx.LSO	指定 TX 大片段卸载(LSO)功能。 在 Red Hat Enterprise Linux 9 中，这个参数的有效值为： ● Maximal（默认），其为 TCPv4 和 TCPv6 启用 LSO 卸载 ● IPv4，其只为 TCPv4 启用 LSO 卸载 ● 禁用禁用了 LSO 卸载的功能

参数	描述
MinRxBufferPercent	指定 RX 队列中可用缓冲区的最小数（以 RX 缓冲区总数的百分比表示）。如果实际的可用缓冲区数低于该值，则 NetKVM 驱动程序向操作系统指示低资源状况（请求操作系统尽快返回 RX 缓冲区） 最小值（默认）- **0**，意味着驱动程序永远不会指示低资源状况。 最大值 - **100**，意味着驱动程序一直指示低资源状况。

其它资源

- INF 枚举关键字
- 可编辑的 INF 关键字

21.2.5. 在 Windows 虚拟机中优化后台进程

要优化运行 Windows OS 的虚拟机(VM)的性能，您可以配置或禁用各种 Windows 进程。

警告

如果您更改配置，某些进程可能无法按预期工作。

流程

您可以通过执行以下任一组合来优化 Windows 虚拟机：

- 删除未使用的设备，如 USB 或 CD-ROM，并禁用端口。
- 禁用后台服务，如 SuperFetch 和 Windows Search。有关停止服务的详情，请参考禁用系统服务或 Stop-Service。
- 禁用 **useplatformclock**。为此，请运行以下命令：

```
# bcdedit /set useplatformclock No
```

- 检查和禁用不必要的调度任务，如调度的磁盘清除。有关如何操作的更多信息，请参阅 禁用调度任务。
- 确定磁盘没有加密。
- 减少服务器应用程序的周期性活动。您可以编辑对应的计时器。如需更多信息，请参阅多媒体计时器。
- 关闭虚拟机上的 Server Manager 应用程序。

- 禁用 antivirus 软件。请注意，禁用 antivirus 可能会破坏虚拟机的安全。
- 禁用屏保。
- 在没有使用时，仍然将 Windows OS 保持在登录屏幕中。

21.3. 在 WINDOWS 虚拟机中启用标准硬件安全性

要保护 Windows 虚拟机，您可以使用 Windows 设备的标准硬件功能启用基本级别安全性。

先决条件

- 请确定您安装了最新的 WHQL 认证的 VirtIO 驱动程序。
- 确保虚拟机固件支持 UEFI 引导。
- 在您的主机上安装 **edk2-OVMF** 软件包。

```
# {PackageManagerCommand} install edk2-ovmf
```

- 在您的主机上安装 **vTPM** 软件包。

```
# {PackageManagerCommand} install swtpm libtpms
```

- 确保虚拟机使用 Q35 机器架构。
- 请确定您有 Windows 安装介质。

流程

1. 通过在虚拟机 XML 配置中的 **<devices>** 部分中添加以下参数来启用 TPM 2.0。

```
<devices>
[...]
  <tpm model='tpm-crb'>
    <backend type='emulator' version='2.0'/>
  </tpm>
[...]
</devices>
```

2. 在 UEFI 模式中安装 Windows。有关如何操作的更多信息，请参阅 创建 SecureBoot 虚拟机。
3. 在 Windows 虚拟机上安装 VirtIO 驱动程序。有关如何操作的更多信息，请参阅 在 Windows 客户端上安装 virtio 驱动程序。
4. 在 UEFI 中，启用安全引导。有关如何操作的更多信息，请参阅 安全引导。

验证

- 确定 Windows 机器中的设备**安全性**页面显示以下信息：
  settings > Update & Security > Windows Security > device Security

> **Your device meets the requirements for standard hardware security.**

21.4. 在 WINDOWS 虚拟机上启用增强的硬件安全

为进一步保护 Windows 虚拟机(VM)，您可以启用基于虚拟化的代码完整性保护，也称为 hypervisor 保护的代码完整性(HVCI)。

先决条件

- 确保启用了标准硬件安全。如需更多信息，请参阅在 Windows 虚拟机上启用标准硬件安全性。
- 确保启用了 Hyper-V enlightenments。如需更多信息，请参阅启用 Hyper-Vlightenments。

流程

1. 打开 Windows 虚拟机的 XML 配置。以下示例打开 *Example-L1* 虚拟机的配置：

```
# virsh edit Example-L1
```

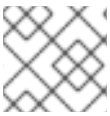
2. 在 **<cpu>** 部分下，指定 CPU 模式并添加策略标记。

重要

- 对于 Intel CPU，启用 **vmx** 策略标记。
- 对于 AMD CPU，启用 **svm** 策略标记。
- 如果您不想指定一个自定义 CPU，您可以将 **<cpu mode>** 设置为 **host-passthrough**。

```
<cpu mode='custom' match='exact' check='partial'>
  <model fallback='allow'>Skylake-Client-IBRS</model>
  <topology sockets='1' dies='1' cores='4' threads='1'/>
  <feature policy='require' name='vmx'/>
</cpu>
```

3. 保存 XML 配置并重启虚拟机。
4. 在虚拟机操作系统中，进入到 **Core 隔离详情**页 ：
settings > Update & Security > Windows Security > Device Security > Core isolated details
5. 切换开关以启用 **内存完整性**。
6. 重启虚拟机。



注意

有关启用 HVCI 的其他方法，请查看相关的 Microsoft 文档。

验证

- 确保 Windows 虚拟机上的 **Device Security** 页面显示以下信息：
settings > Update & Security > Windows Security > device Security

> **Your device meets the requirements for enhanced hardware security.**

- 或者，检查有关 Windows 虚拟机的系统信息：

  a. 在命令提示符下运行 **msinfo32.exe**。

  b. 检查是否在 **基于虚拟化的安全服务运行** 下列出了 **凭据保护，Hypervisor 强制的代码完整性**。

21.5. 后续步骤

- 要使用用于访问、编辑和创建 Windows 虚拟机的虚拟机磁盘或其他磁盘镜像的工具，请在主机机器上安装 **libguestfs-tools** 和 **libguestfs-winsupport** 软件包：

```
$ sudo dnf install libguestfs-tools libguestfs-winsupport
```

- 要使用用于访问、编辑和创建 Windows 虚拟机的虚拟机磁盘或其他磁盘镜像的工具，请在主机机器上安装 **guestfs-tools** 和 **guestfs-winsupport** 软件包：

```
$ sudo dnf install guestfs-tools guestfs-winsupport
```

- 要在 RHEL 9 主机及其 Windows 虚拟机间共享文件，您可以使用 virtiofs 或 NFS。

第 22 章 创建嵌套虚拟机

如果您需要与本地主机运行的不同的主机操作系统，您可以使用嵌套虚拟机(VM)。这消除了对其他物理硬件的需求。

警告

在大多数环境中，嵌套虚拟化在 RHEL 9 中仅作为 技术预览 提供。

有关支持和不支持的环境的详情，请参阅 对嵌套虚拟化 的支持限制。

22.1. 什么是嵌套虚拟化？

通过嵌套虚拟化，您可以在其他虚拟机中运行虚拟机(VM)。在物理主机上运行的标准虚拟机也可以充当第二个 hypervisor，并创建自己的虚拟机。

嵌套虚拟化术语

第 0 级(L0)

物理主机，裸机。

第 1 级(L1)

在 **L0** 物理主机上运行的标准虚拟机，可充当额外的虚拟主机。

第 2 级(L2)

在 **L1** 虚拟主机上运行的嵌套虚拟机。
重要： 虚拟化的第二级严重限制 **L2** 虚拟机的性能。因此，嵌套虚拟化主要用于开发和测试场景，例如：

- 在受限环境中调试虚拟机监控程序
- 在有限的物理资源中测试较大的虚拟部署

警告

在大多数环境中，嵌套虚拟化在 RHEL 9 中仅作为 技术预览 提供。

有关支持和不支持的环境的详情，请参阅 对嵌套虚拟化 的支持限制。

其它资源

- 对嵌套虚拟化的支持限制

22.2. 对嵌套虚拟化的支持限制

在大多数环境中，嵌套虚拟化在 RHEL 9 中仅作为技术预览提供。

但是，您可以使用具有用于 Linux (WSL2)的 Windows 子系统的 Windows 虚拟机(VM)，来在 Windows 虚拟机内创建虚拟 Linux 环境。此用例在特定条件下，在 RHEL 9 上完全支持。

要了解更多有关嵌套虚拟化的相关术语的信息，请参阅 什么是嵌套虚拟化？

支持的环境

要创建支持的嵌套虚拟化的部署，请在 RHEL 9 **L0** 主机上创建一个 **L1** Windows 虚拟机，并使用 WSL2 在 **L1** Windows 虚拟机中创建一个虚拟 Linux 环境。目前，这是唯一支持的嵌套环境。

重要

L0 主机必须是 Intel 或 AMD 系统。目前不支持其他架构，如 ARM 或 IBM Z。

您必须只使用以下操作系统版本之一：

在 L0 主机上：	在 L1 虚拟机上：
RHEL 9.2 及更新版本	带有 WSL2 的 Windows Server 2019
	带有 WSL2 的 Windows Server 2022
	带有 WSL2 的 Windows 10
	带有 WSL2 的 Windows 11

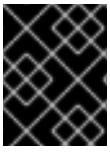
有关安装 WSL2 并选择支持的 Linux 发行版的说明，请参阅 Microsoft 文档。

要创建一个支持的嵌套环境，请使用以下流程之一：

- 在 Intel 上创建嵌套虚拟机
- 在 AMD 上创建嵌套虚拟机

技术预览环境

这些嵌套环境仅作为技术预览提供，并不被支持。



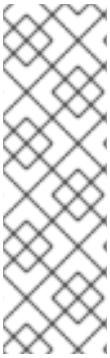
重要

L0 主机必须是 Intel、AMD 或 IBM Z 系统。嵌套虚拟化目前无法在其他架构上工作，如 ARM。

您必须只使用以下操作系统版本之一：

在 L0 主机上：	在 L1 虚拟机上：	在 L2 虚拟机上：
RHEL 9.2 及更新版本	RHEL 8.8 及更新版本	RHEL 8.8 及更新版本

在 L0 主机上 ：	在 L1 虚拟机上 ：	在 L2 虚拟机上 ：
	RHEL 9.2 及更新版本	RHEL 9.2 及更新版本
	带有 Hyper-V 的 Windows Server 2016	Windows Server 2019
	带有 Hyper-V 的 Windows Server 2019	Windows Server 2022
	带有 Hyper-V 的 Windows Server 2022	
	带有 Hyper-V 的 Windows 10	
	带有 Hyper-V 的 Windows 11	



注意

在其他 Red Hat Virtualization 产品中使用时，没有测试过创建 RHEL **L1** 虚拟机。它们是：

- Red Hat Virtualization
- Red Hat OpenStack Platform
- OpenShift Virtualization

要创建技术预览嵌套环境，请使用以下流程之一：

- 在 Intel 上创建嵌套虚拟机
- 在 AMD 上创建嵌套虚拟机
- 在 IBM Z 上创建嵌套的虚拟机

Hypervisor 限制

- 目前，红帽仅在 RHEL-KVM 上测试了嵌套。当 RHEL 被用作 **L0** hypervisor 时，您可以使用 RHEL 或 Windows 作为 **L1** hypervisor。
- 当在非 KVM **L0** hypervisor（如 VMware ESXi 或 Amazon Web Services (AWS)）上使用 **L1** RHEL 虚拟机时，在 RHEL 客户机操作系统中创建 **L2** 虚拟机还没有被测试，且可能无法正常工作。

功能限制

- 使用 **L2** 虚拟机作为 hypervisor，并创建 **L3** 客户机尚未被正确测试，预计无法正常工作。
- 如果在 **L0** 主机上启用了嵌套虚拟化，则迁移虚拟机目前无法在 AMD 系统上工作。
- 在 IBM Z 系统上，无法同时使用大页面支持存储和嵌套虚拟化。

```
# modprobe kvm hpage=1 nested=1
modprobe: ERROR: could not insert 'kvm': Invalid argument
# dmesg |tail -1
[90226.508366] kvm-s390: A KVM host that supports nesting cannot back its KVM guests with huge pages
```

- **L0** 主机上提供的一些功能可能无法用于 **L1** hypervisor 。

其他资源

- 什么是用于 Linux 的 Windows 子系统？
- 在 Intel 上创建嵌套虚拟机
- 在 AMD 上创建嵌套虚拟机
- 在 IBM Z 上创建嵌套的虚拟机

22.3. 在 INTEL 上创建嵌套虚拟机

按照以下步骤在 Intel 主机中启用和配置嵌套虚拟化。

警告

在大多数环境中，嵌套虚拟化在 RHEL 9 中仅作为 技术预览 提供。

有关支持和不支持的环境的详情，请参阅 对嵌套虚拟化 的支持限制。

先决条件

- 运行 L1 虚拟机的 L0 RHEL 9 主机。
- 管理程序 CPU 必须支持嵌套虚拟化。若要验证，可在 L0 hypervisor 上使用 **cat /proc/cpuinfo** 命令。如果命令的输出中包含 **vmx** 和 **ept** 标志，则可以创建 L2 虚拟机。这通常是 Intel Xeon v3 内核及更新版本上的情况。
- 确定在 L0 主机上启用了嵌套虚拟化：

```
# cat /sys/module/kvm_intel/parameters/nested
```

  - 如果命令返回 **1** 或 **Y**，则表示启用了该特性。跳过剩余的先决条件步骤，并继续执行流程部分。
  - 如果该命令返回 **0** 或者 **N**，而您的系统支持嵌套虚拟化，则使用以下步骤启用该功能。
    i. 卸载 **kvm_intel** 模块：

```
# modprobe -r kvm_intel
```

    ii. 激活嵌套功能：

```
# modprobe kvm_intel nested=1
```

iii. 现在启用嵌套功能，但只有在下次重启 L0 主机时才启用。要永久启用它，请在 **/etc/modprobe.d/kvm.conf** 文件中添加以下行：

```
options kvm_intel nested=1
```

流程

1. 为嵌套虚拟化配置 L1 虚拟机。

   a. 打开虚拟机的 XML 配置。以下示例打开 *Intel-L1* 虚拟机的配置：

   ```
   # virsh edit Intel-L1
   ```

   b. 通过编辑 **<cpu>** 元素将 VM 配置为使用 **host-passthrough** CPU 模式：

   ```
   <cpu mode='host-passthrough'/>
   ```

   如果您需要虚拟机使用特定的 CPU 型号，请将虚拟机配置为使用 **custom** CPU 模式。在 **<cpu>** 元素中，添加一个 **<feature policy='require' name='vmx'/>** 元素和一个在其中指定了 CPU 型号的 **<model>** 元素。例如：

   ```
   <cpu mode ='custom' match ='exact' check='partial'>
     <model fallback='allow'>Haswell-noTSX</model>
     <feature policy='require' name='vmx'/>
     ...
   </cpu>
   ```

2. 在 L1 虚拟机中创建 L2 虚拟机。要做到这一点，请遵循与 创建 L1 虚拟机时 相同的流程。

22.4. 在 AMD 上创建嵌套虚拟机

按照以下步骤在 AMD 主机中启用和配置嵌套虚拟化。

警告

在大多数环境中，嵌套虚拟化在 RHEL 9 中仅作为 技术预览 提供。

有关支持和不支持的环境的详情，请参阅 对嵌套虚拟化 的支持限制。

先决条件

- 运行 L1 虚拟机的 L0 RHEL 9 主机。
- 管理程序 CPU 必须支持嵌套虚拟化。若要验证，可在 L0 hypervisor 上使用 **cat /proc/cpuinfo** 命令。如果命令的输出中包含 **svm** 和**npt** 标志，则可以创建 L2 虚拟机。这通常是 AMD EPYC 内核及更新版本上的情况。

- 确定在 L0 主机上启用了嵌套虚拟化：

```
# cat /sys/module/kvm_amd/parameters/nested
```

  - 如果命令返回 **1** 或 **Y**，则表示启用了该特性。跳过剩余的先决条件步骤，并继续执行流程部分。
  - 如果命令返回 **0** 或 **N**，请使用以下步骤启用该功能。
    i. 在 L0 主机上停止所有运行的虚拟机。
    ii. 卸载 **kvm_amd** 模块：

```
# modprobe -r kvm_amd
```

    iii. 激活嵌套功能：

```
# modprobe kvm_amd nested=1
```

    iv. 现在启用嵌套功能，但只有在下次重启 L0 主机时才启用。要永久启用它，请在 **/etc/modprobe.d/kvm.conf** 文件中添加以下内容：

```
options kvm_amd nested=1
```

流程

1. 为嵌套虚拟化配置 L1 虚拟机。
   a. 打开虚拟机的 XML 配置。以下示例将打开 *AMD-L1* 虚拟机的配置：

```
# virsh edit AMD-L1
```

   b. 通过编辑 **<cpu>** 元素将 VM 配置为使用 **host-passthrough** CPU 模式：

```
<cpu mode='host-passthrough'/>
```

   如果您需要虚拟机使用特定的 CPU 型号，请将虚拟机配置为使用 **custom** CPU 模式。在 **<cpu>** 元素中，添加一个 **<feature policy='require' name='svm'/>** 元素和一个在其中指定了 CPU 型号的 **<model>** 元素。例如：

```
<cpu mode="custom" match="exact" check="none">
  <model fallback="allow">EPYC-IBPB</model>
  <feature policy="require" name="svm"/>
  ...
</cpu>
```

2. 在 L1 虚拟机中创建 L2 虚拟机。要做到这一点，请遵循与 创建 L1 虚拟机时 相同的流程。

22.5. 在 IBM Z 上创建嵌套的虚拟机

按照以下步骤在 IBM Z 主机中启用和配置嵌套虚拟化。

注意

IBM Z 没有真正提供裸机 **L0** 主机。相反，用户系统是在逻辑分区(LPAR)上建立的，它已经是一个虚拟化系统，因此通常被称为 **L1**。但是，为了更好地与本指南中的其他架构保持一致，以下步骤参考 IBM Z，就像其提供 **L0** 主机一样。

要了解有关嵌套虚拟化的更多信息，请参阅：什么是嵌套虚拟化？

警告

在大多数环境中，嵌套虚拟化在 RHEL 9 中仅作为 技术预览 提供。

有关支持和不支持的环境的详情，请参阅 对嵌套虚拟化 的支持限制。

先决条件

- 运行 L1 虚拟机的 L0 RHEL 9 主机。
- 管理程序 CPU 必须支持嵌套虚拟化。若要验证确实是这种情况，请在 L0 hypervisor上使用 **cat /proc/cpuinfo** 命令。如果命令的输出中包含 **sie** 标志，则可以创建 L2 虚拟机。
- 确定在 L0 主机上启用了嵌套虚拟化：

```
# cat /sys/module/kvm/parameters/nested
```

  - 如果命令返回 **1** 或 **Y**，则表示启用了该特性。跳过剩余的先决条件步骤，并继续执行流程部分。
  - 如果命令返回 **0** 或 **N**，请使用以下步骤启用该功能。
    i. 在 L0 主机上停止所有运行的虚拟机。
    ii. 卸载 **kvm** 模块：

```
# modprobe -r kvm
```

    iii. 激活嵌套功能：

```
# modprobe kvm nested=1
```

    iv. 现在启用嵌套功能，但只有在下次重启 L0 主机时才启用。要永久启用它，请在 **/etc/modprobe.d/kvm.conf** 文件中添加以下行：

```
options kvm nested=1
```

流程

- 在 L1 虚拟机中创建 L2 虚拟机。要做到这一点，请遵循与 创建 L1 虚拟机时 相同的流程。

第 23 章 诊断虚拟机问题

使用虚拟机(VM)时，您可能会遇到不同严重等级的问题。有些问题可能很快且容易修复，而另外一些问题，您可能需要捕获与虚拟机相关的数据和日志来报告或诊断问题。

以下章节提供了有关生成日志、诊断一些常见虚拟机问题以及报告这些问题的详细信息。

23.1. 生成 LIBVIRT 调试日志

要诊断虚拟机(VM)问题，生成和查看 libvirt 调试日志会很有帮助。当请求支持解决与虚拟机相关的问题时，附加调试日志也很有用。

以下章节解释了 什么是调试日志，如何 将它们设为永久，在运行时启用它们，以及在报告问题时 附加它们。

23.1.1. 了解 libvirt 调试日志

调试日志是文本文件，其中包含关于虚拟机(VM)运行期间所发生的事件的数据。日志提供有关基本的服务器端功能方面的信息，如主机库和 libvirt 守护进程。日志文件还包含所有正在运行的虚拟机的标准错误输出(**stderr**)。

默认不启用 debug 日志记录，且必须在 libvirt 启动时启用。您可以对单个会话启用日志记录，也可以 永久启用记录。您还可以通过 修改守护进程运行时设置，在 libvirt 守护进程会话运行时启用日志。

在请求对虚拟机问题的支持时，附加 libvirt 调试日志也很有用。

23.1.2. 为 libvirt 调试日志启用持久性设置

您可以将 libvirt debug 日志记录配置为在 libvirt 启动时自动启用。默认情况下，**virtqemud** 是 RHEL 9 中的主要 libvirt 守护进程。要在 libvirt 配置中进行持久更改，您必须编辑位于 **/etc/libvirt** 目录中的 **virtqemud.conf** 文件。

注意

在某些情况下，例如，当您从 RHEL 8 升级时，**libvirtd** 可能仍为已启用的 libvirt 守护进程。在这种情况下，您必须编辑 **libvirtd.conf** 文件。

流程

1. 在编辑器中打开 **virtqemud.conf** 文件。
2. 根据您的要求替换或设置过滤器。

表 23.1. 调试过滤器的值

1	记录 libvirt 生成的所有消息。
2	记录所有非调试信息。
3	记录所有警告和错误消息。这是默认值。
4	仅记录错误消息。

例 23.1. 日志过滤器的守护进程设置示例

以下设置：

- 记录来自 **remote**,**util.json** 和 **rpc** 层的所有错误和警告消息
- 仅记录来自 **event** 层的错误消息。
- 将过滤的日志保存到 **/var/log/libvirt/libvirt.log**

```
log_filters="3:remote 4:event 3:util.json 3:rpc"
log_outputs="1:file:/var/log/libvirt/libvirt.log"
```

3. 保存并退出。
4. 重启 libvirt 守护进程。

```
$ systemctl restart virtqemud.service
```

23.1.3. 在运行时启用 libvirt 调试日志

您可以修改 libvirt 守护进程的运行时设置，以启用调试日志并将其保存到一个输出文件中。

这在无法重启 libvirt 守护进程时非常有用，因为重启解决了问题，或者因为有另一个进程（如迁移或备份）同时在运行。如果您要在不编辑配置文件或重启守护进程的情况下尝试命令，修改运行时设置也很有用。

先决条件

- 确保 **libvirt-admin** 软件包已安装。

流程

1. **可选：**备份活跃的日志过滤器集合。

```
# virt-admin -c virtqemud:///system daemon-log-filters >> virt-filters-backup
```

注意

建议您备份一组处于活动状态的过滤器，以便在生成日志后恢复它们。如果您没有恢复过滤器，则会继续记录消息，这可能会影响系统性能。

2. 使用 **virt-admin** 工具启用调试，并根据您的要求设置过滤器。

表 23.2. 调试过滤器的值

1	记录 libvirt 生成的所有消息。
2	记录所有非调试信息。
3	记录所有警告和错误消息。这是默认值。

4	仅记录错误消息。

例 23.2. 日志过滤器的 virt-admin 设置示例

以下命令：

- 记录来自 **远程**、**util.json** 和 **rpc** 层的所有错误和警告消息
- 仅记录来自 **event** 层的错误消息。

```
# virt-admin -c virtqemud:///system daemon-log-filters "3:remote 4:event 3:util.json 3:rpc"
```

3. 使用 **virt-admin** 工具将日志保存到特定的文件或目录中。
例如，以下命令将日志输出保存到 **/var/log/libvirt/** 目录中的 **libvirt.log** 文件中：

```
# virt-admin -c virtqemud:///system daemon-log-outputs "1:file:/var/log/libvirt/libvirt.log"
```

4. **可选：** 您还可以删除过滤器来生成包含所有与虚拟机相关的信息的日志文件。但不建议您这样做，因为这个文件可能包含由 libvirt 模块生成的大量冗余信息。
   - 使用 **virt-admin** 工具指定一组空白的过滤器。

```
# virt-admin -c virtqemud:///system daemon-log-filters
  Logging filters:
```

5. **可选：**使用备份文件将过滤器恢复到其原始状态。
使用保存的值执行第二步，以恢复过滤器。

23.1.4. 将 libvirt 调试日志附加到支持请求中

您可能需要请求额外的支持来诊断和解决虚拟机(VM)问题。强烈建议将调试日志附加到支持请求中，以确保支持团队可以访问所需的所有信息，从而快速解决与虚拟机相关的问题。

流程

- 要报告问题并请求支持，创建一个支持问题单。
- 根据遇到的问题，将以下日志与您的报告一起附加：
  - 对于 libvirt 服务的问题，请附加主机上的 **/var/log/libvirt/libvirt.log** 文件。
  - 对于特定虚拟机的问题，请附加对应的日志文件。
例如，对于 *testguest1* 虚拟机，请附加 **testguest1.log** 文件，该文件可在 **/var/log/libvirt/qemu/testguest1.log** 中找到。

其它资源

- 如何给红帽支持提供日志文件？

23.2. 转储虚拟机内核

要分析虚拟机(VM)崩溃或出现故障的原因，您可以将虚拟机内核转储到磁盘上的文件，以便稍后分析和诊断。

本节提供了简要的 内核转储介绍 ，并解释了如何将 虚拟机内核转储 到一个特定的文件。

23.2.1. 虚拟机内核转储的工作原理

虚拟机(VM)需要许多运行的进程来准确且高效地工作。在某些情况下，运行的虚拟机在使用时可能会意外终止或出现故障。重新启动虚拟机可能会导致数据被重置或丢失，从而难以诊断导致虚拟机崩溃的确切问题。

在这种情况下，您可以在重启虚拟机前使用 **virsh dump** 工具将虚拟机内核保存（或 *转储*）到一个文件中。内核转储文件包含虚拟机的原始物理内存镜像，其中包含有关虚拟机的详细信息。此信息可用于手动诊断，或使用 **crash** 等工具来诊断虚拟机问题。

其它资源

- **crash** 手册页
- **crash** Github 存储库

23.2.2. 创建虚拟机内核转储文件

虚拟机(VM)内核转储包含在任何给定的时间有关虚拟机状态的详细信息。此信息与虚拟机的快照类似，可以帮助您在虚拟机出现故障或突然关闭时检测问题。

先决条件

- 请确定您有足够的磁盘空间保存该文件。请注意，虚拟机消耗的空间取决于分配给虚拟机的 RAM 量。

流程

- 使用 **virsh dump** 工具。
  例如，以下命令将 **lander1** 虚拟机的内核、内存和 CPU 通用寄存器文件转储到 **/core/file** 目录中的 **gargantua.file** ：

```
# virsh dump lander1 /core/file/gargantua.file --memory-only
Domain 'lander1' dumped to /core/file/gargantua.file
```

重要

crash 工具不再支持 virsh dump 命令的默认文件格式。要使用 **crash** 分析内核转储文件，您必须使用 **--memory-only** 选项创建文件。

另外，在创建内核转储文件时，必须使用 **--memory-only** 选项将其附加到红帽支持问题单中。

故障排除

如果 **virsh dump** 命令失败并显示 **System is deadlocked on memory** 错误，请确保为内核转储文件分配了足够的内存。要做到这一点，请使用以下 **crashkernel** 选项值。或者，请勿使用 **crashkernel**，它会自动分配内核转储内存。

```
crashkernel=1G-4G:192M,4G-64G:256M,64G-:512M
```

其它资源

- **virsh dump --help** 命令
- **virsh** 手册页
- 开一个支持问题单

23.3. 回溯虚拟机进程

当与虚拟机(VM)相关的进程出现故障时，您可以使用 **gstack** 命令和进程标识符(PID)来生成一个故障进程的执行堆栈跟踪。如果该进程是线程组的一部分，那么也会跟踪所有线程。

先决条件

- 确保 **GDB** 软件包已安装。
  有关安装 **GDB** 和可用组件的详情，请参阅 安装 GNU 调试器。
- 请确定您知道要追踪进程的 PID。
  您可以使用 **pgrep** 命令，后跟进程的名称来查找 PID。例如：

```
# pgrep libvirt
22014
22025
```

流程

- 使用 **gstack** 工具，后跟您要回溯的进程的 PID。
  例如：以下命令追踪 PID 为 22014 的 libvirt 进程。

```
# gstack 22014
Thread 3 (Thread 0x7f33edaf7700 (LWP 22017)):
#0  0x00007f33f81aef21 in poll () from /lib64/libc.so.6
#1  0x00007f33f89059b6 in g_main_context_iterate.isra () from /lib64/libglib-2.0.so.0
#2  0x00007f33f8905d72 in g_main_loop_run () from /lib64/libglib-2.0.so.0
...
```

其它资源

- **gstack** 手册页
- GNU 调试器(GDB)

用于报告虚拟机问题并提供日志的其他资源

要请求额外的帮助和支持，您可以：

- 使用 **redhat-support-tool** 命令行选项、红帽门户网站 UI 或几种 FTP 方法开一个服务请求。
  - 要报告问题并请求支持,请参阅创建支持问题单。
- 提交服务请求时上传 SOS 报告以及日志文件。

这样可保证红帽支持工程师可以参考所需的全部诊断信息。

- 有关 SOS 报告的更多信息，请参阅 什么是 SOS 报告以及如何在 Red Hat Enterprise Linux 上创建它？
- 有关附加日志文件的详情，请参考如何为红帽支持提供文件？

第 24 章 RHEL 9 虚拟化的功能支持和限制

本文档提供了有关 Red Hat Enterprise Linux 9 (RHEL 9)虚拟化中的支持和限制的功能的信息。

24.1. RHEL 虚拟化支持如何工作

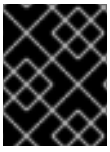
一组支持限制适用于 Red Hat Enterprise Linux 9 (RHEL 9) 的虚拟化。这意味着，在 RHEL 9 中使用虚拟机时，当您使用某些功能，或超过分配资源的一定数量时，红帽将不支持这些客户机，除非您有特定的订阅计划。

RHEL 9 虚拟化中推荐的功能 已经过测试并认证，可以在 RHEL 9 系统中使用 KVM 管理程序。因此，它们被完全支持，推荐在 RHEL 9 虚拟化中使用。

RHEL 9 虚拟化中不支持的功能 列出的功能不被支持，且不适用于在 RHEL 9 中使用。因此，红帽强烈建议您在带有 KVM 的 RHEL 9 中使用这些功能。

RHEL 9 虚拟化中的资源分配限制 列出了 RHEL 9 中 KVM 客户机上支持的特定资源的最大量。红帽不支持超过这些限制的客户端。

此外，除非另有说明，RHEL 9 虚拟化文档使用的所有功能和解决方案均受支持。然而，其中有些还没有进行充分测试，因此可能无法完全优化。



重要

这些限制不适用于红帽提供的其他虚拟化解决方案，如 OpenShift Virtualization 或 Red Hat OpenStack Platform (RHOSP)。

24.2. RHEL 9 虚拟化中的推荐功能

建议与 Red Hat Enterprise Linux 9 (RHEL 9) 中包含的 KVM 管理程序一起使用以下功能：

主机系统构架

只有在以下主机构架中才支持带有 KVM 的 RHEL 9：

- AMD64 和 Intel 64
- IBM Z - IBM z13 系统及更新版本
- ARM 64

对于使用 RHEL 9 作为 KVM 虚拟化主机的任何其他硬件架构都不支持，红帽强烈反对这样做。

客户机操作系统

红帽为使用特定客户机操作系统(OS)的 KVM 虚拟机提供支持。有关支持的客户机操作系统的详细列表，请参阅 红帽知识库中 的认证的客户机操作系统。

但请注意，默认情况下您的客户机操作系统不使用与您的主机相同的订阅。因此，您必须激活单独的许可或者订阅方可使客户机操作系统正常工作。

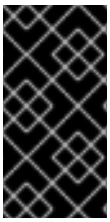
另外，您附加到虚拟机的直通设备必须被主机操作系统和客户机操作系统支持。

同样，对于部署的最佳功能，红帽建议虚拟机 XML 配置中定义的 CPU 型号和特性被主机操作系统和客户机操作系统支持。

要查看各种 RHEL 版本认证的 CPU 和其他硬件，请参阅 红帽生态系统目录。

机器类型

为确保虚拟机与您的主机架构兼容，且客户机操作系统最佳运行，虚拟机必须使用合适的机器类型。



重要

在 RHEL 9 中，不再支持 **pc-i440fx-rhel7.5.0** 及更早的机器类型（在早期的 RHEL 主版本中是默认的）。因此，在 RHEL 9 主机上尝试启动此类机器类型的虚拟机会失败，并显示 **unsupported configuration** 错误。如果您在将主机升级到 RHEL 9 后遇到这个问题，请参阅 红帽知识库。

当 使用命令行创建虚拟机 时，**virt-install** 工具提供了多种设置机器类型的方法。

- 当使用 **--os-variant** 选项时，**virt-install** 会自动选择主机 CPU 推荐的和客户机操作系统支持的机器类型。
- 如果不使用 **--os-variant** 或需要其他机器类型，请使用 **--machine** 选项来明确指定机器类型。
- 如果您指定了一个不受支持或与主机不兼容的 **--machine** 值，**virt-install** 将会失败，并显示一条错误信息。

在支持的架构上推荐的 KVM 虚拟机的机器类型，以及 **--machine** 选项的相应值如下。*Y* 代表 RHEL 9 的最新次版本。

- 在 Intel 64 and AMD64 (x86_64): **pc-q35-rhel9.*Y*.0 → --machine=q35**
- 在 IBM Z (s390x): **s390-ccw-virtio-rhel9.*Y*.0 → --machine=s390-ccw-virtio**
- 在 ARM 64 上：virt-rhel9.*Y*.0 → **--machine=virt**

获取现有虚拟机的机器类型：

```
# virsh dumpxml VM-name | grep machine=
```

要查看主机上支持的完整的机器类型列表：

```
# /usr/libexec/qemu-kvm -M help
```

其他资源

- RHEL 9 虚拟化不支持的功能
- RHEL 9 虚拟化中的资源分配限制

24.3. RHEL 9 虚拟化不支持的功能

Red Hat Enterprise Linux 9 (RHEL 9) 中的 KVM hypervisor 不支持以下特性：

重要

这些限制可能不适用于红帽提供的其他虚拟化解决方案，如 OpenShift Virtualization 或 Red Hat OpenStack Platform (RHOSP)。

其他虚拟化解决方案支持的功能如下：

主机系统构架

没有在RHEL 9 虚拟化中推荐的功能中列出的任何主机构架上都不支持带有 KVM 的 RHEL 9。

客户机操作系统

RHEL 9 主机上不支持使用以下客户机操作系统(OS)的 KVM 虚拟机(VM)：

- Windows 8.1 及更早版本
- Windows Server 2012 R2 及更早版本
- macOS
- 用于 x86 系统的 Solaris
- 2009 年前发布的任何操作系统

有关 RHEL 主机上支持的客户机操作系统的列表和其他虚拟化解决方案，请参阅 Red Hat OpenStack Platform、Red Hat Virtualization、OpenShift Virtualization 和带有 KVM 的 Red Hat Enterprise Linux 中认证的客户机操作系统。

在容器中创建虚拟机

红帽不支持在包含 RHEL 9 hypervisor 元素（如 **QEMU** 模拟器或 **libvirt** 软件包）的任何类型的容器中创建 KVM 虚拟机。

要在容器中创建虚拟机，红帽建议使用 OpenShift Virtualization 产品。

特定的 virsh 命令和选项

不是与 **virsh** 工具一起使用的每个参数都经过了红帽测试并认证为生产环境就绪。因此，红帽文档未明确推荐的 **virsh** 命令和选项可能无法正常工作，红帽建议不要在生产环境中使用它们。

值得注意的是，不受支持的 **virsh** 命令包括：

- **virsh iface-*** 命令，如 **virsh iface-start** 和 **virsh iface-destroy**
- **virsh blkdeviotune**
- **virsh snapshot-*** 命令，如 **virsh snapshot-create** 和 **virsh snapshot-revert**

QEMU 命令行

QEMU 是 RHEL 9 中虚拟化架构的一个基本组件，但难以手动管理，而不正确的 QEMU 配置可能导致安全漏洞。因此，红帽不支持使用 **qemu-*** 命令行工具，如 **qemu-kvm**。反之，使用 *libvirt* 工具程序（如 **virt-install**, **virt-xml**）以及支持的 **virsh** 命令，根据最佳实践编配 QEMU。但是对于 虚拟磁盘镜像的管理，支持 **qemu-img** 工具。

vCPU 热拔

从正在运行的虚拟机中删除虚拟 CPU(vCPU)，也称为 vCPU 热插拔，在 RHEL 9 中不支持。

内存热拔

删除附加在正在运行的虚拟机上的内存设备，也称为内存热插拔，在 RHEL 9 中不支持。

QEMU 端的 I/O 节流

使用 **virsh blkdeviotune** 工具为虚拟磁盘上的操作配置最大输入和输出级别，也称为 QEMU 侧 I/O 节流，在 RHEL 9 中不支持。

要在 RHEL 9 中设置 I/O 节流，请使用 **virsh blkiotune**。这也被称为 libvirt-side I/O 节流。具体步骤请查看虚拟机中的磁盘 I/O 节流。

其他解决方案：

- RHOSP 还支持 QEMU 边的 I/O 节流。详情请参阅 RHOSP 存储指南 中的 在磁盘上设置资源限制 和 **使用服务质量规范** 章节。
- 此外，OpenShift Virtualizaton 也支持 QEMU 侧 I/O 节流。

存储动态迁移

在主机之间迁移正在运行的虚拟机的磁盘镜像在 RHEL 9 中不支持。

其他解决方案：

- RHOSP 中支持存储实时迁移，但有一些限制。详情请参阅迁移卷。

内部快照

为虚拟机创建和使用内部快照在 RHEL 9 中已弃用，因此不建议在生产环境中使用。反之，请使用外部快照。详情请参阅 support-limitations-for-virtual-machine-snapshots_creating-virtual-machine-snapshots。

其他解决方案：

- RHOSP 支持实时快照。详情请参阅把虚拟机导入到 overcloud 部分。
- OpenShift Virtualization 上也支持实时快照。

vhost 数据路径加速

在 RHEL 9 主机上，可以为 virtio 设备配置 vHost Data Path Acceleration(vDPA)，但红帽目前不支持此功能，并且强烈建议在生产环境中使用它。

vhost-user

RHEL 9 不支持用户空间 vHost 接口的实现。

其他解决方案：

- RHOSP 支持 **vhost-user**，但只适用于 **virtio-net** 接口。详情请查看 virtio-net 实现和 vhost 用户端口。
- OpenShift Virtualization 还支持 **vhost-user**。

S3 和 S4 系统电源状态

不支持将虚拟机挂起到 **Suspend to RAM** (S3) 或 **Suspend to disk** (S4)系统电源状态。请注意，这些功能默认情况下是禁用的，启用这些功能将使您的虚拟机不被红帽支持。

请注意，S3 和 S4 状态目前还不支持红帽提供的其它虚拟化解决方案。

多路径 vDisk 中的 s3-PR

RHEL 9 不支持在多路径 vDisk 上的 SCSI3 持久保留(S3-PR)。因此，RHEL 9 不支持 Windows 集群。

virtio-crypto

在 RHEL 9 中使用 *virtio-crypto* 设备不被支持，RHEL 强烈建议不使用它。

请注意，红帽提供的其他虚拟化解决方案还不支持 *virtio-crypto* 设备。

virtio-multitouch-device, virtio-multitouch-pci

在 RHEL 9 中使用 *virtio-multitouch-device* 和 *virtio-multitouch-pci* 设备不被支持，RHEL 强烈建议不使用它们。

增量实时备份

配置仅保存自上次备份以来虚拟机变化（也称为增量实时备份）的虚拟机备份在 RHEL 9 中不支持，红帽强烈建议不使用它。

net_failover

使用 **net_failover** 驱动程序设置自动网络设备故障切换机制，在 RHEL 9 中不支持。

请注意，红帽提供的其他虚拟化解决方案目前还不支持 **net_failover**。

TCG

QEMU 和 libvirt 包含使用 QEMU Tiny Code Generator(TCG)的动态转换模式。这个模式不需要硬件虚拟化支持。但是，红帽不支持 TCG。

通过检查其 XML 配置可识别基于 TCG 的客户机，例如使用 **virsh dumpxml** 命令。

- TCG 客户端的配置文件包括以下行：

```
<domain type='qemu'>
```

- KVM 客户端的配置文件包含以下行：

```
<domain type='kvm'>
```

SR-IOV InfiniBand 网络设备

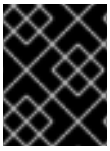
不支持使用单根 I/O 虚拟化 (SR-IOV) 将 InfiniBand 网络设备附加到虚拟机。

其他资源

- RHEL 9 虚拟化中的推荐功能
- RHEL 9 虚拟化中的资源分配限制

24.4. RHEL 9 虚拟化中的资源分配限制

以下限制适用于可分配给 Red Hat Enterprise Linux 9 (RHEL 9) 主机上单个 KVM 虚拟机(VM)的虚拟化资源。



重要

这些限制不适用于红帽提供的其他虚拟化解决方案，如 OpenShift Virtualization 或 Red Hat OpenStack Platform (RHOSP)。

每个虚拟机的最大 vCPU

有关在 RHEL 9 主机上运行的单个虚拟机上支持的最大 vCPU 数和内存量，请参阅：带有 KVM 的 Red Hat Enterprise Linux 的虚拟化限制

每个虚拟机的 PCI 设备

RHEL 9 支持每个虚拟机总线 **32** 个 PCI 设备插槽，每个设备插槽支持 **8** 个 PCI 功能。当虚拟机中启用了多功能且没有使用 PCI 桥接时，每个总线最多可以提供 256 个 PCI 功能。

每个 PCI 网桥都添加了一个新的总线，可能会启用其它 256 设备地址。但是，对于一些总线，用户不能使用所有 256 个设备地址，例如： root 总线有几个内置设备占用的插槽。

虚拟 IDE 设备

KVM 限制为每个虚拟机最多 **4** 个虚拟化 IDE 设备。

24.5. IBM Z 上的虚拟化与 AMD64 和 INTEL 64 有什么不同

IBM Z 系统上的 RHEL 9 中的 KVM 虚拟化与 AMD64 和 Intel 64 系统的 KVM 不同，如下所示：

PCI 和 USB 设备

IBM Z 不支持虚拟 PCI 和 USB 设备。这意味着不支持 **virtio-*-pci** 设备，应该改为使用 **virtio-*-ccw** 设备。例如，使用 **virtio-net-ccw** 而不是 **virtio-net-pci**。
请注意，支持直接附加 PCI 设备（也称 PCI 透传）。

支持的客户端操作系统

如果红帽只使用 RHEL 7、8 或 9 作为客户机操作系统，红帽只支持在 IBM Z 上托管的虚拟机。

设备引导顺序

IBM Z 不支持 **<boot dev='*device*'>** XML 配置元素。要定义设备引导顺序，请使用 XML 的 **<devices>** 部分中的 **<boot order='*number*'>** 元素。
另外，您可以在 **<boot>** 元素中使用特定于架构的 **loadparm** 属性来选择所需的引导条目。例如，以下命令决定了在引导序列中应首先使用磁盘，如果该磁盘上有 Linux 发行版，它将选择第二个引导条目：

```
<disk type='file' device='disk'>
  <driver name='qemu' type='qcow2'/>
  <source file='/path/to/qcow2'/>
  <target dev='vda' bus='virtio'/>
  <address type='ccw' cssid='0xfe' ssid='0x0' devno='0x0000'/>
  <boot order='1' loadparm='2'/>
</disk>
```

注意

在 AMD64 和 Intel 64 主机上，首选使用用于引导顺序管理的 **<boot order='*number*'>**。

内存热插拔

在 IBM Z 上无法将内存添加到正在运行的虚拟机。请注意，在 IBM Z 上，以及 AMD64 和 Intel64 上，从正在运行的虚拟机（*内存热插拔*）中不可能删除内存。

NUMA 拓扑

IBM Z 上的 **libvirt** 不支持 CPU 的非统一内存访问(NUMA)拓扑。因此，在这些系统上无法使用 NUMA 调优 vCPU 性能。

GPU 设备

IBM Z 系统上不支持 分配 GPU 设备。

vfio-ap

IBM Z 主机上的虚拟机可以使用 *vfio-ap* 加密设备 passthrough，这在其它构架上都不支持。

vfio-ccw

IBM Z 主机上的虚拟机可以使用 *vfio-ccw* 磁盘设备透传，这在其它构架中都不支持。

SMBIOOS

IBM Z 不提供 SMBIOS 配置。

watchdog 设备

如果在 IBM Z 主机上的虚拟机中使用看门狗设备，请使用 **diag288** 模型。例如：

```
<devices>
  <watchdog model='diag288' action='poweroff'/>
</devices>
```

kvm-clock

kvm-clock 服务特定于 AMD64 和 Intel 64 系统，不需要为 IBM Z 上的虚拟机时间管理配置。

v2v 和 p2v

virt-v2v 和 **virt-p2v** 工具只在 AMD64 和 Intel 64 构架中被支持，且没有在 IBM Z 中提供。

迁移

要成功迁移到以后的主机型号（例如从 IBM z14 到 z15），或更新 hypervisor，请使用 **host-model** CPU 模式。不建议使用 **host-passthrough** 和 **maximum** CPU 模式，因为它们通常不适合迁移。如果要在 **custom** CPU 模式中指定明确的 CPU 模式，请按照以下步骤操作：

- 不要使用以 **-base** 结尾的 CPU 模式。
- 不要使用 **qemu**、**max** 或 **host** CPU 模式。

要成功迁移到较旧的主机型号（如从 z15 到 z14），或者迁移到早期版本的 QEMU、KVM 或 RHEL 内核，请使用结尾没有 **-base** 的最老的可用的主机型号的 CPU 类型。

- 如果您的源主机和目标主机正在运行，您可以在目标主机上使用 **virsh hypervisor-cpu-baseline** 命令来获取合适的 CPU 模式。详情请参阅 验证虚拟机迁移的主机 CPU 兼容性。
- 有关 RHEL 9 中支持的机器类型的更多信息，请参阅 RHEL 9 虚拟化中推荐的功能。

PXE 安装和引导

当使用 PXE 在 IBM Z 上运行虚拟机时，**pxelinux.cfg/default** 文件需要一个特定的配置。例如：

```
# pxelinux
default linux
label linux
```

```
kernel kernel.img
initrd initrd.img
append ip=dhcp inst.repo=example.com/redhat/BaseOS/s390x/os/
```

安全执行

您可以通过在虚拟机的 XML 配置中定义 **<launchSecurity type="s390-pv"/>** 来使用准备好的安全客户虚拟机引导虚拟机。这会加密虚拟机的内存，以防止其免受 hypervisor 的不需要的访问。

请注意，在安全执行模式下运行虚拟机时不支持以下功能：

- 使用 **vfio** 的设备 passthrough
- 使用 **virsh domstats** 和 **virsh memstat** 获取内存信息
- **memballoon** 和 **virtio-rng** 虚拟设备
- 使用巨页支持的内存
- 实时和非实时虚拟机迁移
- 保存和恢复虚拟机
- 虚拟机快照，包括内存快照（使用 **--memspec** 选项）
- 完整的内存转储。反之，为 **virsh dump** 命令指定 **--memory-only** 选项。
- 248 个或更多的 vCPU。安全客户机的 vCPU 限制为 247 个。

其他资源

- 跨构架支持的虚拟化功能概述

24.6. ARM 64 上的虚拟化与 AMD64 和 INTEL 64 上的虚拟化有何不同

ARM 64 系统上 RHEL 9 中的 KVM 虚拟化（也称为 AArch64）与 AMD64 和 Intel 64 系统上的 KVM 在许多方面不同。这包括但不限于：

客户机操作系统

ARM 64 虚拟机(VM)中唯一支持的客户机操作系统是 RHEL 9。

Web 控制台管理

RHEL 9 web 控制台中虚拟机管理 的某些功能在 ARM 64 硬件上无法正常工作。

vCPU 热插和热拔

在 ARM 64 主机上不支持将虚拟 CPU (vCPU)附加到正在运行的虚拟机（也称为 vCPU 热插）。另外，与 AMD64 和 Intel 64 主机类似，ARM 64 上不支持从正在运行的虚拟机中移除 vCPU (vCPU 热拔)。

SecureBoot

SecureBoot 功能在 ARM 64 系统上不提供。

Migration

目前不支持在 ARM 64 主机之间迁移虚拟机。

Memory page sizes

ARM 64 目前仅支持运行具有 64 KB 内存页大小的虚拟机，且仅在具有 64 KB 内存页大小的主机上。目前在主机或客户机中不支持 4 KB 页大小。
要在 ARM 64 上成功创建虚拟机，您的主机必须 使用 64 KB 内存页大小的内核 ，并且子创建虚拟机时，您必须使用 **kernel-64k 软件包** 安装它，例如通过在 kickstart 文件中包含以下参数：

```
%packages
-kernel
kernel-64k
%end
```

巨页

具有 64 KB 内存页大小的 ARM 64 主机支持具有以下大小的大内存页：

- 2 MB
- 512 MB
- 16 GB
  当您在 ARM 64 主机上使用透明巨页(THP)文件系统时，它只支持 512 MB 巨页。

SVE

ARM 64 架构提供 Scalable Vector Expansion (SVE) 功能。如果主机支持该功能，在虚拟机中使用 SVE 可以在这些虚拟机中提高向量数计算和字符串操作的速度。
在支持它的主机 CPU 上默认启用了 SVE 的基线级别。但是，红帽建议明确配置每个向量长度。这确保虚拟机只能在兼容的主机上启动。要做到这一点：

1. 验证您的 CPU 是否有 SVE 功能：

   ```
   # grep -m 1 Features /proc/cpuinfo | grep -w sve

   Features: fp asimd evtstrm aes pmull sha1 sha2 crc32 atomics fphp asimdhp cpuid asimdrdm fcma dcpop sve
   ```

   如果这个命令的输出中包含 **sve** 或其退出代码是 0，则您的 CPU 支持 SVE。

2. 打开您要修改的虚拟机的 XML 配置：

   ```
   # virsh edit vm-name
   ```

3. 编辑 **<cpu>** 元素，类似如下：

   ```
   <cpu mode='host-passthrough' check='none'>
   <feature policy='require' name='sve'/>
   <feature policy='require' name='sve128'/>
   <feature policy='require' name='sve256'/>
   <feature policy='disable' name='sve384'/>
   <feature policy='require' name='sve512'/>
   </cpu>
   ```

   这个示例明确启用了 SVE 向量长度 128、256 和 512，并明确禁用了向量长度 384。

CPU 型号

ARM 64 上的虚拟机目前仅支持 **host** CPU 型号。

保存和恢复虚拟机

目前在 ARM 64 主机上不支持 保存和恢复 虚拟机。

PXE

在预引导执行环境(PXE)功能中引导，但不支持，红帽强烈建议不要在生产环境中使用它。
如果您需要 PXE 引导，则只能使用 **virtio-net-pci** 网络接口控制器(NIC)。另外，需要使用虚拟机 UEFI 平台固件（带有 **edk2-aarch64** 软件包安装的）内置 **VirtioNetDxe** 驱动程序进行 PXE 引导。请注意，不支持 iPXE 选项 ROM。

设备内存

设备内存功能，如双列直插式内存模块(DIMM)和非易失性 DIMM (NVDIMM)在 ARM 64 上无法工作。

pvpanic

ARM 64 上目前不支持 pvpanic 设备。确保从 ARM 64 上客户机 XML 配置的 **<devices>** 部分中删除 **<panic>** 元素，因为它的存在可能导致虚拟机无法引导。

OVMF

ARM 64 主机上的虚拟机无法使用 AMD64 和 Intel 64 上使用的 OVMF UEFI 固件，其包含在 **edk2-ovmf** 软件包中。相反，这些虚拟机使用 **edk2-aarch64** 软件包中包含的 UEFI 固件，它提供类似的接口并实施类似的功能集。
具体来说，**edk2-aarch64** 提供了一个内置的 UEFI shell，但不支持以下功能：

- SecureBoot
- 管理模式

kvm-clock

不需要为 ARM 64 上虚拟机中的时间管理配置 **kvm-clock** 服务。

外设设备

ARM 64 系统支持与 AMD64 和 Intel 64 设备部分不同的外围设备的集合。

- 仅支持 PCIe 拓扑。
- ARM 64 系统通过使用 **virtio-*-pci** 驱动程序来支持 **virtio** 设备。另外，不支持 **virtio-iommu** 和 **virtio-input** 设备。
- **virtiofs** 功能仅作为技术预览提供，因此不被支持。
- **virtio-gpu** 驱动程序只支持图形安装。
- ARM 64 系统只支持 **usb-mouse** 和 **usb-tablet** 设备用于图形安装。不支持其他 USB 设备、USB passthrough 或 USB 重定向。
- 使用虚拟功能 I/O (VFIO)的设备分配只支持 NIC（物理和虚拟功能）。

模拟设备

ARM 64 上不支持以下设备：

- 模拟声音设备，如 ICH9、ICH6 或 AC97。
- 模拟图形卡，如 VGA 卡。
- 模拟网络设备，如 **rtl8139**。

GPU 设备

ARM 64 系统上不支持 分配 GPU 设备。

串行控制台配置

当在虚拟机上设置串行控制台时，请使用 **console=ttyAMA0** 内核选项，而不是使用 **grubby** 实用程序的 **console=ttyS0**。

不可屏蔽中断

目前无法向 ARM 64 虚拟机发送不可屏蔽中断(NMI)。

嵌套虚拟化

目前无法在 ARM 64 主机上创建嵌套虚拟机。

v2v 和 p2v

virt-v2v 和 **virt-p2v** 工具只在 AMD64 和 Intel 64 构架上被支持，因此在 ARM 64 上不提供。

24.7. RHEL 9 中支持虚拟化功能概述

下表提供有关在可用系统架构的 RHEL 9 中所选虚拟化功能支持状态的信息。

表 24.1. 常规支持

Intel 64 和 AMD64	IBM Z	ARM 64
支持	支持	支持

表 24.2. 设备热插和热拔

	Intel 64 和 AMD64	IBM Z	ARM 64
CPU 热插	支持	支持	*不支持*
CPU 热拔	*不支持*	*不支持*	*不支持*
内存热插拔	支持	*不支持*	*不支持*
内存热拔	*不支持*	*不支持*	*不支持*
外设设备热插	支持	支持 [a]	支持
外设设备热拔	支持	支持 [b]	支持

[a] 需要使用 **virtio-*-ccw** 设备，而不是 **virtio-*-pci**

[b] 需要使用 **virtio-*-ccw** 设备，而不是 **virtio-*-pci**

表 24.3. 其他选择的功能

	Intel 64 和 AMD64	IBM Z	ARM 64
NUMA 调整	支持	*不支持*	支持
SR-IOV 设备	支持	*不支持*	支持
virt-v2v 和 p2v	支持	*不支持*	*不可用*

请注意，有些不支持的特性在 Red Hat Virtualization 和 Red Hat OpenStack platform 等其他红帽产品上支持。如需更多信息，请参阅 RHEL 9 虚拟化中的不支持的功能。

其他资源

- RHEL 9 虚拟化不支持的功能