Red Hat Enterprise Linux 7

虚拟化部署和管理指南

在 RHEL 物理机器上安装、配置和管理虚拟机

Last Updated: 2023-10-19

Red Hat Enterprise Linux 7 虚拟化部署和管理指南

在 RHEL 物理机器上安装、配置和管理虚拟机

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

摘要

本指南介绍了如何配置 Red Hat Enterprise Linux 7 机器来作为虚拟化主机系统，以及如何使用 KVM hypervisor 来安装和配置客户机虚拟机。其他主题包括 PCI 设备配置、SR-IOV、网络、存储、设备和客户机虚拟机管理，以及故障排除、兼容性和限制。需要在客户机虚拟机上运行的流程会被明确标记为这样。 本指南中描述的所有流程都应该在 AMD64 或 Intel 64 主机机器上执行，除非另有说明。有关在 AMD64 和 Intel 64 以外的构架上使用 Red Hat Enterprise Linux 7 虚拟化说明，请参考 。 有关红帽提供的虚拟化解决方案的更通用的介绍，请参阅 Red Hat Enterprise Linux 7 虚拟化入门指南。

目录

部分 I. DEPLOYMENT

这部分提供了有关如何安装和配置 Red Hat Enterprise Linux 7 机器来作为虚拟化主机系统的说明，以及如何使用 KVM hypervisor 来安装和配置客户机虚拟机。

第 1 章 系统要求

Intel 64 和 AMD64 构架上 Red Hat Enterprise Linux 7 的 KVM hypervisor 提供虚拟化。本章列出了运行虚拟机（也称为 VM）的系统要求。

有关安装虚拟化软件包的详情，请参考 第 2 章 *安装虚拟化软件包*。

1.1. 主机系统要求

最低主机系统要求

- 6 GB 空闲磁盘空间。
- 2 GB RAM。

推荐的系统要求

- 每个虚拟化 CPU 和主机各有一个核或线程。
- 2 GB RAM，外加虚拟机的额外 RAM。
- 主机有 6 GB 的磁盘空间，外加虚拟机所需的磁盘空间。

  大多数客户机操作系统需要至少 6 GB 的磁盘空间。每个客户机的额外存储空间，这取决于它们的工作负载。

  交换空间

  当物理内存(RAM)已满时，将使用 Linux 中的交换空间。如果系统需要更多的内存资源并且 RAM 已满，内存中的不活动页面将移到交换空间。虽然交换空间可以帮助具有少量 RAM 的计算机，但不应将其视为更多 RAM 的替代品。交换空间位于硬盘驱动器上，其访问时间比物理内存要慢。交换分区的大小可以通过主机的物理 RAM 计算。红帽客户门户网站包含有关安全、有效地确定交换分区大小的文章：https://access.redhat.com/site/solutions/15244.

  - 当使用原始镜像文件时，所需磁盘空间总量等于或大于镜像文件所需的空间、主机操作系统所需的 6 GB 空间以及客户机的交换空间的总和。

    公式 1.1. 计算使用原始镜像的客户机虚拟机所需的空间

    total for raw format = images + hostspace + swap

    对于 qcow 镜像，您也必须计算客户机期望的最大存储要求 **（qcow 格式的总量）**，因为 qcow 和 qcow2 镜像可以根据需要增加。要允许此扩展，首先将客户机期望的最大存储要求 **（期望的最大客户机存储）** 乘以 1.01，再加上主机 **（host）** 和必要的交换空间 **（swap）** 所需的空间。

    公式 1.2. 计算使用 qcow 镜像的客户机虚拟机所需的空间

    total for qcow format = (expected maximum guest storage * 1.01) + host + swap

第 7 章 *使用 KVM 进行过量使用* 中进一步概述了客户机虚拟机的要求。

1.2. KVM HYPERVISOR 要求

KVM Hypervisor 要求：

- 一个具有适用于基于 x86 系统的 Intel VT-x 和 Intel 64 虚拟化扩展的 Intel 处理器；或者
- 一个具有 AMD-V 和 AMD64 虚拟化扩展的 AMD 处理器。

完全虚拟化需要虚拟化扩展（Intel VT-x 或 AMD-V）。输入以下命令来确定您的系统是否有硬件虚拟化扩展，以及它们是否已启用。

过程 1.1. 验证虚拟化扩展

1. **验证 CPU 虚拟化扩展是否可用**
   输入以下命令来验证 CPU 虚拟化扩展是否可用：

   ```
   $ grep -E 'svm|vmx' /proc/cpuinfo
   ```

2. **分析输出**

   - 以下示例输出包含了一个 **vmx** 条目，表示一个具有 Intel VT-x 扩展的 Intel 处理器：

     ```
     flags   : fpu tsc msr pae mce cx8 vmx apic mtrr mca cmov pat pse36 clflush
     dts acpi mmx fxsr sse sse2 ss ht tm syscall lm constant_tsc pni monitor ds_cpl
     vmx est tm2 cx16 xtpr lahf_lm
     ```

   - 以下示例输出包含了一个 **svm** 条目，表示一个具有 AMD-V 扩展的 AMD 处理器：

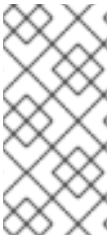
     ```
     flags   :  fpu tsc msr pae mce cx8 apic mtrr mca cmov pat pse36 clflush
     mmx fxsr sse sse2 ht syscall nx mmxext svm fxsr_opt lm 3dnowext 3dnow pni cx16
     lahf_lm cmp_legacy svm cr8legacy ts fid vid ttp tm stc
     ```

   如果 **grep -E 'svm|vmx' /proc/cpuinfo** 命令返回任何输出，则处理器包含硬件虚拟化扩展。在某些情况下，制造商在 BIOS 中禁用了虚拟化扩展。如果没有显示扩展，或者完整虚拟化无法正常工作，请参阅 过程 A.3, “在 BIOS 中启用虚拟化扩展”有关在 BIOS 配置工具中启用扩展的说明。

3. **确定 KVM 内核模块是否已加载**
   另外，还可以使用以下命令验证 **kvm** 模块是否已被加载到内核中：

   ```
   # lsmod | grep kvm
   ```

   如果输出包括 **kvm_intel** 或 **kvm_amd**，则 **kvm** 硬件虚拟化模块已加载。



注意

virsh 工具（由 libvirt-client 软件包提供）可通过以下命令输出系统虚拟化功能的完整列表：

```
# virsh capabilities
```

1.3. KVM 客户机虚拟机兼容性

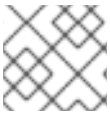
Red Hat Enterprise Linux 7 服务器有一定的支持限制。

以下 URL 解释了 Red Hat Enterprise Linux 的处理器和内存量限制：

- 对于主机系统：https://access.redhat.com/articles/rhel-limits
- 对于 KVM hypervisor：https://access.redhat.com/articles/rhel-kvm-limits

以下 URL 列出了认证的、可以在 Red Hat Enterprise Linux KVM 主机上运行的客户机操作系统：

- https://access.redhat.com/articles/973133



注意

有关 KVM hypervisor 的限制和支持限制的其它信息，请参考 附录 C, *虚拟化限制*。

1.4. 支持的客户机 CPU 型号

每个 hypervisor 对 CPU 功能都有自己的策略，客户机默认都会看到。hypervisor 呈现给客户机的 CPU 功能集取决于客户机虚拟机配置中选择的 CPU 型号。

1.4.1. 列出客户机 CPU 型号

要查看一个架构类型所支持的 CPU 型号的完整列表，请运行 **virsh cpu-models *architecture*** 命令。例如：

```
$ virsh cpu-models x86_64
486
pentium
pentium2
pentium3
pentiumpro
coreduo
n270
core2duo
qemu32
kvm32
cpu64-rhel5
cpu64-rhel6
kvm64
qemu64
Conroe
Penryn
Nehalem
Westmere
SandyBridge
Haswell
athlon
phenom
Opteron_G1
Opteron_G2
Opteron_G3
Opteron_G4
Opteron_G5
```

```
$ virsh cpu-models ppc64
POWER7
POWER7_v2.1
```

```
POWER7_v2.3
POWER7+_v2.1
POWER8_v1.0
```

支持的 CPU 型号和功能的完整列表包含在 **cpu_map.xml** 文件中，位于 **/usr/share/libvirt/** 中：

```
# cat /usr/share/libvirt/cpu_map.xml
```

客户机的 CPU 型号和功能可在域 XML 文件的 **<cpu>** 部分进行更改。如需更多信息，请参阅 第 23.12 节 “CPU 型号和拓扑”。

可以根据需要将主机型号配置为使用指定的功能集。如需更多信息，请参阅 第 23.12.1 节 “更改指定 CPU 的 Feature Set”。

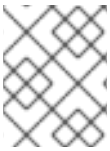
第 2 章 安装虚拟化软件包

要使用虚拟化，必须在您的计算机上安装红帽虚拟化软件包。虚拟化软件包可以在安装 Red Hat Enterprise Linux 时安装，或使用 **yum** 命令和订阅管理器应用程序来安装。

KVM hypervisor 使用默认的带有 **kvm** 内核模块的 Red Hat Enterprise Linux 内核。

2.1. 在 RED HAT ENTERPRISE LINUX 安装过程中安装虚拟化软件包

这部分提供了有关在安装 Red Hat Enterprise Linux 过程中安装虚拟化软件包的信息。



注意

有关安装 Red Hat Enterprise Linux 的详情，请查看 Red Hat Enterprise Linux 7 安装指南。

重要

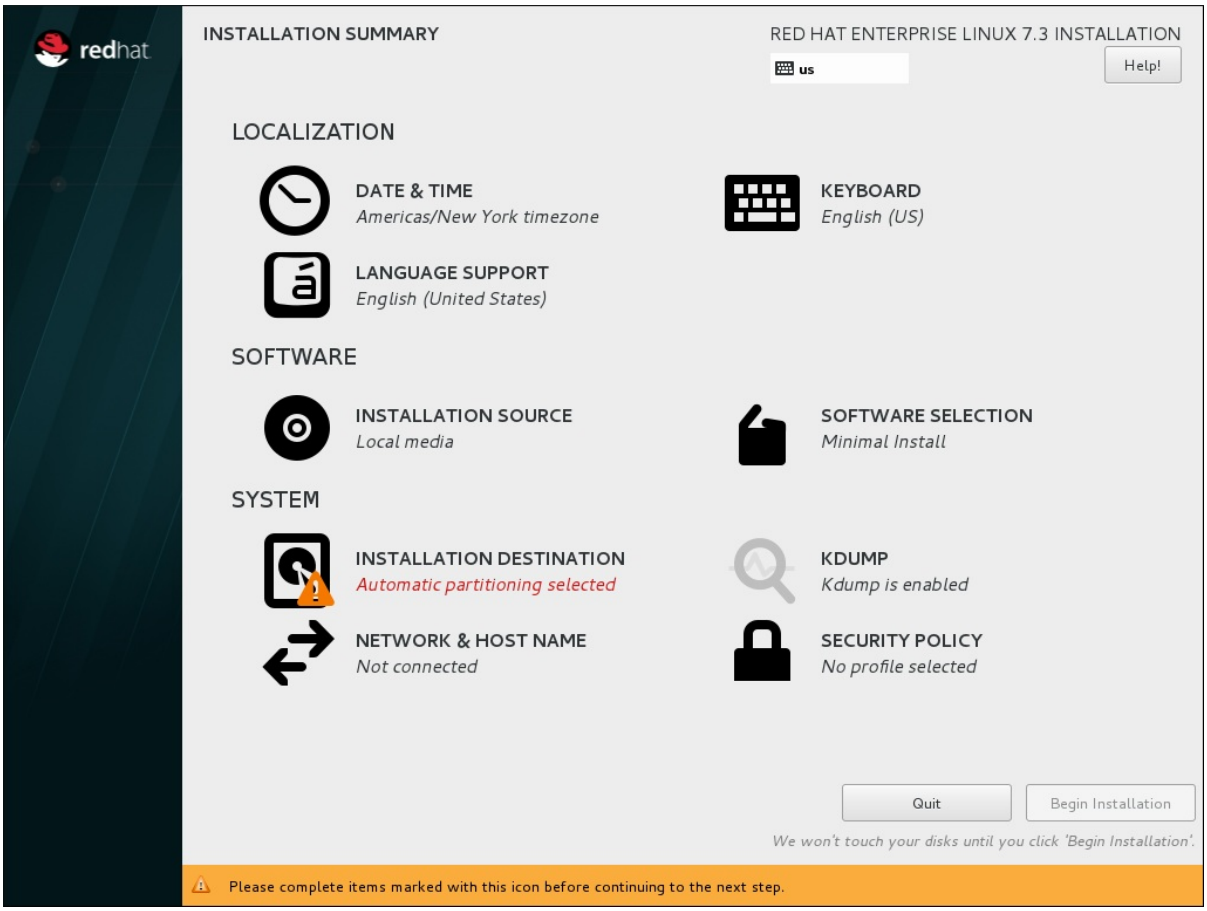
Anaconda 界面只提供在安装 Red Hat Enterprise Linux 服务器过程中安装红帽虚拟化软件包的选项。

安装 Red Hat Enterprise Linux 工作站时，只能在工作站安装完成后安装红帽虚拟化软件包。请查看 第 2.2 节 “在现有 Red Hat Enterprise Linux 系统上安装虚拟化软件包”

过程 2.1. 安装虚拟化软件包

1. **选择软件**
   按照安装流程操作，直到 **安装概述** 屏幕出现。

图 2.1. 安装概述屏幕



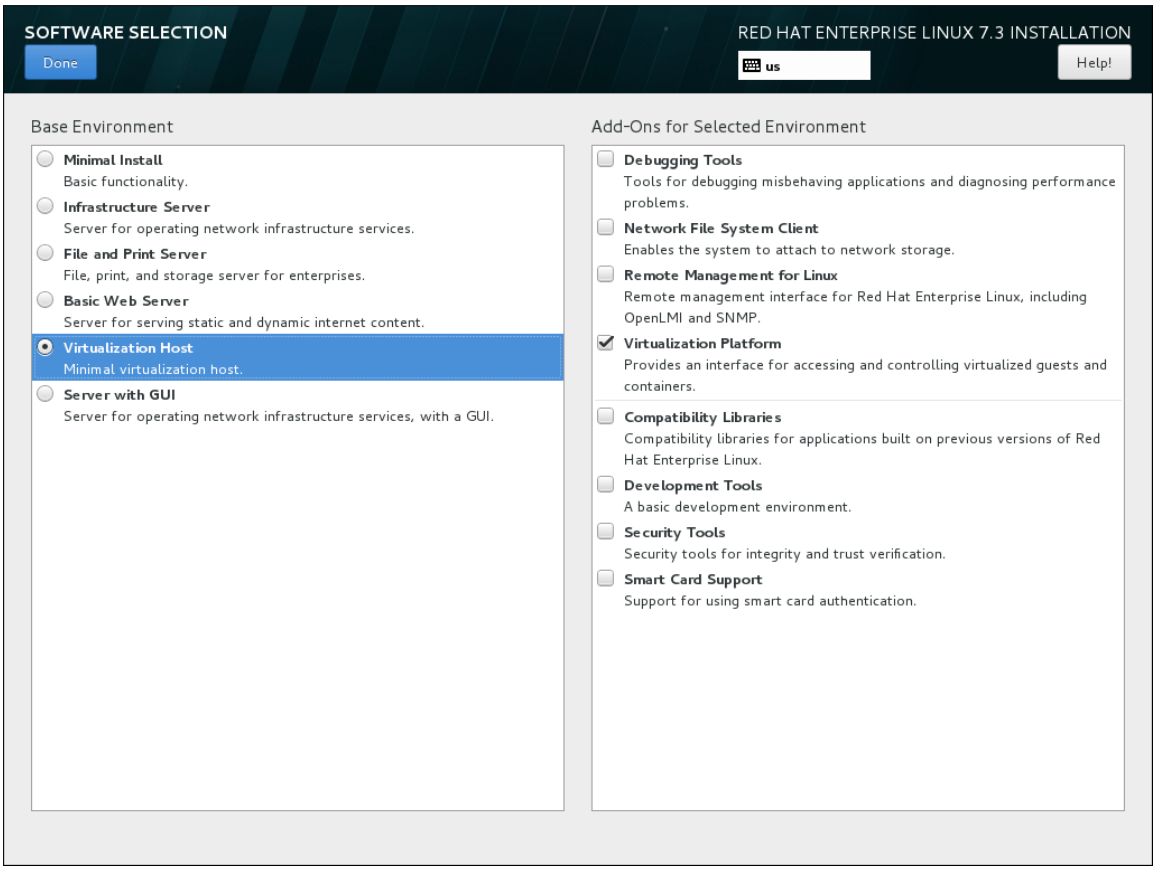
在 **安装概述** 屏幕中，点击 **软件选择**。此时会打开 **软件选择** 屏幕。

2. 选择**服务器类型和软件包**组
   您只能安装带有基本虚拟化软件包或带有允许通过图形用户界面管理客户机的软件包的 Red Hat Enterprise Linux 7。执行以下操作之一：

   - **安装最小虚拟化主机**

     在 **Base Environment** 窗格中选择 **Virtualization Host** 单选按钮，在**Add-Ons for Selected Environment** 窗格中选择 **Virtualization Platform** 复选框。这会安装一个基本的虚拟化环境，可以使用 **virsh** 或通过网络远程运行它。

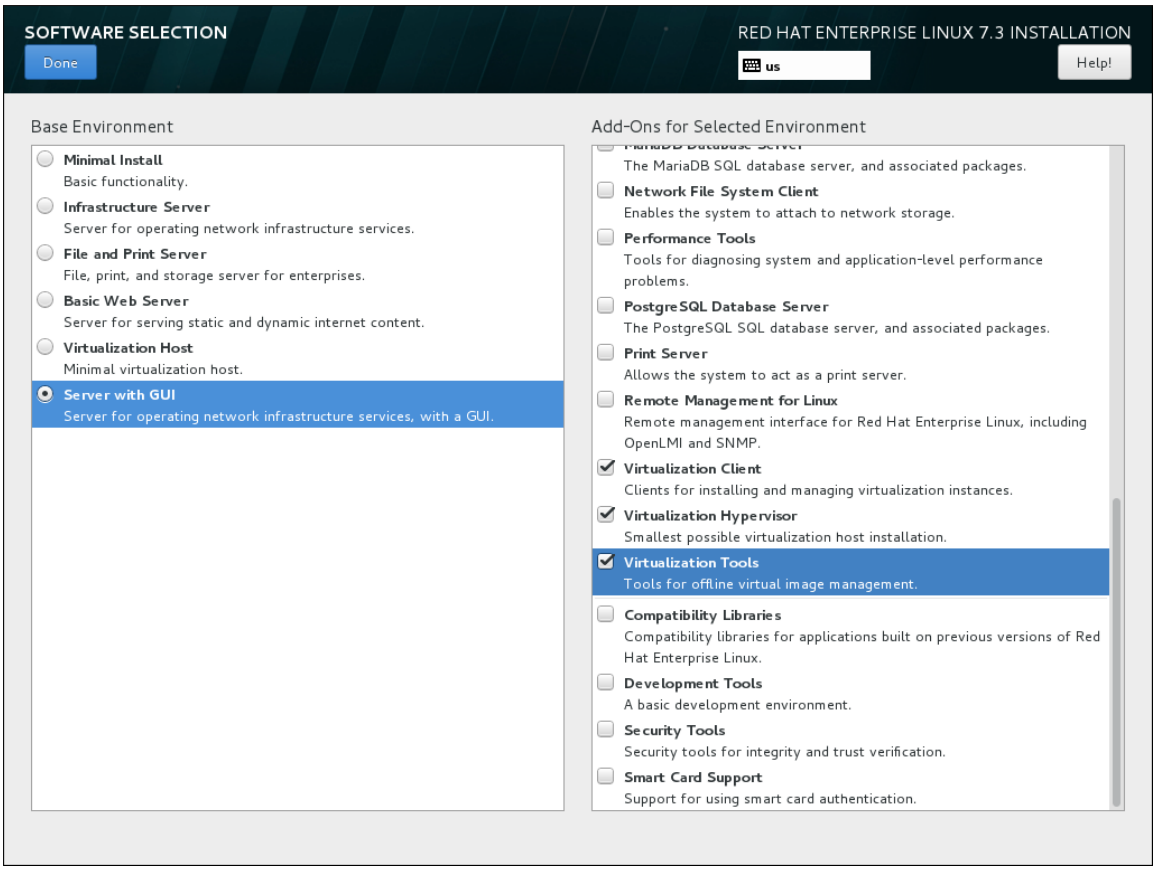
图 2.2. 软件选择屏幕中选择的虚拟化主机



- **使用图形用户界面安装虚拟化主机**

  在 **Base Environment** 窗格中选择 **Server with GUI** 单选按钮，在 **Add-Ons for Selected Environment**窗格中选择 **Virtualization Client**、**Virtualization Hypervisor** 和 **Virtualization Tools** 复选框。这会安装一个虚拟化环境，以及用于安装和管理客户机虚拟机的图形工具。

图 2.3. 在软件选择屏幕中选择带有 GUI 的服务器



3. **完成安装**
   点击 **Done** 并继续安装。

重要

您需要一个有效的 Red Hat Enterprise Linux 订阅才能接受虚拟化软件包的更新。

2.1.1. 使用 Kickstart 文件安装 KVM 软件包

要使用 Kickstart 文件来安装带有虚拟化软件包的 Red Hat Enterprise Linux，请在 Kickstart 文件的 **%packages** 部分中追加以下软件包组：

```
@virtualization-hypervisor
@virtualization-client
@virtualization-platform
@virtualization-tools
```

有关使用 Kickstart 文件安装的详情，请参考 Red Hat Enterprise Linux 7 安装指南 。

2.2. 在现有 RED HAT ENTERPRISE LINUX 系统上安装虚拟化软件包

这部分描述了在现有 Red Hat Enterprise Linux 7 系统上安装 KVM 管理程序的步骤。

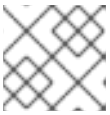
要安装软件包，您的计算机必须在红帽客户门户网站已进行了注册和订阅。要使用 Red Hat Subscription Manager 进行注册，请运行 **subscription-manager register** 命令，并按照提示进行操作。或者，从桌面上的 **Applications → System Tools** 运行 Red Hat Subscription Manager 应用程序来注册。

如果您没有有效的红帽订阅，请访问 红帽在线商店 来获取一个红帽订阅。有关在红帽客户门户网站注册和订阅系统的更多信息，请参阅 https://access.redhat.com/solutions/253273。

2.2.1. 手动安装虚拟化软件包

要在 Red Hat Enterprise Linux 上使用虚拟化，您至少需要安装以下软件包：

- qemu-kvm ：这个软件包提供了用户级的 KVM 模拟器，方便了主机和客户机虚拟机之间的通信。
- qemu-img ：这个软件包为客户机虚拟机提供磁盘管理。



注意

qemu-img 软件包是作为 qemu-kvm 软件包的依赖项安装的。

- libvirt ：此软件包提供用于与 hypervisor 和主机系统进行交互的服务器和主机端的库，以及用于处理库调用、管理虚拟机和控制 hypervisor 的 **libvirtd** 守护进程。

要安装这些软件包，请输入以下命令：

```
# yum install qemu-kvm libvirt
```

还提供了几个额外的虚拟化管理软件包，在使用虚拟化时推荐使用：

- virt-install:这个软件包提供 **virt-install** 命令，可用于从命令行创建虚拟机。
- libvirt-python:这个软件包包含一个模块，它允许使用 Python 编程语言编写的应用程序可以使用 libvirt API 提供的接口。
- virt-manager ：这个软件包提供了 **virt-manager** 工具，也称为 **虚拟机管理器**。这是用于管理虚拟机的图形化工具。它使用 libvirt-client 库作为管理 API。
- libvirt-client:这个软件包提供了用于访问 libvirt 服务器的客户端 API 和库。libvirt-client 软件包包括 **virsh** 命令行工具，用于从命令行或特殊的虚拟化 shell 来管理和控制虚拟机及 hypervisor 。

您可以使用以下命令安装所有这些推荐的虚拟化软件包：

```
# yum install virt-install libvirt-python virt-manager virt-install libvirt-client
```

2.2.2. 安装虚拟化软件包组

虚拟化软件包也可以从软件包组来安装。您可以通过运行 **yum grouplist hidden** 命令来查看可用的组的列表。

在可用的包组的完整列表中，下表描述了虚拟化软件包组及其所提供的内容。

表 2.1. 虚拟化软件包组

软件包组	描述	强制的软件包	可选的软件包
虚拟化 Hypervisor	最小的虚拟化主机安装	libvirt、qemu-kvm、qemu-img	qemu-kvm-tools

软件包组	描述	强制的软件包	可选的软件包
虚拟化客户端	用于安装和管理虚拟化实例的客户端	gnome-boxes, virt-install, virt-manager, virt-viewer, qemu-img	virt-top, libguestfs-tools, libguestfs-tools-c
虚拟化平台	提供用于访问和控制虚拟机和容器的接口	libvirt、libvirt-client、virt-who、qemu-img	fence-virtd-libvirt, fence-virtd-multicast, fence-virtd-serial, libvirt-cim, libvirt-java, libvirt-snmp, perl-Sys-Virt
虚拟化工具	用于离线虚拟镜像管理的工具	libguestfs, qemu-img	libguestfs-java, libguestfs-tools, libguestfs-tools-c

要安装软件包组，请运行 **yum group install *package_group*** 命令。例如，要安装带有所有软件包类型的 **Virtualization Tools** 软件包组，请运行：

```
# yum group install "Virtualization Tools" --setopt=group_package_types=mandatory,default,optional
```

有关安装软件包组的详情，请参考 *如何在 Red Hat Enterprise Linux 上使用 yum 安装一组软件包？* 知识库文章。

第 3 章 创建虚拟机

在 Red Hat Enterprise Linux 7 主机系统上安装了 虚拟化软件包 后，您可以使用 **virt-manager** 界面创建虚拟机并安装客户机操作系统。或者，您可以使用 **virt-install** 命令行工具和一个参数列表或使用脚本。本章将涵盖这两种方法。

3.1. 客户机虚拟机部署注意事项

创建任何客户机虚拟机之前应考虑各种因素。部署前应评估虚拟机的角色，但也应根据可变因素（负载，客户端的数量）来定期监控和评估。因素包括：

性能

应根据预期的任务来部署和配置客户机虚拟机。有些客户端系统（例如，运行数据库服务器的客户机）可能需要特殊的性能考虑。可能需要根据客户机的角色和预计的系统负载来给它们分配更多的 CPU 或内存。

输入/输出要求和输入/输出类型

某些客户机虚拟机可能有特别的高 I/O 要求，或者可能根据 I/O 类型（例如，典型的磁盘块大小访问或客户端的数量）需要做进一步的考虑或预测。

存储

有些客户虚拟机可能需要更高的优先级来访问存储或快速的磁盘类型，或者可能需要对存储区域进行独占访问。在部署和维护存储时，还应定期监控和考虑虚拟机所使用的存储量。请务必阅读 Red Hat Enterprise Linux 7 虚拟化安全指南 中概述的所有注意事项。另请务必了解您的物理存储可能会限制您的虚拟存储中的选项。

网络和网络基础架构

根据您的环境，某些客户机虚拟机可能需要比其他客户机更快的网络连接。在部署和维护客户机时，带宽或延迟通常是要考虑的因素，特别是在需求或负载发生变化时。

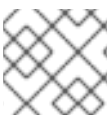
请求要求

如果整个磁盘都支持 virtio 驱动器，则只能对 virtio 驱动器上的客户机虚拟机发出 SCSI 请求，磁盘设备参数在 域 XML 文件 中被设为 **lun**，如下例所示：

```
<devices>
  <emulator>/usr/libexec/qemu-kvm</emulator>
  <disk type='block' device='lun'>
```

3.2. 使用 VIRT-INSTALL 创建客户机

您可以使用 **virt-install** 命令来创建虚拟机，并通过命令行在这些虚拟机上安装操作系统。virt-install 可以交互式使用，也可以作为脚本的一部分来自动创建虚拟机。如果您使用的是交互式图形安装，则必须在运行 **virt-install** 之前安装 **virt-viewer**。另外，您可以使用 **virt-install** 和 kickstart 文件启动虚拟机操作系统的无人值守安装。



注意

您可能需要 root 特权才能使某些 **virt-install** 命令成功完成。

virt-install 工具使用很多命令行选项。但是，大多数 **virt-install** 选项都不需要。

虚拟客户机安装的主要所需选项为：

--name

虚拟机的名称。

--memory

分配给客户机的内存量(RAM)，以 MiB 为单位。

客户机存储

使用以下客户机存储选项之一：

- **--disk**

  虚拟机的存储配置详情。如果您使用 **--disk none** 选项，则创建的虚拟机没有磁盘空间。

- **--filesystem**

  虚拟客户机的文件系统的路径。

安装方法

使用以下安装方法之一：

- **--location**

  安装介质的位置。

- **--cdrom**

  用作虚拟 CD-ROM 设备的文件或设备。它可以是 ISO 镜像的路径，也可以是要从中获取或访问最小引导 ISO 镜像的 URL。但是，它不能是 物理主机的 CD-ROM 或 DVD-ROM 设备。

- **--pxe**

  使用 PXE 引导协议来加载初始 ramdisk 和内核以启动客户机安装过程。

- **--import**

  跳过操作系统安装过程，并围绕现有磁盘镜像构建客户机。用于引导的设备是 **disk** 或 **filesystem** 选项指定的第一个设备。

- **--boot**

  安装后虚拟机的引导配置。这个选项允许指定引导设备的顺序，使用可选的内核参数永久引导内核和 initrd，并启用 BIOS 引导菜单。

要查看选项的完整列表，请输入以下命令：

```
# virt-install --help
```

要查看某个选项的属性的完整列表，请输入以下命令：

```
# virt install --option=?
```

virt-install 手册页还记录了每个命令选项、重要的变量和示例。

运行 **virt-install** 之前，您可能还需要使用 **qemu-img** 来配置存储选项。有关使用 **qemu-img** 的说明，请参考 第 14 章 *使用 qemu-img*。

3.2.1. 从 ISO 镜像安装虚拟机

以下示例从 ISO 镜像安装虚拟机：

```
# virt-install \
  --name guest1-rhel7 \
  --memory 2048 \
  --vcpus 2 \
  --disk size=8 \
  --cdrom /path/to/rhel7.iso \
  --os-variant rhel7
```

--cdrom /path/to/rhel7.iso 选项指定虚拟机将在指定的位置从 CD 或 DVD 镜像安装。

3.2.2. 导入虚拟机镜像

以下示例从虚拟磁盘镜像导入虚拟机：

```
# virt-install \
  --name guest1-rhel7 \
  --memory 2048 \
  --vcpus 2 \
  --disk /path/to/imported/disk.qcow \
  --import \
  --os-variant rhel7
```

--import 选项指定虚拟机将从 **--disk /path/to/imported/disk.qcow** 选项指定的虚拟磁盘镜像导入。

3.2.3. 从网络安装虚拟机

以下示例从网络位置安装虚拟机：

```
# virt-install \
  --name guest1-rhel7 \
  --memory 2048 \
  --vcpus 2 \
  --disk size=8 \
  --location http://example.com/path/to/os \
  --os-variant rhel7
```

location http://example.com/path/to/os 选项指定安装树位于指定的网络位置。

3.2.4. 使用 PXE 安装虚拟机

当使用 PXE 引导协议安装虚拟机时，指定桥接网络的**--network** 选项和 **--pxe** 选项必须指定。

以下示例使用 PXE 安装虚拟机：

```
# virt-install \
  --name guest1-rhel7 \
  --memory 2048 \
  --vcpus 2 \
  --disk size=8 \
  --network=bridge:br0 \
  --pxe \
  --os-variant rhel7
```

3.2.5. 使用 Kickstart 安装虚拟机

以下示例使用 kickstart 文件安装虚拟机：

```
# virt-install \
  --name guest1-rhel7 \
  --memory 2048 \
  --vcpus 2 \
  --disk size=8 \
  --location http://example.com/path/to/os \
  --os-variant rhel7 \
  --initrd-inject /path/to/ks.cfg \
  --extra-args="ks=file:/ks.cfg console=tty0 console=ttyS0,115200n8"
```

initrd-inject 和 **extra-args** 选项指定将要使用 Kickstarter 文件来安装虚拟机。

3.2.6. 在客户机创建过程中配置客户机虚拟机网络

在创建客户机虚拟机时，您可以为虚拟机指定和配置网络。本节提供了每种客户机虚拟机主网络类型的选项。

带有 NAT 的默认网络

默认网络使用 **libvirtd** 的网络地址转换(NAT)虚拟网络交换机。有关 NAT 的更多信息，请参阅 第 6.1 节 “使用 libvirt 进行网络地址转换(NAT)”。

在创建带有 NAT 的默认网络的虚拟客户机之前，请确定安装了 libvirt-daemon-config-network 软件包。

要为客户端虚拟机配置 NAT 网络，请对 **virt-install** 使用这个选项：

```
--network default
```

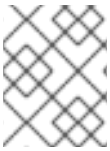
注意

如果没有指定 **network** 选项，将使用带有 NAT 的默认网络配置客户机虚拟机。

带有 DHCP 的桥接网络

当配置带有 DHCP 的桥接网络时，客户机使用外部的 DHCP 服务器。如果主机有静态网络配置，并且客户机需要与局域网(LAN)的完全入站和出站连接，则应使用这个选项。如果需要对客户机虚拟机执行实时迁移，则应使用它。要为虚拟机配置带有 DHCP 的桥接网络，请使用以下选项：

```
--network br0
```



注意

在运行 **virt-install** 之前，必须单独创建网桥。有关创建网桥的详情，请参考 第 6.4.1 节 “在 Red Hat Enterprise Linux 7 主机上配置桥接网络”。

带有静态 IP 地址的桥接网络
桥接网络也可用于将客户机配置为使用静态 IP 地址。要为虚拟机配置带有静态 IP 地址的桥接网络，请使用以下选项：

```
--network br0 \
--extra-args "ip=192.168.1.2::192.168.1.1:255.255.255.0:test.example.com:eth0:none"
```

有关网络引导选项的更多信息，请参阅 Red Hat Enterprise Linux 7 安装指南 。

无网络
要配置没有网络接口的客户机虚拟机，请使用以下选项：

```
--network=none
```

3.3. 使用 VIRT-MANAGER 创建客户机

虚拟机管理器（也称为 **virt-manager** ）是创建和管理客户机虚拟机的图形化工具。

这部分论述了如何使用 **virt-manager** 在 Red Hat Enterprise Linux 7 主机上安装 Red Hat Enterprise Linux 7 虚拟机 。

这些流程假设 KVM hypervisor 和所有其他必要的软件包都已安装，并已为主机配置了虚拟化。有关安装虚拟化软件包的详情，请参考 第 2 章 *安装虚拟化软件包*。

3.3.1. virt-manager 安装概述

New VM 向导将虚拟机创建过程分为五个步骤：

1. 选择 hypervisor 和安装类型
2. 定位和配置安装介质
3. 配置内存和 CPU 选项
4. 配置虚拟机的存储
5. 配置虚拟机名称、网络、架构和其他硬件设置

在继续之前，请确保 **virt-manager** 可以访问安装介质（无论是本地还是通过网络）。

3.3.2. 使用 virt-manager 创建 Red Hat Enterprise Linux 7 客户机

这个步骤涵盖了使用本地存储的安装 DVD 或 DVD 镜像创建 Red Hat Enterprise Linux 7 客户机虚拟机。Red Hat Enterprise Linux 7 DVD 镜像可以通过 红帽客户门户网站 获得。

注意

如果要安装启用了 SecureBoot 的虚拟机，请参阅 使用 virt-manager 创建 SecureBoot Red Hat Enterprise Linux 7 客户机。

过程 3.1. 使用本地安装介质通过 virt-manager 创建 Red Hat Enterprise Linux 7 客户机虚拟机

1. **可选：准备**
   为虚拟机准备存储环境。有关准备存储的详情请参考 第 13 章 *为虚拟机管理存储*。

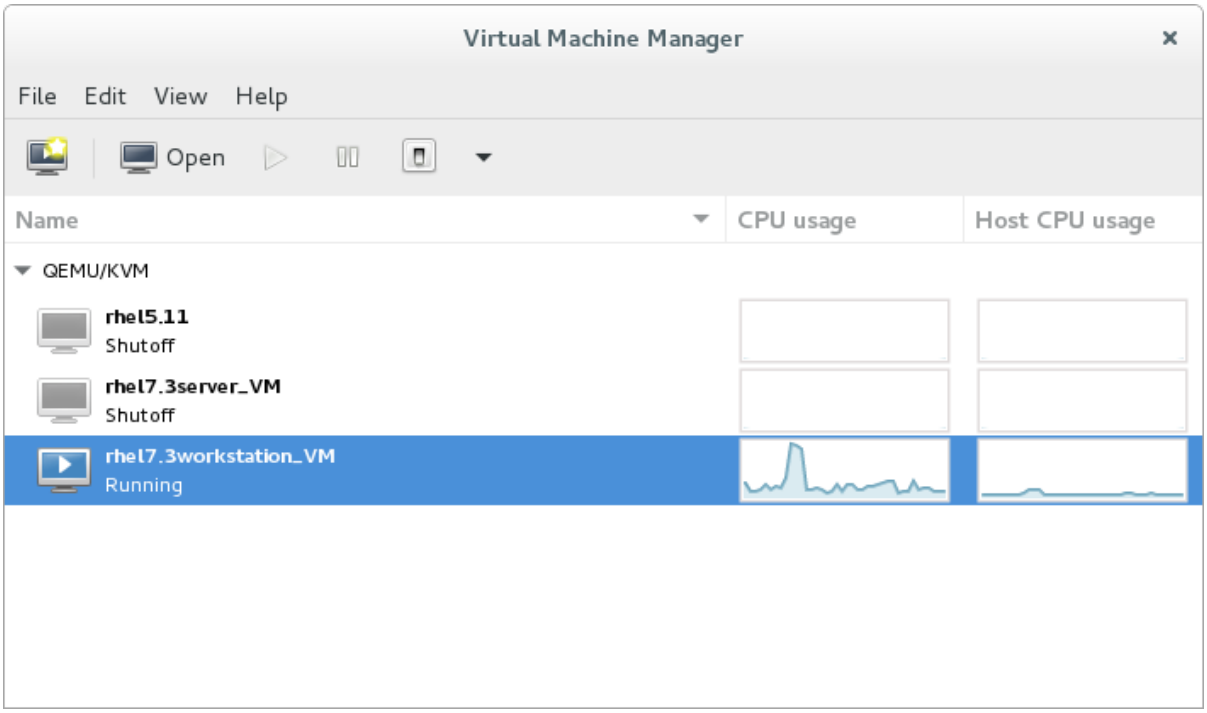
   重要

   各种存储类型可用于存储客户机虚拟机。但是，要使虚拟机能够使用迁移功能，虚拟机必须在网络存储上创建。

   Red Hat Enterprise Linux 7 至少需要 1 GB 存储空间。但是，红帽建议为 Red Hat Enterprise Linux 7 安装以及本指南中的步骤推荐至少 5 GB 存储空间。

2. **打开 virt-manager 并启动向导**
   以 root 用户身份执行 virt-manager 命令打开 **virt-manager**，或打开 **Applications → System Tools → Virtual Machine Manager**。

   图 3.1. Virtual Machine Manager 窗口

   （可选）通过选择管理程序并单击 **连接** 按钮打开远程管理程序。

   点击  启动新的虚拟化客户机向导。

   此时会打开 **New VM** 窗口。

3. **指定安装类型**
   选择安装类型：

   本地安装介质（ISO 镜像或 CDROM）

   此方法使用安装磁盘的镜像（如 **.iso**）。但是，无法使用 主机 CD-ROM 或者 DVD-ROM 设备。

   网络安装（HTTP、FTP 或 NFS）

此方法涉及使用已镜像的 Red Hat Enterprise Linux 或 Fedora 安装树来安装客户机。安装树必须可通过 HTTP、FTP 或 NFS 访问。

如果您选择 **Network Install**，请提供安装 URL 和 Kernel 选项（如果需要）。

网络启动(PXE)

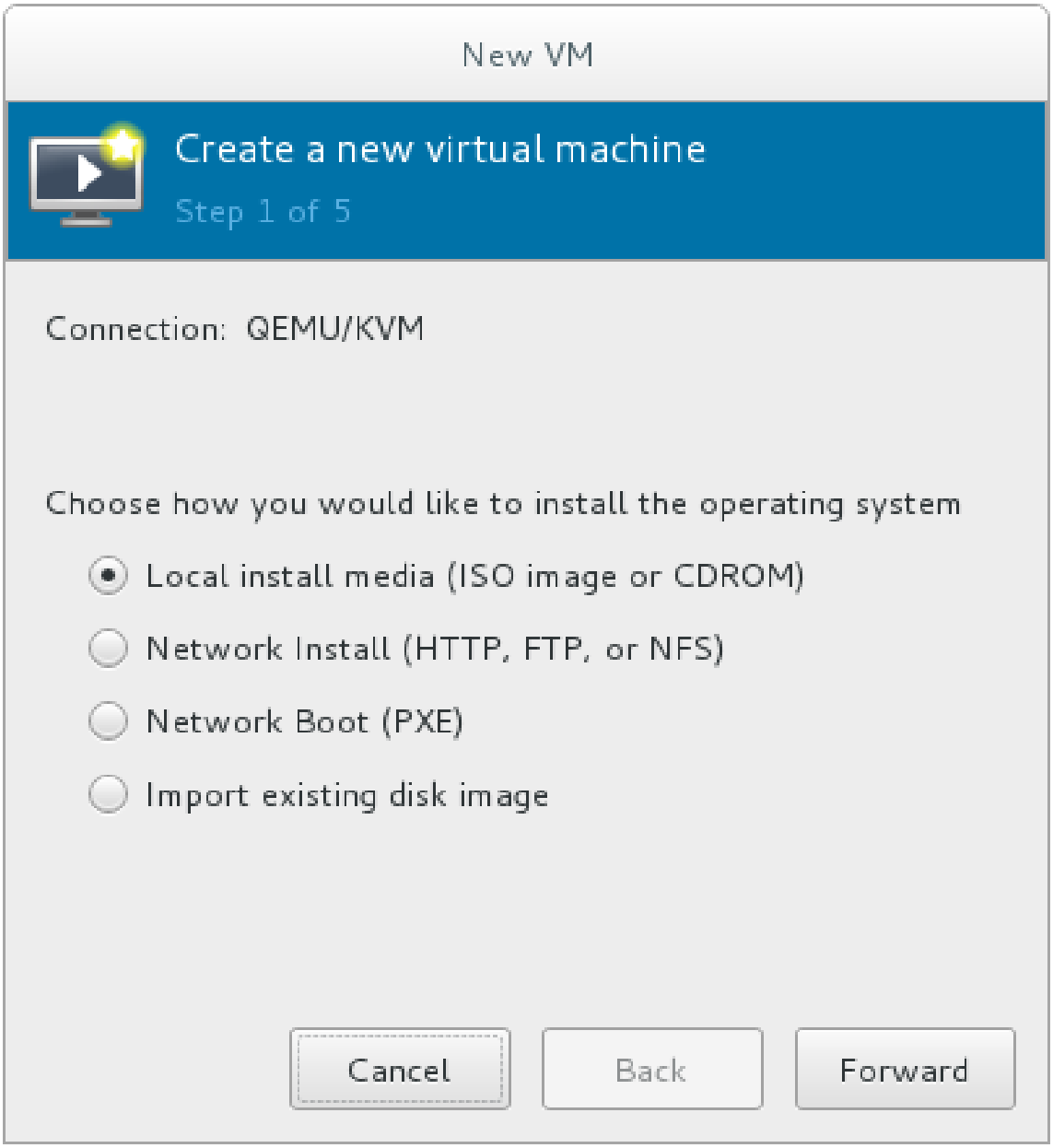
此方法使用 Preboot eXecution Environment(PXE)服务器来安装客户机虚拟机。*Red Hat Enterprise Linux 7 安装指南* 中介绍了如何设置 PXE 服务器。要使用网络引导，客户机必须具有可路由的 IP 地址或共享网络设备。

如果您选择 **Network Boot**，请继续到 STEP 5。完成所有步骤后，将发送 DHCP 请求，如果找到有效的 PXE 服务器将启动 guest 虚拟机的安装过程。

导入现有磁盘镜像

此方法可用于创建新的客户机虚拟机，并将磁盘镜像（包含预安装、可引导的操作系统）导入到其中。

图 3.2. 虚拟机安装方法

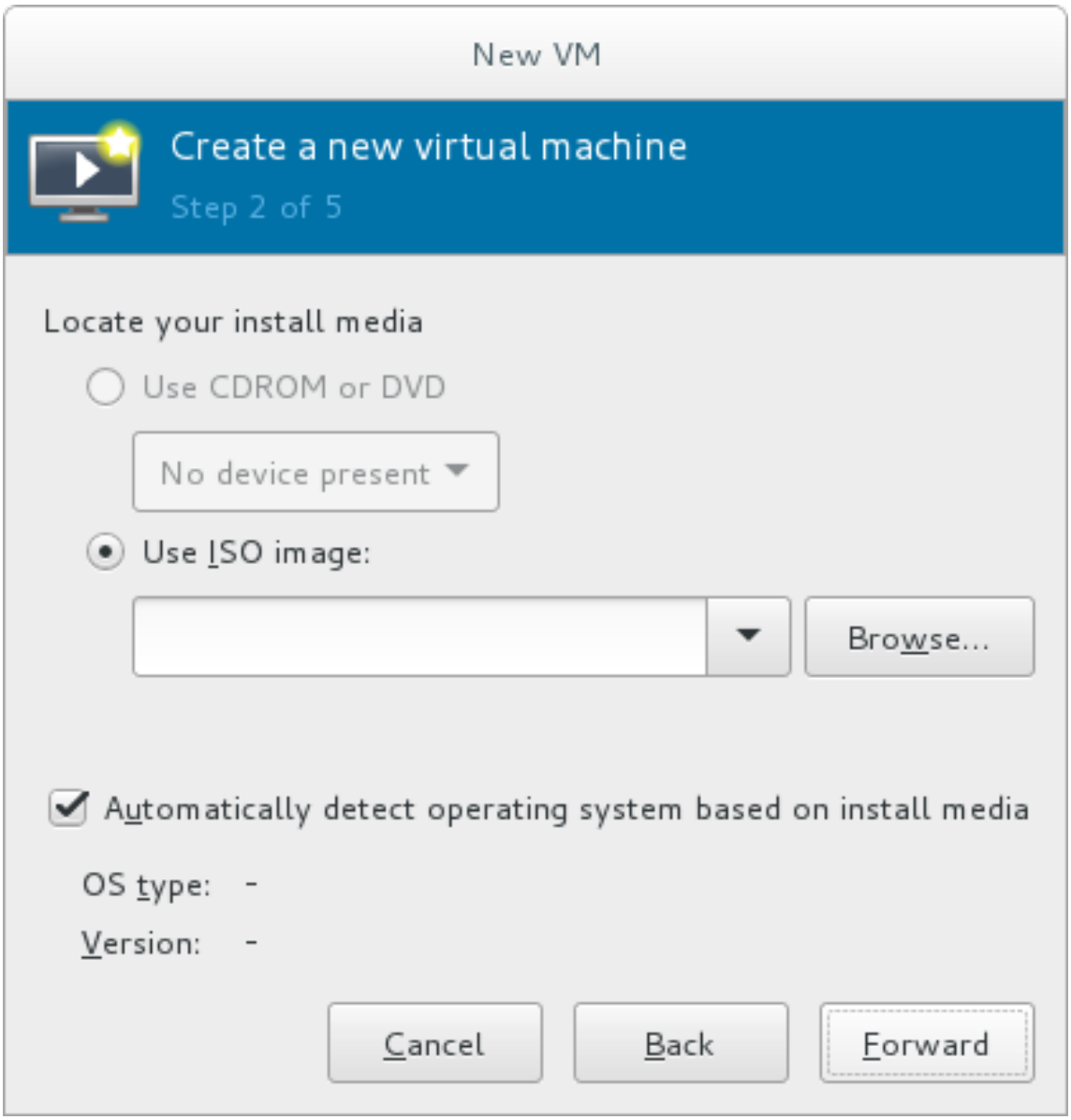


点 **Forward** 继续。

4. **选择安装源**

   a. 如果选择了 **Local 安装介质（ISO 镜像或 CDROM）**， 请指定您预期的本地安装介质。

   图 3.3. 本地 ISO 镜像安装

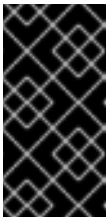


> **警告**
>
> 虽然 GUI 中目前存在 选项，但无法从主机上的物理 CD-ROM 或者 DVD 设备安装。因此，选择 **Use CDROM 或 DVD** 选项会导致虚拟机安装失败。详情请查看 Red Hat 知识库。

要从 ISO 镜像安装，请选择 **Use ISO image** 并点 **Browse...** 按钮以打开 Locate **介质卷** 窗口。

选择要使用的安装镜像，然后单击 **Choose Volume**。

如果在 Locate **介质卷** 窗口中没有显示镜像，点 **Browse Local** 按钮浏览包含安装磁盘的安装镜像或 DVD 驱动器的主机机器。选择包含安装磁盘的安装镜像或者 DVD 驱动器并点击 **Open**; 选择了卷供使用，并返回到 **Create a new virtual machine** 向导。



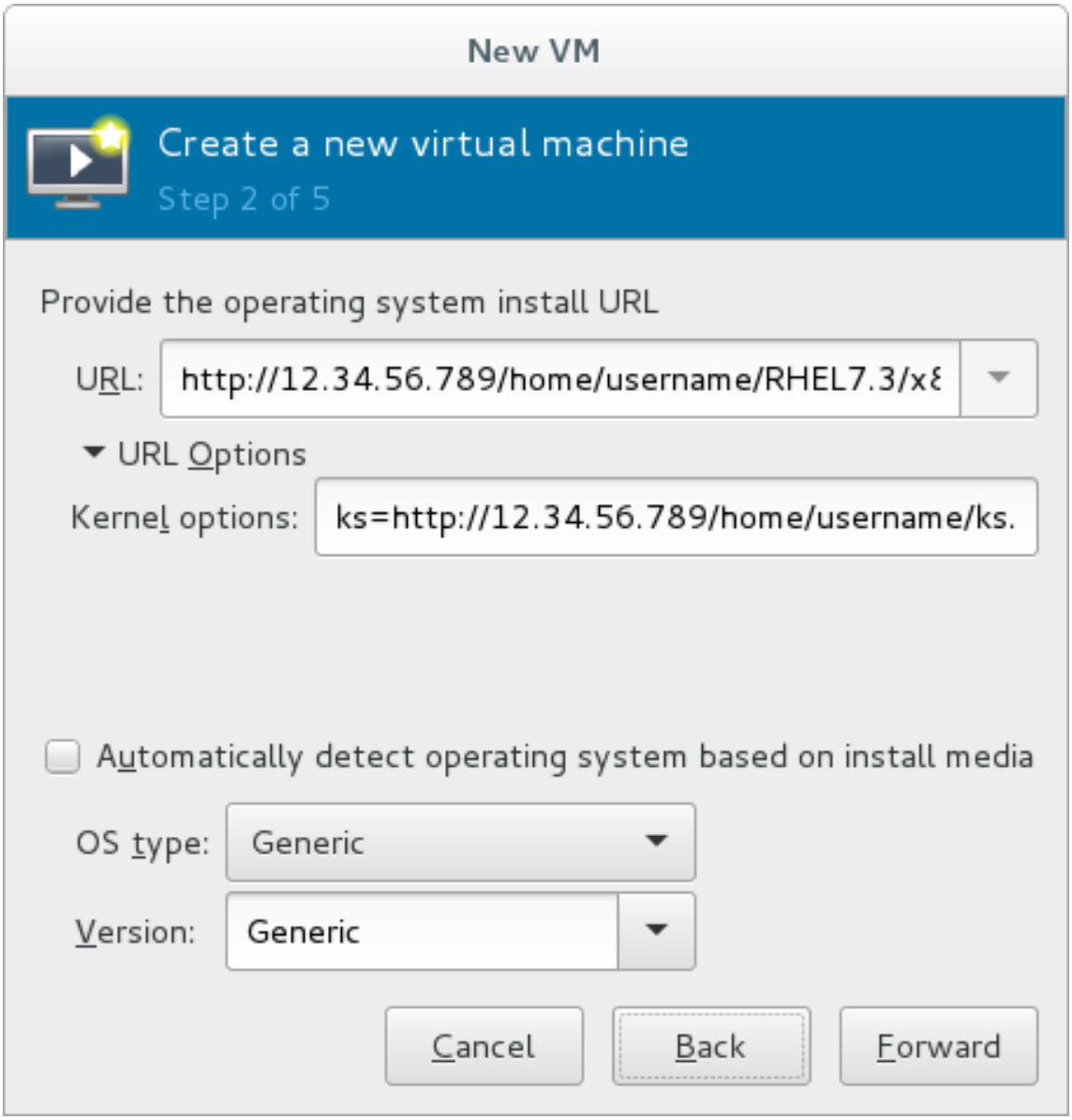
重要

对于 ISO 镜像文件和客户机存储镜像，建议使用的位置是 **/var/lib/libvirt/images/**。任何其他位置可能需要 SELinux 的额外配置。有关配置 SELinux 的详情，请查看 Red Hat Enterprise Linux Virtualization 安全指南 或 Red Hat Enterprise Linux SELinux User's Guide。

b. 如果选择了 **Network Install**，输入安装源的 URL 以及所需的内核选项（若有）。URL 必须指向安装树的根目录，它必须能够通过 HTTP、FTP 或 NFS 访问。

要执行 kickstart 安装，请在内核选项中指定 kickstart 文件的 URL，从 **ks=**开始。

图 3.4. 网络 kickstart 安装



注意

有关内核选项的完整列表，请参阅 Red Hat Enterprise Linux 7 安装指南 。

接下来，**配置安装的操作系统 类型和 版本**。确保为虚拟机选择适当的操作系统类型。这可以手动指定，或者根据 **安装介质复选框选择 Automatically detect operating system**。

点 **Forward** 继续。

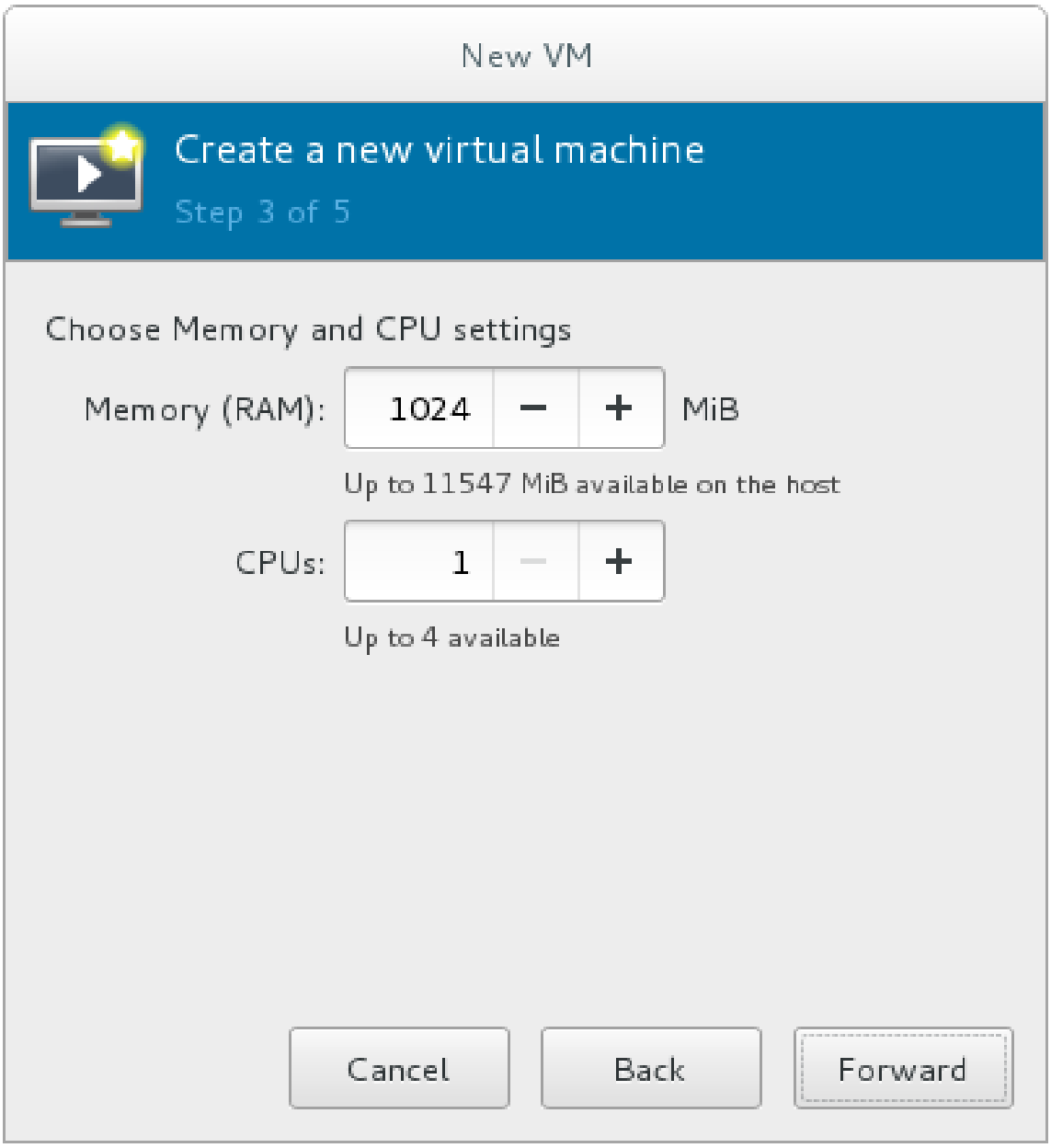
5. **配置内存(RAM)和虚拟 CPU**
   指定要分配给虚拟机的 CPU 和内存量。向导显示您可以分配的 CPU 数和内存量；这些值会影响主机和客户机的性能。

   虚拟机需要足够的物理内存(RAM)来高效地运行。红帽为虚拟机支持至少 512MB 的 RAM。红帽建议为每个逻辑内核至少 1024MB RAM。

   为虚拟机分配足够虚拟 CPU。如果虚拟机运行多线程应用程序，请分配客户机虚拟机需要运行的虚拟 CPU 数量。

   您不能分配超过主机系统中可用的物理处理器（或超线程）的虚拟 CPU。可用的虚拟 CPU 数量在 **Up to *X* available** 字段中记录。

图 3.5. 配置内存和 CPU



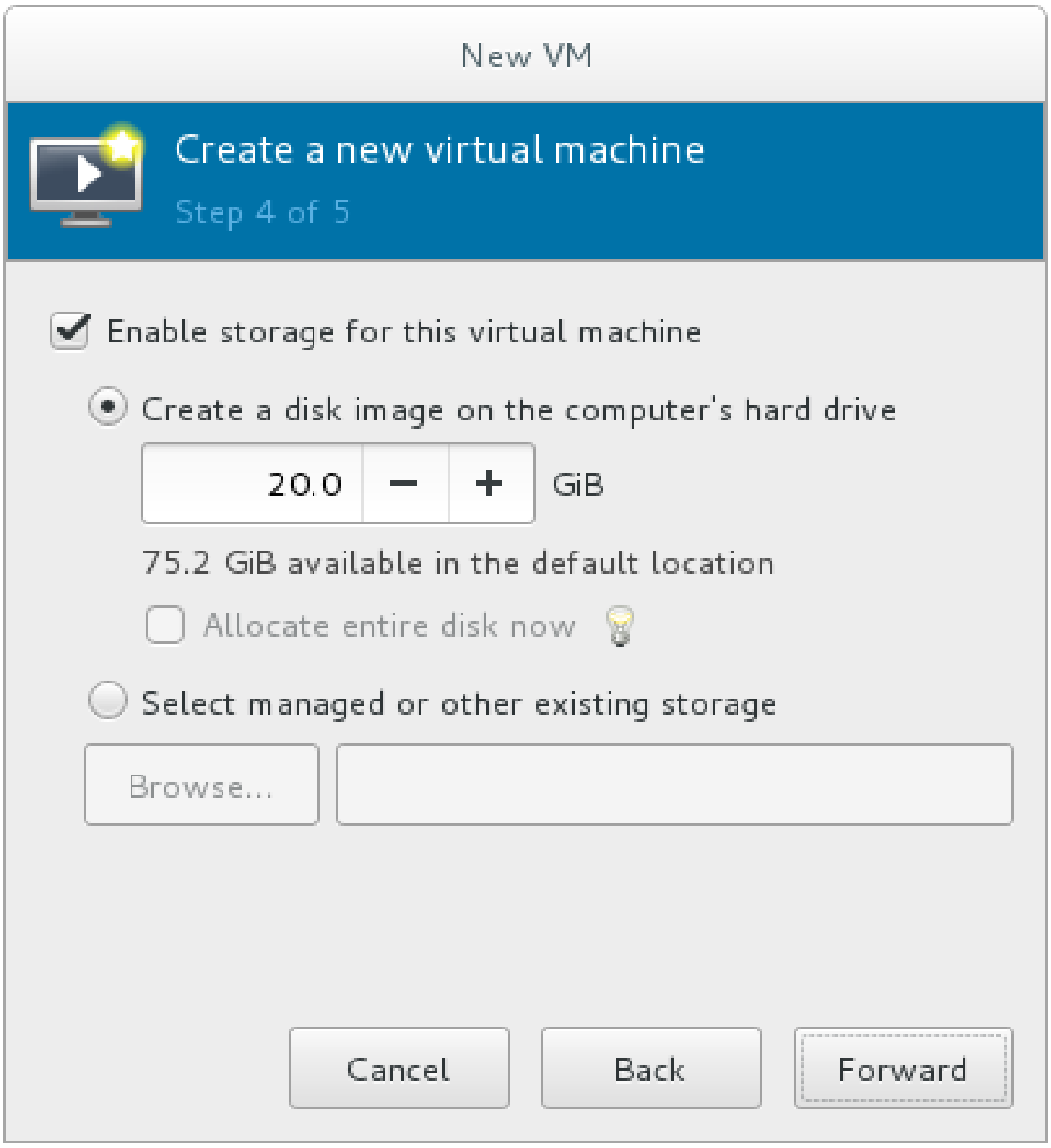
配置内存和 CPU 设置后，单击" **下一步"** 以继续。

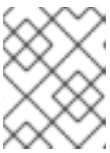
注意

内存和虚拟 CPU 可以过量使用。有关过量使用的详情请参考 *第 7 章 使用 KVM 进行过量使用*。

6. **配置存储**
为您的虚拟机启用并分配足够的空间，以及它所需的任何应用程序。对于最低安装，至少为桌面安装分配 5 GB 或至少 1 GB。

图 3.6. 配置虚拟存储





注意

实时迁移和离线迁移需要在共享网络存储上安装虚拟机。有关为虚拟机设置共享存储的详情，请参考 第 15.4 节 “共享存储示例：用于简单迁移的 NFS”。

a. **使用默认本地存储**
选择 **计算机硬盘驱动器单选按钮上创建磁盘映像**，以便在默认存储池中创建基于文件的镜像，即 **/var/lib/libvirt/images/** 目录。输入要创建的磁盘镜像的大小。如果选择了 **Allocate entire disk** 复选框，则将立即创建指定大小的磁盘镜像。如果没有，磁盘镜像会在填充时增大。

注意

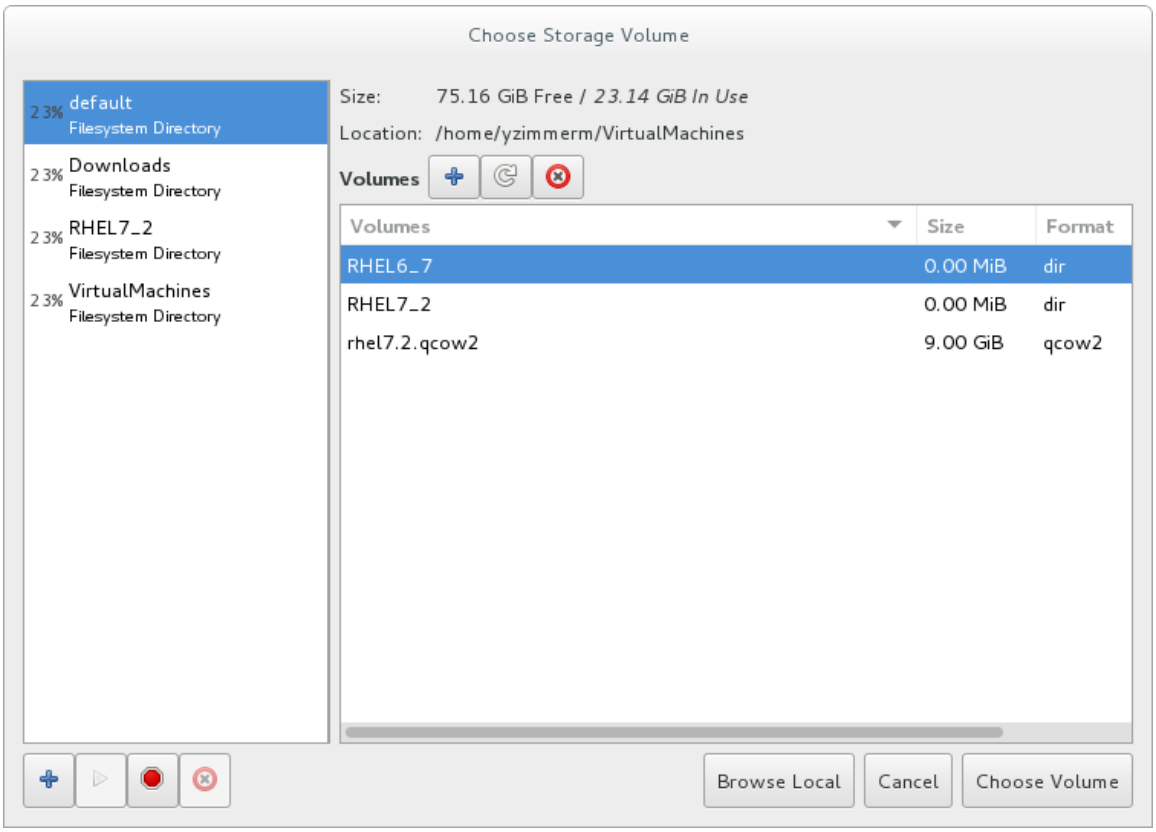
虽然存储池是一个虚拟容器，它受到两个因素的限制： 虽然它的最大大小由 qemu-kvm 和主机物理机器上的磁盘大小进行调整。存储池可能没有超过主机物理机器上磁盘的大小。最大大小如下：

- virtio-blk = 2^63 字节或 8 Exabytes（使用原始文件或磁盘）
- Ext4 = ~ 16 TB（使用 4 KB 的块大小）
- XFS = ~8 Exabytes
- 在尝试非常大的镜像大小时，应评估/调整其 metadata 和主机文件系统，保持自己的元数据和可扩展性。使用原始磁盘意味着会影响可扩展性或最大大小的层数。

点击 **Forward** 在本地硬盘中创建磁盘镜像。或者，选择 **Select managed 或其他现有存储**，然后选择 **Browse** 来配置受管存储。

b. **使用存储池**
如果您选择 **Select managed 或 other existing storage** 来使用存储池，请点击 **Browse** 打开 Locate **或 create storage volume** 窗口。

图 3.7. Choose Storage Volume 窗口



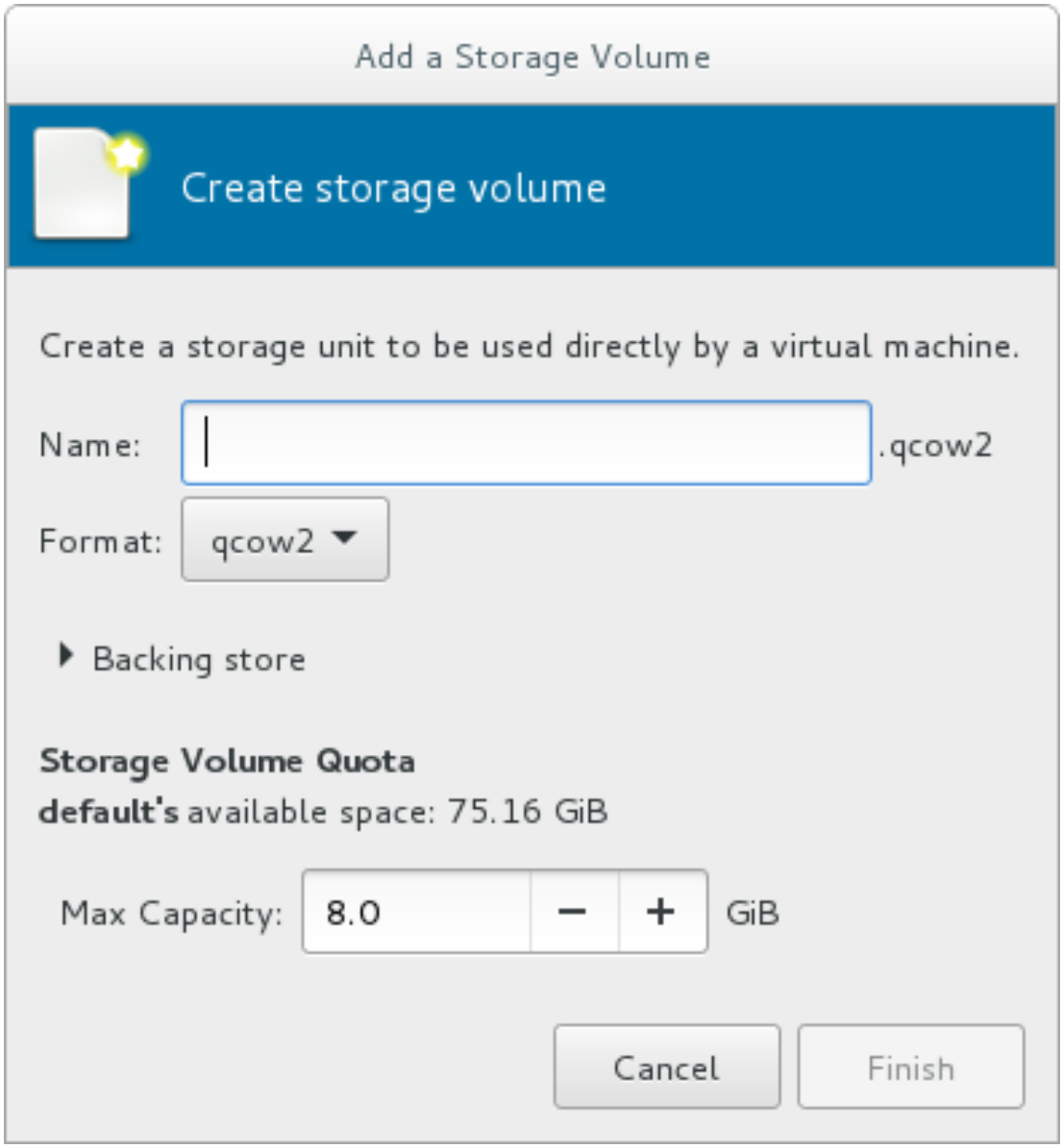
i. 从 **Storage Pools** 列表中选择存储池。

ii. 可选：点击  创建新存储卷。此时会出现 **Add a Storage Volume** 屏幕。输入新存储卷的名称。

从 **Format** 下拉菜单中选择 Format 选项。格式选项包括 raw、qcow2、和 qed。根据需

从 **Format** 下拉菜单中选择 Format 选项。格式选项包括 raw、qcow2、和 qed。根据需要调整其他字段。请注意，这里使用的 qcow2 版本为版本 3。要更改 qcow 版本，请参阅 第 23.19.2 节 “设置目标元素”

图 3.8. Add a Storage Volume 窗口



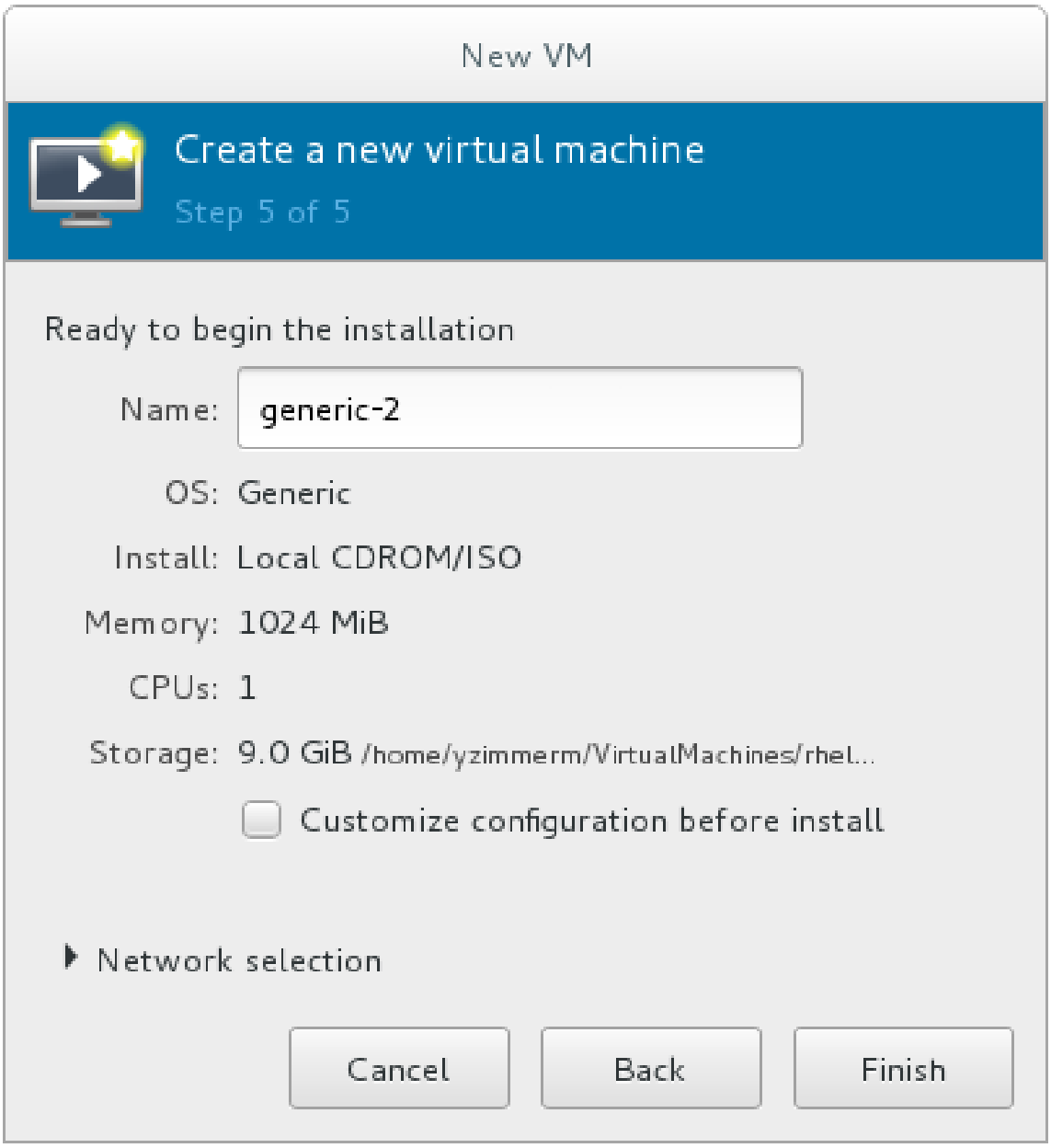
选择新卷，再单击 **Choose volume**。接下来，单击 **Finish** 以返回到 **New VM** 向导。点 **Forward** 继续。

7. **名称和最终配置**
将虚拟机命名为。虚拟机名称可以包含字母、数字和以下字符：下划线(_)、句点( )和连字符(-)。虚拟机名称对于迁移来说必须是唯一的，且不能仅包含数字。

默认情况下，将使用名为"default"的网络的网络地址转换(NAT)创建虚拟机。要更改网络选择，请点击 **网络选择** 并选择主机设备和源模式。

验证虚拟机的设置，并在您满意时单击 **"完成"** ；这将创建具有指定网络设置、虚拟化类型和架构的虚拟机。

图 3.9. 验证配置



或者，要进一步配置虚拟机的硬件，**请在安装前选中 Customize configuration**，以更改客户机的存储或网络设备，以使用半虚拟化(virtio)驱动程序或添加额外的设备。这会打开另一个向导，供您添加、删除和配置虚拟机的硬件设置。

注意

Red Hat Enterprise Linux 4 或 Red Hat Enterprise Linux 5 虚拟机无法使用图形模式安装。因此，您必须选择"Cirrus"而不是"QXL"作为显卡。

配置虚拟机硬件后，请单击"**应用** "。**virt-manager** 将根据指定的硬件设置创建虚拟机。

警告

从远程介质安装 Red Hat Enterprise Linux 7 客户机虚拟机时，但没有配置的 TCP/IP 连接时，安装会失败。但是，在这样的情况下，当安装 Red Hat Enterprise Linux 5 或 6 的客户机虚拟机时，安装程序会打开"配置 TCP/IP"接口。

有关这一差异的更多信息，请参阅 相关知识库文章。

点 **Finish** 继续进入 Red Hat Enterprise Linux 安装序列。有关安装 Red Hat Enterprise Linux 7 的详情，请参考 Red Hat Enterprise Linux 7 安装指南 。

Red Hat Enterprise Linux 7 客户机虚拟机现在从 ISO 安装磁盘镜像创建。

3.4. VIRT-INSTALL 和 VIRT-MANAGER 安装选项的比较

这个表格提供了与安装虚拟机时对应的 **virt-install** 和 **virt-manager** 安装选项进行比较的快速参考。

大多数 **virt-install** 选项都不是必须的。最低要求是 **--name**、--**memory**、guest 存储（**--disk** 、**--filesystem** 或 **--disk none**），以及一个安装方法（**--location** 、**--cdrom** 、**--pxe**、-**import** 或 **boot**）。这些选项可以通过参数进一步指定；要查看命令选项和相关参数的完整列表，请输入以下命令：

```
# virt-install --help
```

在 **virt-manager** 中，至少需要名称、安装方法、内存(RAM)、vCPU 和存储。

表 3.1. guest 安装的 virt-install 和 virt-manager 配置比较

虚拟机上的配置	virt-install 选项	virt-manager 安装向导标签和步骤号
虚拟机名称	--name, -n	名称（步骤 5）
分配的 RAM(MiB)	--ram, -r	内存(RAM)（第 3 步）
Storage - 指定存储介质	--disk	为此虚拟机启用存储 → 在计算机的硬盘驱动器中创建磁盘镜像，或者选择托管或其他现有存储（第 4 步）
Storage - 将主机目录导出到客户端	--filesystem	为此虚拟机启用存储 → Select managed 或其它现有存储（第 4 步）
Storage - 在客户机中配置本地磁盘存储	--nodisks	取消选择此虚拟机的启用存储复选框（第 4 步）

虚拟机上的配置	virt-install 选项	virt-manager 安装向导标签和步骤号
安装介质位置（本地安装）	--file	本地安装介质 → Locate your安装介质（第 1-2 步）
使用分发树（网络安装）进行安装.	--location	网络 install → URL（步骤 1-2）
使用 PXE 安装客户端	--pxe	网络引导（第 1 步）
vCPU 数量	--vcpus	CPU（第 3 步）
主机网络	--network	高级选项下拉菜单（第 5 步）
操作系统变体/版本	--os-variant	版本（第 2 步）
图形显示方法	--graphics, --nographics	** virt-manager 只提供 GUI 安装*

第 4 章 克隆虚拟机

创建客户机副本时有两种客户机虚拟机实例：

- *克隆* 是指单个虚拟机的实例。克隆可用于设置相同虚拟机的网络，也可以将它们分发到其他目的地。
- *模板是虚拟机的* 实例，设计为用作克隆的源。您可以从模板创建多个克隆，并对每个克隆进行小的修改。这对于查看系统中这些更改的影响非常有用。

克隆和模板都是虚拟机实例。它们之间的差别在于如何使用它们。

要使创建的克隆正常工作，在克隆之前，必须删除与正在克隆的虚拟机唯一的信息和配置。需要删除的信息会因使用克隆的方式而异。

要删除的信息和配置可在以下任意级别中：

- *平台级别* 信息和配置包括虚拟化解决方案分配给虚拟机的任何内容。示例包括网络接口卡(NIC)及其 MAC 地址的数量。
- *客户机操作系统级别* 信息和配置包括虚拟机中配置的任何内容。示例包括 SSH 密钥。
- *应用程序级别* 信息和配置包括在虚拟机上安装的应用程序所配置的所有内容。示例包括激活代码和注册信息。

注意

本章不包括有关删除应用级别的信息，因为信息和方法特定于每个应用。

因此，必须从虚拟机中删除一些信息和配置，而其他相关信息和配置必须使用虚拟化环境（如虚拟机管理器或 VMware）从虚拟机中删除。

注意

有关克隆存储卷的详情请参考 第 13.3.2.1 节 “使用 virsh 创建存储卷”。

4.1. 为 CLONING 准备虚拟机

在克隆虚拟机前，必须在其磁盘镜像上运行 virt-sysprep 程序或执行以下步骤来准备：

过程 4.1. 准备虚拟机进行克隆

1. **设置虚拟机**
   a. 构建要用于克隆或模板的虚拟机。
      - 在克隆上安装所需的任何软件。
      - 为操作系统配置任何非唯一设置。
      - 配置任何非唯一应用设置。
2. **删除网络配置**
   a. 使用以下命令删除所有持久性 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 的名称可以是 eth1 而不是 eth0。

b. 通过编辑 **/etc/sysconfig/network-scripts/ifcfg-eth[x]** 来删除 ifcfg 脚本中的唯一网络详情：

i. 删除 HWADDR 和 Static 行

注意

如果 HWADDR 与新客户机的 MAC 地址不匹配，则会忽略 ifcfg。因此，务必要从文件中删除 HWADDR。

```
DEVICE=eth[x]
BOOTPROTO=none
ONBOOT=yes
#NETWORK=10.0.1.0       <- REMOVE
#NETMASK=255.255.255.0  <- REMOVE
#IPADDR=10.0.1.20       <- REMOVE
#HWADDR=xx:xx:xx:xx:xx  <- REMOVE
#USERCTL=no             <- REMOVE
# Remove any other *unique* or non-desired settings, such as UUID.
```

ii. 确保 DHCP 配置仍不包含 HWADDR 或任何唯一信息。

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

iii. 确保该文件包含以下行：

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

c. 如果有以下文件，请确保它们包含相同的内容：

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

注意

如果虚拟机使用了 NetworkManager 或任何特殊设置，请确保从 ifcfg 脚本中删除任何其他唯一信息。

3. **删除注册详情**

a. 使用以下命令之一删除注册详情：

- 对于 Red Hat Network(RHN)注册的客户机虚拟机，使用以下命令：

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

- 对于 Red Hat Subscription Manager(RHSM)注册的客户机虚拟机：
  - 如果没有使用原始虚拟机，使用以下命令：

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

  - 如果使用原始虚拟机，请只运行以下命令：

```
# subscription-manager clean
```

    原始 RHSM 配置集保留在门户网站中。要在克隆虚拟机后重新激活 RHSM 注册，请执行以下操作：

    1. 获取您的客户身份代码：

```
# subscription-manager identity
subscription-manager identity: 71rd64fx-6216-4409-bf3a-e4b7c7bd8ac9
```

    2. 使用获取的 ID 代码注册虚拟机：

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

4. **删除其他唯一详情**

   a. 使用以下命令删除任何 sshd 公钥/私钥对：

```
# rm -rf /etc/ssh/ssh_host_*
```

注意

删除 ssh 密钥可防止 ssh 客户端不信任这些主机的问题。

   b. 删除任何其他特定于应用的标识符或配置，这可能在多台计算机上运行时造成冲突。

5. **将虚拟机配置为在下一次启动时运行配置向导**

   a. 将虚拟机配置为在下一次引导时运行相关的配置向导，方法是执行以下操作之一：

   - 对于 Red Hat Enterprise Linux 6 和以下项，使用以下命令在 root 文件系统中创建一个名为 .un configure 的空文件：

```
# touch /.unconfigured
```

   - 对于 Red Hat Enterprise Linux 7，运行以下命令启用第一个引导和 initial-setup 向导：

```
# sed -ie 's/RUN_FIRSTBOOT=NO/RUN_FIRSTBOOT=YES/'
/etc/sysconfig/firstboot
# systemctl enable firstboot-graphical
# systemctl enable initial-setup-graphical
```

注意

在下次引导时运行的向导取决于已从虚拟机中删除的配置。另外，在第一次引导克隆时，建议您更改主机名。

4.2. 克隆虚拟机

在继续克隆前，请关闭虚拟机。您可以使用 **virt-clone** 或 **virt-manager** 克隆虚拟机。

4.2.1. 使用 virt-clone 克隆虚拟机

您可以使用 **virt-clone** 从命令行克隆虚拟机。

请注意，您需要 root 权限让 **virt-clone** 成功完成。

virt-clone 命令提供了可以在命令行上传递的多个选项。这包括常规选项、存储配置选项、网络配置选项和各种选项。只需要 **--original**。要查看选项的完整列表，请输入以下命令：

```
# virt-clone --help
```

virt-clone man page 还记录了每个命令选项、重要的变量和示例。

以下示例演示了如何在默认连接上克隆名为"demo"的客户机虚拟机，并自动生成新的名称和磁盘克隆路径。

例 4.1. 使用 virt-clone 克隆客户端

```
# virt-clone --original demo --auto-clone
```

以下示例演示了如何使用多个磁盘克隆名为"demo"的 QEMU 客户机代理虚拟机。

例 4.2. 使用 virt-clone 克隆客户端

```
# virt-clone --connect qemu:///system --original demo --name newdemo --file
/var/lib/libvirt/images/newdemo.img --file /var/lib/libvirt/images/newdata.img
```

4.2.2. 使用 virt-manager 克隆客户机

这个步骤描述了使用 virt-manager 实用程序克隆客户机虚拟机。

过程 4.2. 使用 virt-manager 克隆虚拟机

1. open virt-manager

启动 **virt-manager**。从 **应用程序菜单** 和 **系统工具** 子菜单 **启动虚拟机管理器** 应用程序。或者，以 root 身份运行 **virt-manager** 命令。

从 **Virtual Machine Manager** 中的客户机虚拟机列表中选择您要克隆的客户机虚拟机。

在您要克隆的 guest 虚拟机上单击鼠标右键，然后选择 **Clone**。此时会打开 Clone Virtual Machine 窗口。

图 4.1. 克隆虚拟机窗口

2. **配置克隆**

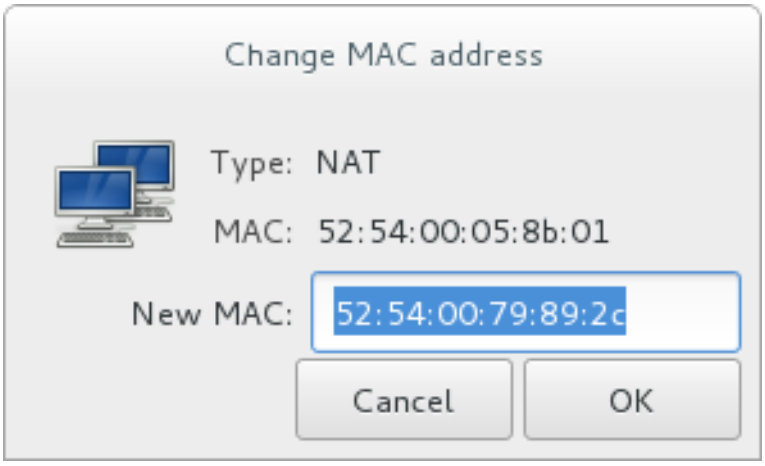
   - 要更改克隆的名称，请为克隆输入新名称。

- 要更改网络配置，请点击 **Details**。

  为克隆输入新的 MAC 地址。

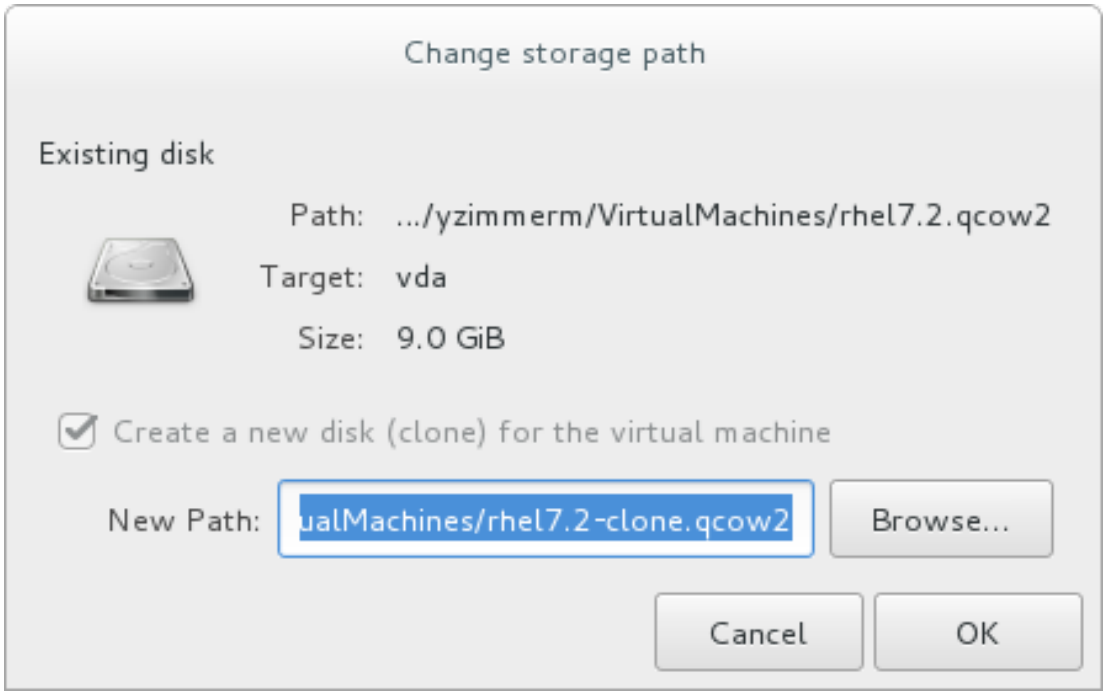
  点**确定**。

  图 4.2. 更改 MAC 地址窗口



- 对于克隆的客户机虚拟机中的每个磁盘，请选择以下选项之一：
  - **克隆此磁盘** - 将克隆为克隆的客户机虚拟机的磁盘
  - ***与 guest 虚拟机名称*共享磁盘 - 磁盘将由将被克隆及其克隆的客户端虚拟机共享**
  - **Details** - 打开更改存储路径窗口，该窗口为磁盘启用新路径

    图 4.3. 更改 存储路径 窗口

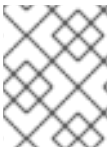


3. **克隆客户机虚拟机**
   单击 **Clone**。

第 5 章 KVM 半虚拟化(VIRTIO)驱动程序

半虚拟化驱动程序可增强客户机性能，降低客户机 I/O 延迟，并将吞吐量几乎增加到裸机级别。建议将半虚拟化驱动程序用于运行 I/O 密集型任务和应用程序的完全虚拟化客户机。

VirtIO 驱动程序是 KVM 泛虚拟化设备驱动程序，可用于 KVM 主机上运行的 虚拟机。这些驱动程序包含在 **virtio** 软件包中。virtio 软件包支持块（存储）设备和网络接口控制器。



注意

PCI 设备受虚拟化系统架构的限制。在使用分配的设备时，请参阅 第 16 章 *虚拟机设备配置* 以了解其他限制。

5.1. 为现有存储设备使用 KVM VIRTIO 驱动程序

您可以修改附加到客户机的现有硬盘设备，以使用 **virtio** 驱动程序，而不是虚拟化 IDE 驱动程序。本节中显示的示例编辑 libvirt 配置文件。请注意，不需要关闭客户端虚拟机来执行这些步骤，但更改不会应用，直到 guest 完全关闭并重新引导为止。

过程 5.1. 为现有设备使用 KVM virtio 驱动程序

1. 在继续执行此流程前，请确保**您已安装了**适当的驱动程序(viostor)。
2. 以 root 身份运行 **virsh edit *guestname*** 命令，编辑设备的 XML 配置文件。例如： **virsh edit guest1**。配置文件位于 **/etc/libvirt/qemu/** 目录中。
3. 以下是使用虚拟化 IDE 驱动程序基于文件的块设备。这对于虚拟机的典型条目不使用 virtio 驱动程序。

```
<disk type='file' device='disk'>
  ...
  <source file='/var/lib/libvirt/images/disk1.img'/>
  <target dev='hda' bus='ide'/>
 <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
</disk>
```

4. 通过将 **bus=** 条目修改为 virtio，将条目更改为使用 **virtio** 设备。请注意，如果磁盘之前是 IDE，它有一个类似于 **hda**、**hdb** 或 **hdc** 的目标。当更改为 **bus=virtio** 时，目标需要相应地更改为 **vda**、**vdb** 或 **vdc**。

```
<disk type='file' device='disk'>
  ...
  <source file='/var/lib/libvirt/images/disk1.img'/>
  <target dev='vda' bus='virtio'/>
 <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
</disk>
```

5. 删除 **磁盘标签** 中的 **地址** 标签。此过程必须完成。libvirt **将在虚拟机下次启动时正确重新生成地址** 标签。

或者，**virt-manager**、**virsh attach-disk** 或 **virsh attach-interface** 可使用 virtio 驱动程序添加新设备。

有关使用 Virtio 的详情，请查看 libvirt 网站： http://www.linux-kvm.org/page/Virtio

5.2. 为新存储设备使用 KVM VIRTIO 驱动程序

此流程涵盖在 **virt-manager** 中使用 KVM virtio 驱动程序创建新存储设备。

或者，可以使用 **virsh attach-disk** 或 **virsh attach-interface** 命令使用 virtio 驱动程序附加设备。

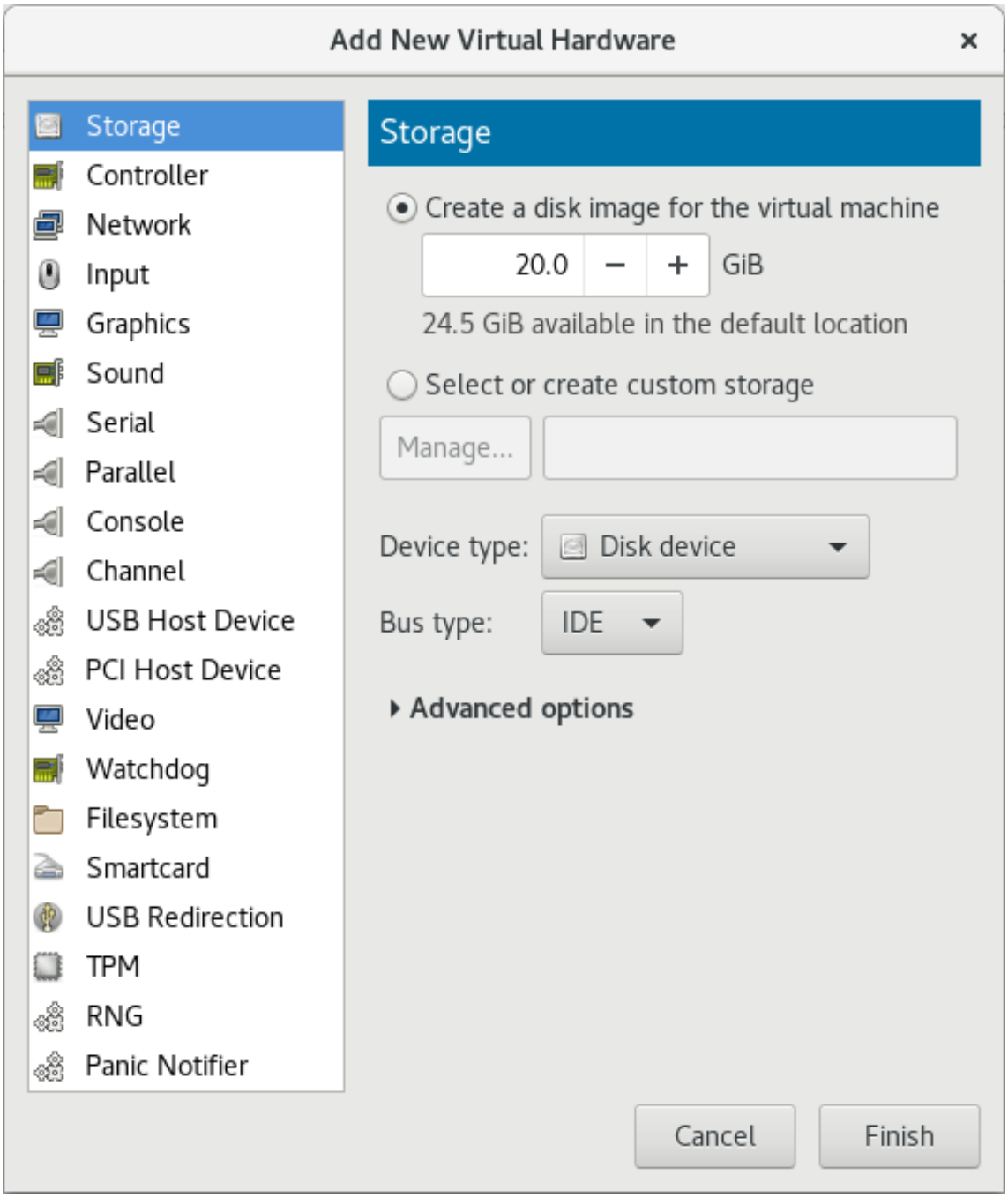
重要

在继续安装新设备前，请确保已在客户机上安装驱动程序。如果驱动程序不可用，该设备将无法识别且无法使用。

过程 5.2. 使用 virtio 存储驱动程序添加存储设备

1. 通过双击 **virt-manager** 中的 guest 名称，打开 guest 虚拟机。
2. 点击  **打开 Show Virtual hardware details** 选项卡。
3. 在 **显示虚拟硬件详细信息** 选项卡中，单击 **添加硬件** 按钮。
4. **选择硬件类型**
   选择 **Storage** 作为 **硬件类型**。

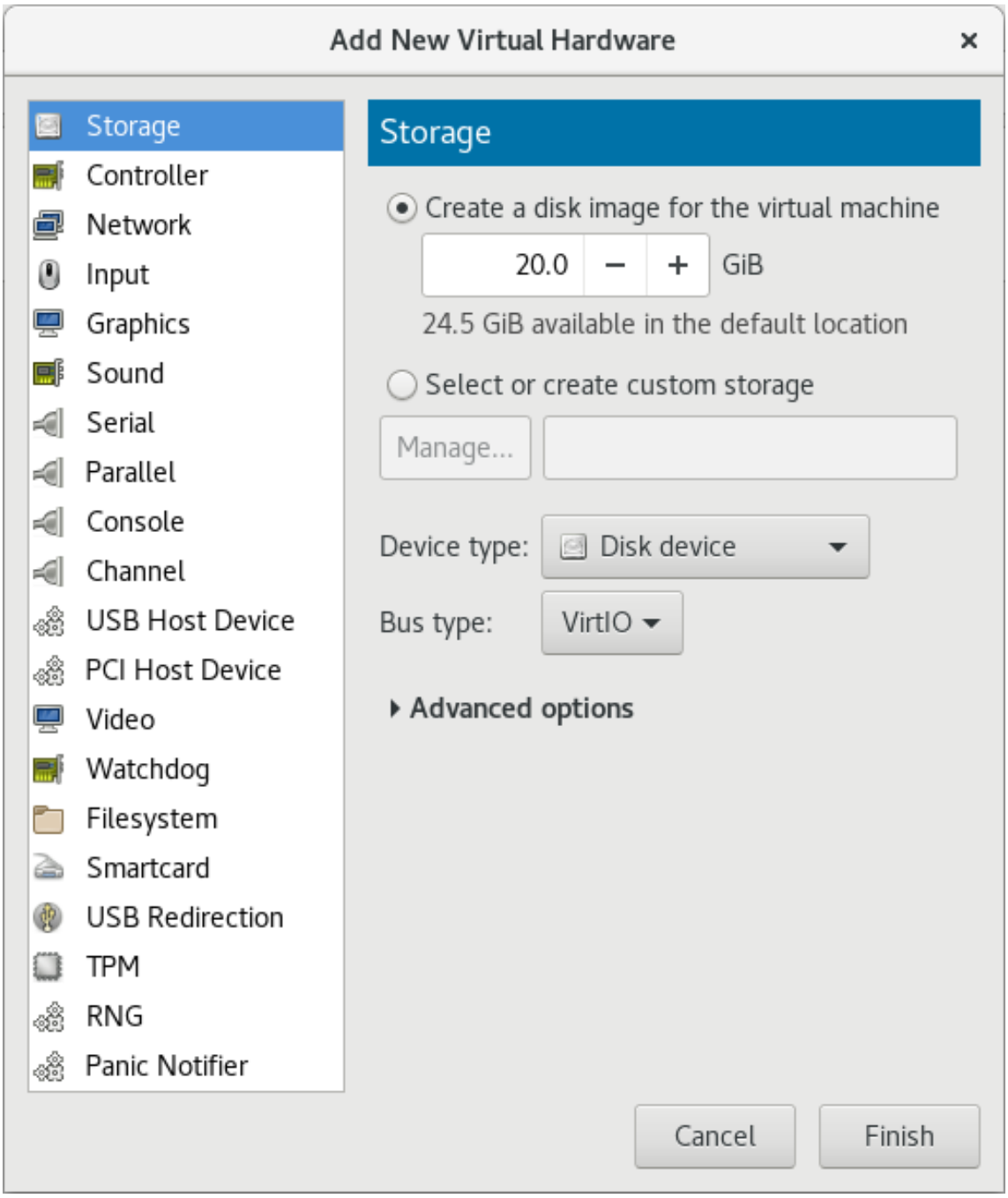
图 5.1. Add new virtual hardware 向导



5. **选择存储设备和驱动程序**
创建新磁盘镜像或选择存储池卷。

将 **设备类型设置为** 磁盘设备，并将 **Bus type** 设为 **VirtIO** 以使用 virtio 驱动程序。

图 5.2. Add New Virtual Hardware 向导

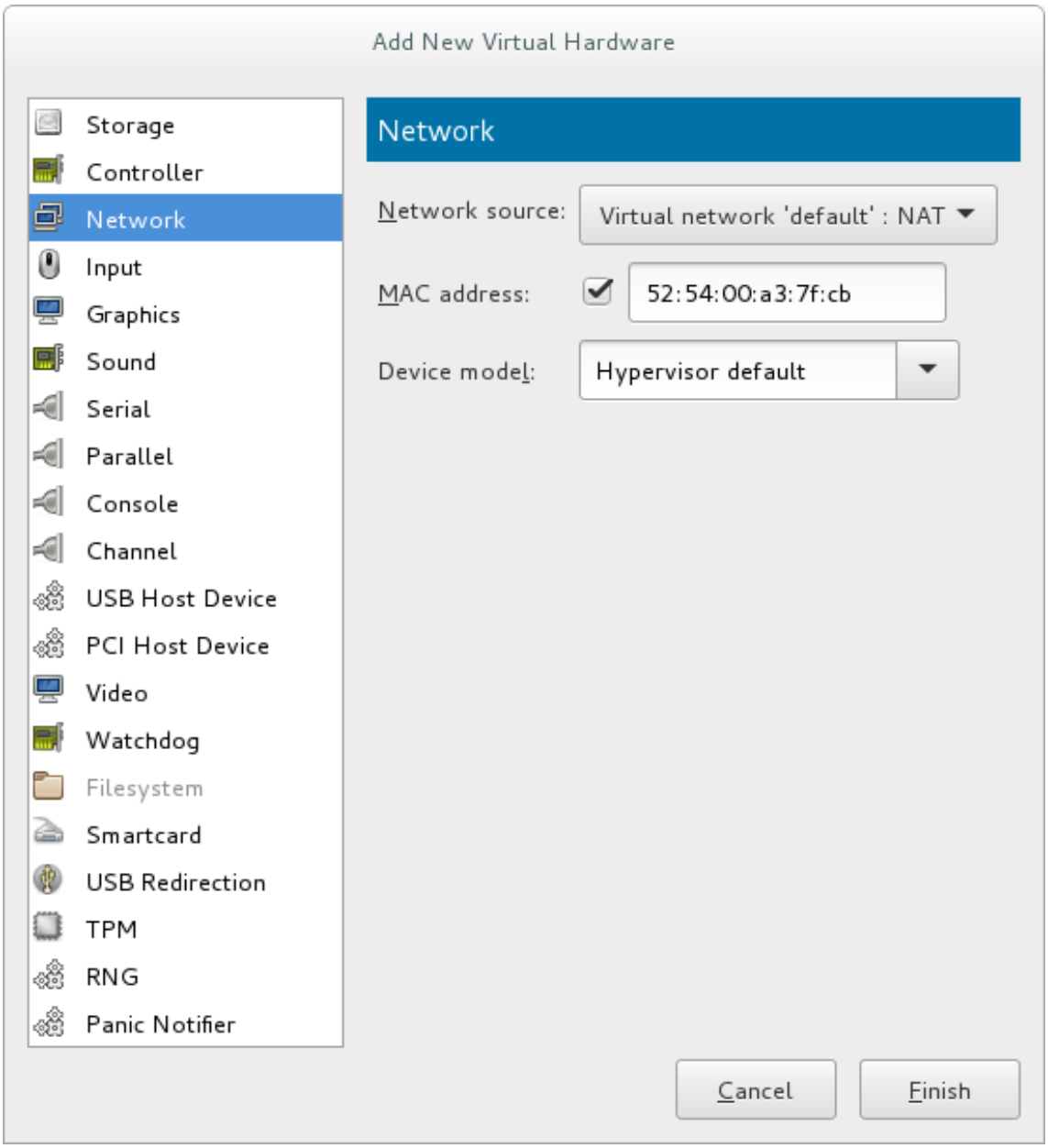


点 **Finish** 以完成此流程。

过程 5.3. 使用 virtio 网络驱动程序添加网络设备

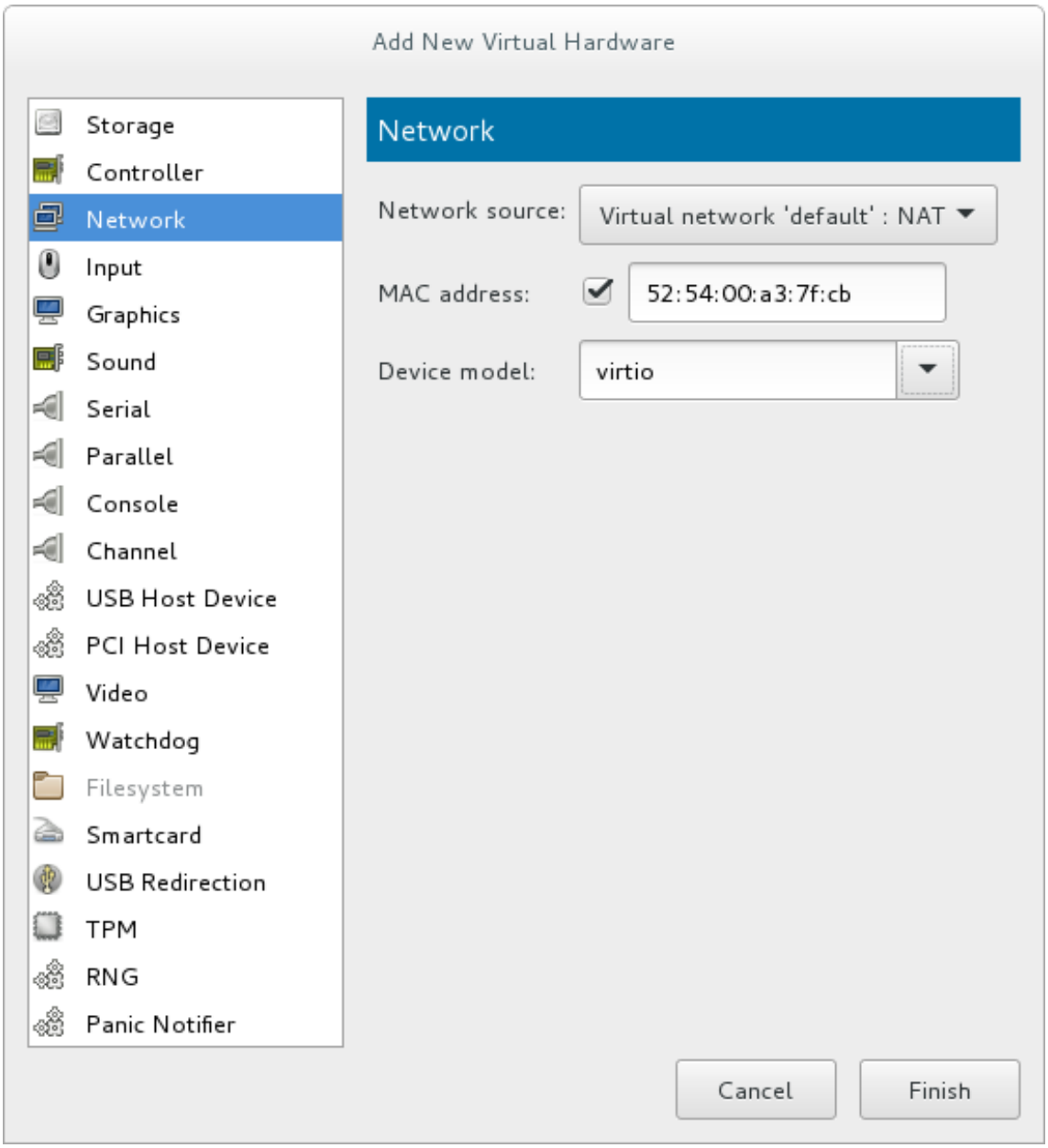
1. 通过双击 **virt-manager** 中的 guest 名称，打开 guest 虚拟机。
2. 点击  **打开 Show Virtual hardware details** 选项卡。
3. 在 **显示虚拟硬件详细信息** 选项卡中，单击 **添加硬件** 按钮。
4. **选择硬件类型**
   选择 **Network** 作为 **硬件类型**。

图 5.3. Add new virtual hardware 向导



5. **选择网络设备和驱动程序**
   将 **设备模型** 设置为 **virtio** 以使用 virtio 驱动程序。选择所需的 **主机设备**。

图 5.4. Add new virtual hardware 向导



点 **Finish** 以完成此流程。

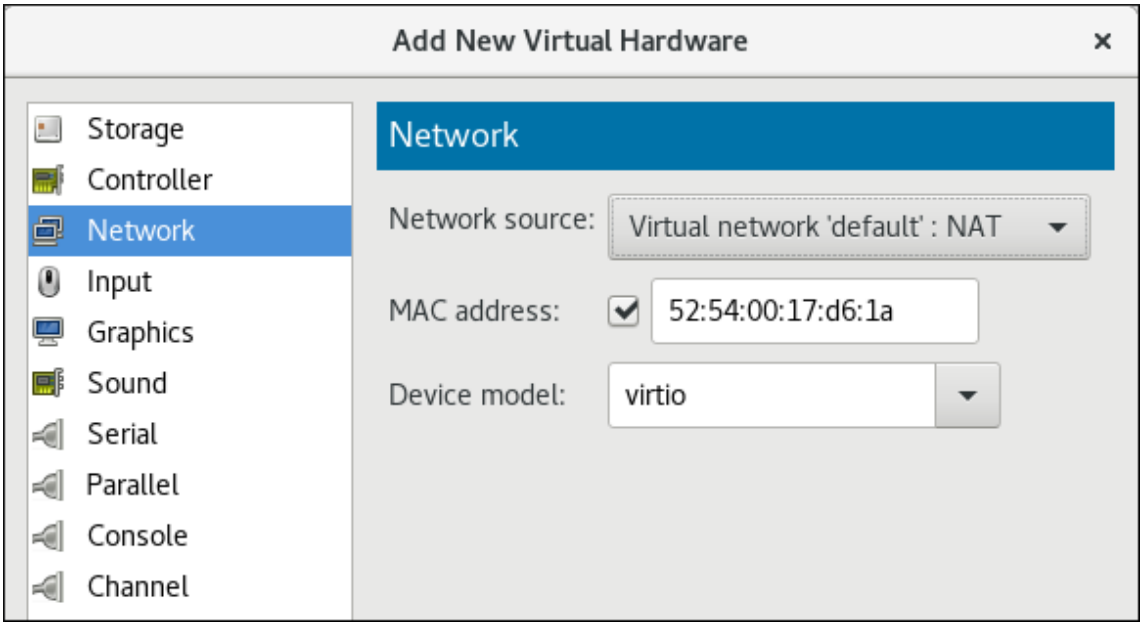
添加所有新设备后，重启虚拟机。在 guest 重新启动之前，虚拟机可能无法识别设备。

5.3. 为网络接口设备使用 KVM VIRTIO 驱动程序

当网络接口使用 KVM VirtIO 驱动程序时，KVM 不会模拟删除处理开销的网络硬件，并可提高客户机性能。在 Red Hat Enterprise Linux 7 中，virtio 用作默认网络接口类型。但是，如果在您的系统中进行了不同的配置，您可以使用以下步骤：

- 要将 **virtio 网络设备** 附加到客户端，请使用带有 **model --virtio** 选项的 **virsh attach-interface** 命令。

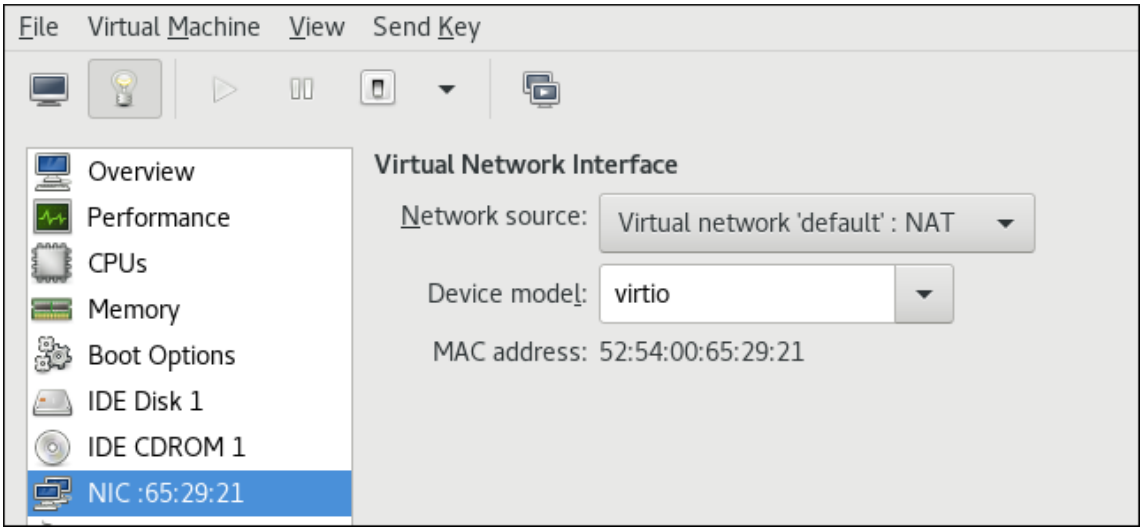
  或者，在 **virt-manager** 界面中，导航到 guest **的虚拟硬件详细信息屏幕，然后单击添加硬件。** 在 **Add New Virtual Hardware** 屏幕中，选择 **Network**，并将 **设备模型** 改为 **virtio** ：



- **要将现有接口的类型更改为 virtio，** 请使用 **virsh edit** 命令编辑预期客户机的 XML 配置，并将 **型号类型** 属性改为 **virtio**，例如：

```
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <model type='virtio'/>
    <driver name='vhost' txmode='iothread' ioeventfd='on' event_idx='off'/>
  </interface>
</devices>
...
```

或者，在 virt-manager 界面中，导航到客户机 **的虚拟硬件详细信息屏幕**，选择 **NIC** 项，并将 **设备模型** 改为 **virtio** ：



注意

如果客户机中的网络接口命名在重新引导时不一致，请确保向客户机提供的所有接口都使用相同的设备模型，最好是 **virtio-net**。详情请查看 红帽知识库。

第 6 章 网络配置

本章介绍基于 libvirt 的客户机虚拟机所使用的常见网络配置。

Red Hat Enterprise Linux 7 支持以下虚拟化网络设置：

- 使用网络地址转换(*NAT*)的虚拟网络.
- 使用 PCI 设备分配直接分配的物理设备
- 使用 PCIe SR-IOV 直接分配的虚拟功能
- 网桥网络

您必须启用 NAT、网络桥接或直接分配 PCI 设备，以允许外部主机访问客户机虚拟机上的网络服务。

6.1. 使用 LIBVIRT 进行网络地址转换(NAT)

共享网络连接的最常见方法是使用网络地址转换(NAT)转发（也称为虚拟网络）。

主机配置

每个标准 **libvirt** 安装都提供虚拟机的基于 NAT 的连接作为默认虚拟网络。使用 **virsh net-list --all** 命令验证它可用。

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

如果缺少这个选项，可针对客户机在 XML 配置文件（如 /etc/libvirtd/qemu/myguest.xml）中使用以下内容：

```
# ll /etc/libvirt/qemu/
total 12
drwx------. 3 root root 4096 Nov  7 23:02 networks
-rw-------. 1 root root 2205 Nov 20 01:20 r6.4.xml
-rw-------. 1 root root 2208 Nov  8 03:19 r6.xml
```

默认网络从 **/etc/libvirt/qemu/networks/default.xml**定义

将默认网络标记为自动启动：

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

启动默认网络：

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

libvirt 默认网络运行后，您将看到一个隔离的网桥设备。这个设备 *没有* 添加任何物理接口。新设备使用 NAT 和 IP 转发来连接物理网络。不要添加新接口。

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
virbr0          8000.000000000000       yes
```

libvirt 添加 **iptables** 规则，允许连接到 **INPUT**、**FORWARD** 和 **POSTROUTING** 链中的 **virbr0** 设备和 guest 虚拟机的流量。libvirt 然后尝试启用 **ip_forward** 参数。**其他某些应用程序可能会禁用 ip_forward，因此最好的选项是将以下内容添加到 /etc/sysctl.conf 中：**

```
 net.ipv4.ip_forward = 1
```

虚拟机配置

主机配置完成后，客户机虚拟机就可以根据其名称连接到虚拟网络。要将客户端连接到 'default' 虚拟网络，可在 XML 配置文件中使用以下内容（如 /etc/libvirtd/qemu/myguest.xml）用于客户机：

```
<interface type='network'>
  <source network='default'/>
</interface>
```

注意

定义 MAC 地址是可选的。如果您没有定义 MAC 地址，则自动生成 MAC 地址，并将其用作网络使用的网桥设备的 MAC 地址。手动设置 MAC 地址对于在整个环境中保持一致性或便于参考，或者避免非常小的冲突几率。

```
<interface type='network'>
  <source network='default'/>
  <mac address='00:16:3e:1a:b3:4a'/>
</interface>
```

6.2. 禁用 VHOST-NET

vhost-net 模块是用于 virtio 网络的内核级后端，通过将 virtio 数据包处理任务移出用户空间（QEMU 进程）和到内核（ vhost-net 驱动程序））。vhost-net 驱动程序只可用于 virtio 网络接口。如果载入 vhost-net 内核模块，则默认为所有 virtio 接口启用它，但如果特定的工作负载在使用中时，可以在接口配置中禁用它。

特别是，当从主机将 UDP 流量发送到该主机上的虚拟客户机虚拟机时，如果客户机虚拟机处理传入的数据的速度比主机发送的速度慢，则可能会出现性能降级。在这种情况下，启用 vhost-net 会导致 UDP 套接字接收缓冲区更快地出现溢出，这会导致数据包丢失。因此最好禁用 vhost-net，以减慢流量的速度，并提高整体性能。

要禁用 vhost-net，编辑客户机虚拟机 XML 配置文件中的<interface> 子元素，并按如下所示定义网络：

```
<interface type="network">
  ...
  <model type="virtio"/>
  <driver name="qemu"/>
  ...
</interface>
```

将驱动程序名称设置为 qemu 会强制数据包处理到 QEMU 用户空间，从而有效地为该接口禁用 vhost-net。

6.3. 启用 VHOST-NET 零复制

在 Red Hat Enterprise Linux 7 中，默认禁用 vhost-net zero-copy。要永久启用此操作，请在包含以下内容的 /etc/modprobe.d 中添加新的文件 vhost-net.conf ：

```
options vhost_net  experimental_zcopytx=1
```

如果要再次禁用此功能，您可以运行以下命令：

```
modprobe -r vhost_net
```

```
modprobe vhost_net experimental_zcopytx=0
```

第一个命令删除旧文件，第二个命令生成新文件（如上方）并禁用零复制。您可以使用它启用，但更改不具有持久性。

要确认这已生效，请检查 cat /sys/module/vhost_net/parameters/experimental_zcopytx 的输出。它应该显示：

```
$ cat /sys/module/vhost_net/parameters/experimental_zcopytx
0
```

6.4. 网桥网络

网桥网络（也称为网络桥接或虚拟网络交换机）用于将虚拟机网络接口放在与物理接口相同的网络中。网桥需要最少的配置，并使虚拟机显示在现有网络上，从而降低管理开销和网络复杂性。由于网桥包含几个组件和配置变量，它们提供了一个透明的设置，它易于理解并进行故障排除。

可以使用标准 Red Hat Enterprise Linux 工具、 virt-manager 或 libvirt 在虚拟化环境中配置桥接，并请参考以下部分。

但是，即使在虚拟化环境中，也可以使用主机操作系统的网络工具更轻松地创建网桥。有关此网桥创建方法的更多信息，请参阅 Red Hat Enterprise Linux 7 Networking Guide。

6.4.1. 在 Red Hat Enterprise Linux 7 主机上配置桥接网络

桥接网络可以独立于虚拟化管理工具，为 Red Hat Enterprise Linux 主机上的虚拟机进行配置。强烈建议使用这个配置，当虚拟化桥接只是主机的网络接口，或者是主机的管理网络接口时。

有关使用虚拟化工具配置网络桥接的说明，请参阅 Red Hat Enterprise Linux 7 网络指南。

6.4.2. 使用虚拟机管理器桥接网络

这部分提供了使用 virt-manager 从主机机器接口创建到客户机虚拟机的桥接的说明。

注意

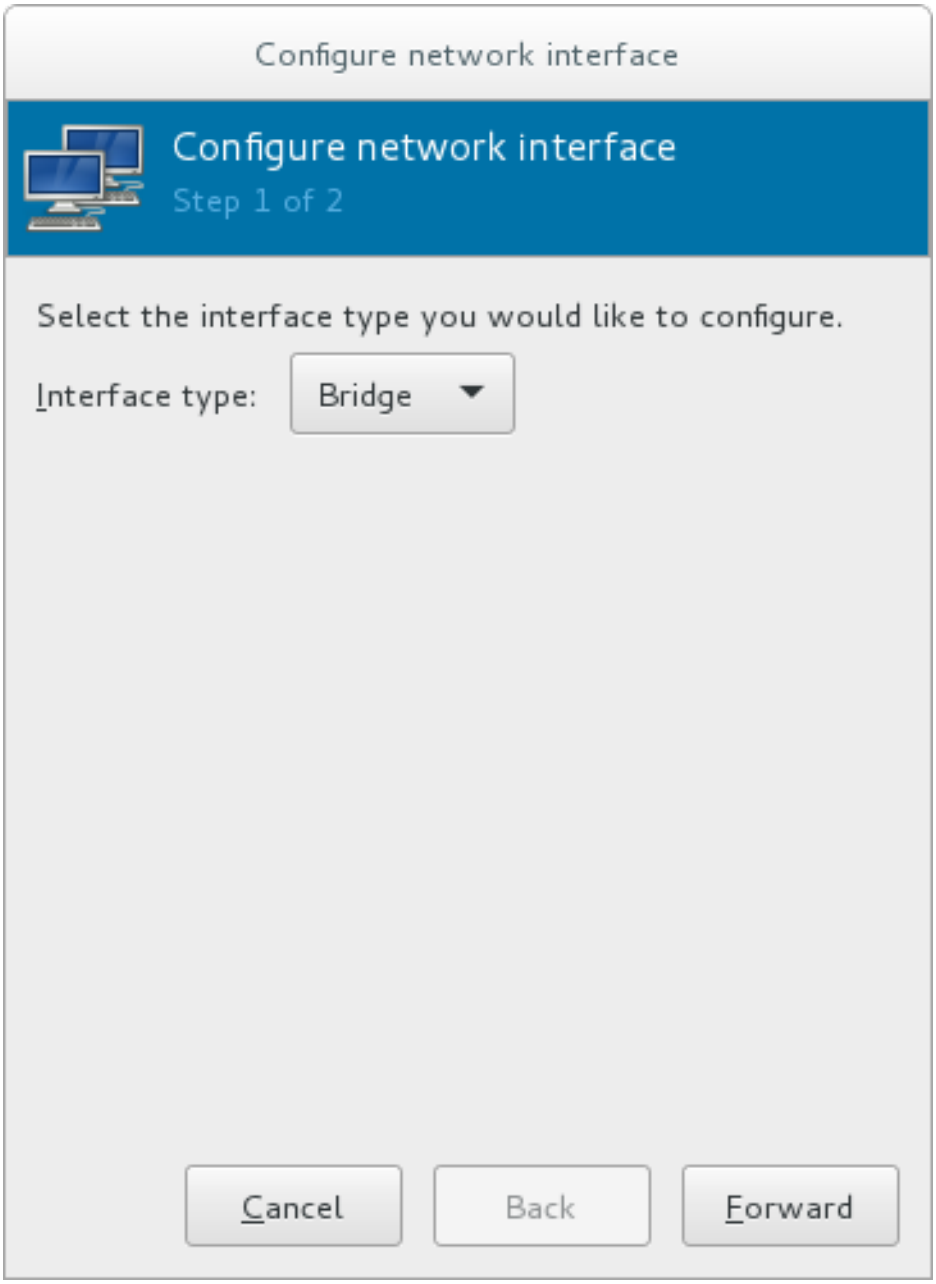
根据您的环境，在 Red Hat Enterprise Linux 7 中使用 libvirt 工具设置桥接可能需要禁用网络管理器（红帽不推荐）。使用 libvirt 创建的网桥还需要运行 libvirtd，以便网桥维护网络连接。

建议在物理 Red Hat Enterprise Linux 主机上配置桥接网络，如 Red Hat Enterprise Linux 7 Networking Guide 所述，在创建网桥后使用 libvirt 向网桥添加虚拟机接口。

过程 6.1. 使用 virt-manager 创建桥接

1. 在 virt-manager 主菜单中，点击 Edit zfcp Connection Details打开 Connection Details 窗口。
2. 点 Network Interfaces选项卡。
3. 单击窗口底部的 +，以配置新的网络接口。
4. 在 Interface 类型 下拉菜单中，选择 Bridge，然后单击 Forward 以继续。

   图 6.1. 添加桥接



5. a. 在 Name 字段中输入网桥名称，如 *br0*。

   b. 从下拉菜单中选择 Start 模式。从以下之一中选择：

   - none - 取消激活网桥
   - ONBOOT - 在下一个客户机虚拟机重新引导时激活桥接
   - 热插拔 - 即使客户机虚拟机正在运行，也会激活桥接

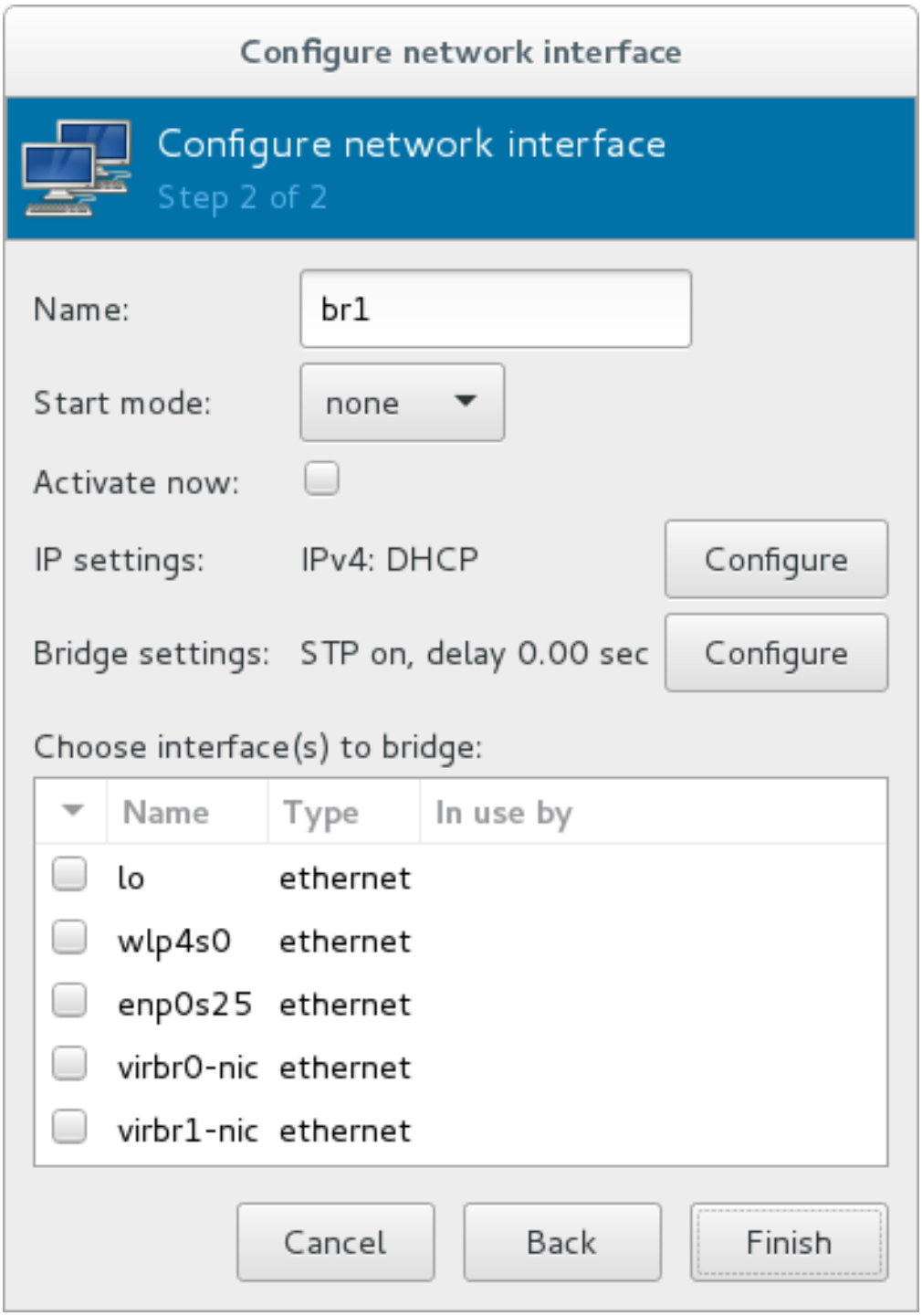
c. 选中 激活现在 复选框以立即激活该网桥。

d. 要配置 IP 设置 或网桥 设置，请单击相应的 配置 按钮。将打开一个单独的窗口，以指定所需的设置。进行必要的更改，完成后单击 确定。

e. 选择要连接到您的虚拟机的物理接口。 如果接口当前被另一个虚拟客户机使用，您将收到一条警告消息。

6. 点 Finish 和向导关闭，把您返回到 Connections 菜单。

图 6.2. 添加桥接



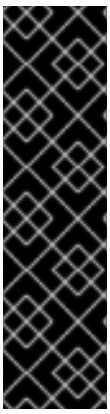
选择要使用的网桥，然后单击 Apply 以退出向导。

要停止接口，点 Stop Interface 键。停止网桥后，若要删除接口，可单击 Delete Interface 密钥。

6.4.3. 使用 libvirt 进行桥接网络

根据您的环境，在 Red Hat Enterprise Linux 7 中使用 libvirt 设置桥接可能需要禁用网络管理器（红帽不推荐）。这还需要 libvirtd 才能运行，以便网桥操作。

建议您在物理 Red Hat Enterprise Linux 主机上配置桥接网络，如 Red Hat Enterprise Linux 7 网络指南所述。



重要

libvirt 现在能够利用新的内核可调项参数来管理主机桥接转发数据库(FDB)条目，从而在桥接多个虚拟机时可能会提高系统性能。在主机的 XML 配置文件中，将网络的 *<bridge>* 元素的 *macTableManager* 属性设置为 'libvirt' ：

```
<bridge name='br0' macTableManager='libvirt'/>
```

这将在所有网桥端口上关闭学习(flood)模式，libvirt 将根据需要添加或删除条目。除了删除学习 MAC 地址正确转发端口的开销外，这也允许内核禁用将网桥连接到网络的物理设备上的混杂模式，从而进一步减少开销。

第 7 章 使用 KVM 进行过量使用

7.1. 简介

KVM 管理程序自动过量使用 CPU 和内存。这意味着，比系统中物理资源相比，可向虚拟机分配更多虚拟化 CPU 和内存。这是因为大多数进程都无法随时访问其分配资源的 100%。

因此，利用不足的虚拟化服务器或桌面可以在更少的主机上运行，这样可以节省大量系统资源，同时降低了电源、冷却和服务器硬件投资的影响。

7.2. 过量使用内存

在 KVM 管理程序中运行的客户机虚拟机没有专用的物理 RAM 块。相反，每个客户机虚拟机作为 Linux 进程的功能，其中的主机物理机器的 Linux 内核仅在请求时分配内存。此外，主机的内存管理器也可以在自己的物理内存和交换空间之间移动客户机虚拟机的内存。

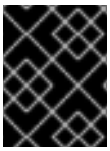
过量使用要求在主机物理机器上分配足够的交换空间，以容纳所有客户机虚拟机，并为主机物理计算机的进程提供足够的内存。作为基本规则，主机物理计算机的操作系统最多需要 4 GB 内存，至少 4 GB 交换空间。有关为 swap 分区确定适当大小的高级说明，请查看红帽知识库。

重要

过量使用不是一般内存问题的理想解决方案。建议使用内存短缺的方法为每个客户机分配较少的内存，向主机添加更多物理内存，或使用 swap 空间。

如果虚拟机经常被交换，则虚拟机运行速度较慢。另外，过量使用会导致系统内存不足 (OOM)，这可能会导致 Linux 内核关闭重要的系统进程。如果您决定过量使用内存，请确定执行足够的测试。请联络红帽支持以获得过量使用的协助。

过量使用不可用于所有虚拟机，但发现在具有少量密集型使用情况的桌面虚拟化设置中工作，或使用 KSM 运行多个相同的虚拟机。有关 KSM 和过量使用的更多信息，请参阅 Red Hat Enterprise Linux 7 虚拟化调整和优化指南。



重要

设备分配不支持内存过量使用。这是因为，当使用 设备分配 时，所有虚拟机内存都必须静态分配才能启用具有分配设备的直接内存访问(DMA)。

7.3. 过量使用虚拟化 CPU

KVM 管理程序支持过量使用虚拟化 CPU(vCPU)。虚拟化 CPU 尽可能超量，因为客户机虚拟机的负载限制允许。在提交 vCPU 时要小心，因为负载接近 100% 可能会导致请求丢失或者无法响应。

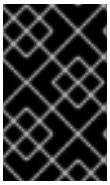
在 Red Hat Enterprise Linux 7 中，可以过量使用带有多个 vCPU 的客户机，称为对称多处理(SMP)虚拟机。但是，当您在虚拟机上运行比物理 CPU 更多的内核时，您可能会遇到性能下降。

例如，具有四个 vCPU 的虚拟机不应在具有双核心处理器的主机上运行，而是在四核主机上运行。由于程序比所需 CPU 时间更少，超过物理处理内核数量的 SMP 虚拟机会导致性能下降。另外，不建议为每个物理处理器内核分配 10 个以上 vCPU。

对于 SMP 客户机，一些处理开销固有。CPU 过量使用可能会增加 SMP 开销，因为使用时间分片将资源分配给客户端，因此客户机内的间 CPU 通信会较慢。随着具有更多 vCPU 数量或更大过量使用比例的客户机，这种开销会增加。

当单个主机有多个虚拟机时，与主机 CPU 数量相比，虚拟化 CPU 的最佳限制是较少的 vCPU。KVM 应以 5 个 vCPU（在 5 台虚拟机）与一个主机上一个物理 CPU 的比例安全支持 100% 下的客户机。KVM 管理程序将在所有虚拟机之间进行切换，确保负载是平衡的。

为了获得最佳性能，红帽建议根据运行每个客户机中的程序而需要的 vCPU 数量只分配客户机。



重要

使用 100% 内存或处理资源的应用程序可能会在过量使用的环境中不稳定。不要在生产环境中过量使用内存或 CPU，且没有广泛的测试，因为 CPU 过量使用比例和 SMP 的数量取决于工作负载。

第 8 章 KVM 客户机计时管理

虚拟化在客户端虚拟机中及时存在诸多挑战。

- 中断无法始终同时交付到所有客户机虚拟机。这是因为虚拟机中的中断不是真正的中断。相反，它们会被主机注入到客户机虚拟机中。
- 主机可以运行另一个虚拟客户机或不同的进程。因此，中断通常需要的精确时间可能并不总是可行。

没有准确的时间的客户机虚拟机可能遇到网络应用程序和进程的问题，因为会话有效、迁移和其他网络活动依赖于时间戳以保持正确。

KVM 通过提供带半虚拟化时钟(kvm-clock)的客户机虚拟机来避免这些问题。但是，在尝试可能受时间不准确（如虚拟机迁移）可能会受到影响的活动之前，仍然务必要测试时间。

重要

为避免上述问题，在主机和客户机虚拟机中应配置网络时间协议(NTP)。在使用红帽企业 Linux 6 及更早版本的虚拟客户机上，NTP 由 ntpd 服务实施。如需更多信息，请参阅 Red Hat Enterprise 6 部署指南。

在使用 Red Hat Enterprise Linux 7 的系统上，NTP 时间同步服务可由 ntpd 或 chronyd 服务提供。请注意，Chrony 在虚拟机上有一些优点。如需更多信息，请参阅《Red Hat Enterprise Linux 7 系统管理员指南》 中的使用 chrony 套件配置 NTP 以及使用 ntpd 的配置 NTP 部分。

客户机虚拟机时间同步的机学

默认情况下，客户端将其时间与虚拟机监控程序同步，如下所示：

- 当客户机系统引导时，客户机会从模拟的 Real Time Clock(RTC)读取时间。
- 启动 NTP 协议时，它会自动同步客户机时钟。之后，在普通 guest 操作期间，NTP 在客户机执行时钟调整。
- 当一个 guest 在暂停或恢复过程后恢复时，应由管理软件（如 virt-manager）发布将客户机时钟同步到指定值的命令。只有在客户机中安装 QEMU 客户机代理 并支持该功能时，此同步才可以正常工作。客户端时钟同步的值是主机时钟值。

恒定的时间戳计数器(TSC)

现代 Intel 和 AMD CPU 提供恒定的时间戳计数器(TSC)。当 CPU 内核本身更改频率时，恒定 TSC 的计数频率不同，例如遵守节能策略。需要具有恒定的 TSC 频率 CPU，以便使用 TSC 作为 KVM 客户机的时钟源。

如果存在 constant_tsc 标志，您的 CPU 有一个恒定的时间戳计数器。运行以下命令，检查您的 CPU 是否具有 constant_tsc 标志：

```
$ cat /proc/cpuinfo | grep constant_tsc
```

如果给出了任何输出，则您的 CPU 具有 constant_tsc 位。如果未指定输出，请遵循以下步骤。

在没有 Constant 时间戳计数器的情况下配置主机

没有恒定的 TSC 频率的系统无法使用 TSC 作为虚拟机的时钟源，需要额外的配置。电源管理功能会干扰准确的时间保留，必须禁用 guest 虚拟机才能准确使用 KVM 保持时间。

重要

这些指令仅适用于 AMD 修订 F CPU。

如果 CPU 缺少 constant_tsc 位，则禁用所有电源管理功能 。每个系统都使用多个计时器来保留时间。主机上 TSC 不稳定，有时是由 cpufreq 更改、深度 C 状态导致的，或者迁移到具有更快 TSC 的主机。C 深度睡眠状态可以停止 TSC。 为防止内核使用深度 C 状态，可将 processor.max_cstate=1 附加到内核启动中。要使这一更改持久，请在 /etc/default/grub文件中编辑 GRUB_CMDLINE_LINUX 键的值。例如，如果要为每个引导启用紧急模式，请按如下方式编辑该条目：

```
GRUB_CMDLINE_LINUX="emergency"
```

请注意，您可以为 GRUB_CMDLINE_LINUX 键指定多个参数，与在 GRUB 2 启动菜单中添加参数类似。

要禁用 cpufreq （只在没有 constant_tsc的主机上），安装 kernel-tools 并启用 cpupower.service (systemctl enable cpupower.service)。如果要在每次客户端虚拟机引导时禁用此服务，更改 /etc/sysconfig/cpupower 中的配置文件，并更改 CPUPOWER_START_OPTS 和 CPUPOWER_STOP_OPTS。有效限制可在 /sys/devices/system/cpu/*cpuid*/cpufreq/scaling_available_governors 文件中找到。有关此软件包或电源管理以及管理者的更多信息，请参阅 *Red Hat Enterprise Linux 7 Power Management Guide*。

8.1. HOST-WIDE TIME SYNC

KVM 客户端中的虚拟网络设备不支持硬件时间戳，这意味着难以同步使用 NTP 或 PTP 等网络协议（如 NTP 或 PTP）的客户机时钟，其准确性要高于微秒的十秒。

当需要更准确的客户端同步时，建议使用 NTP 或 PTP 与硬件时间戳同步主机时钟，并将客户机直接同步到主机。Red Hat Enterprise Linux 7.5 及更新的版本提供虚拟 PTP 硬件时钟(PHC)，它允许客户机与主机同步，并可以通过子微秒的准确性进行同步。

重要

请注意，为了让 PHC 正常工作，主机和客户机都需要使用 RHEL 7.5 或更高版本作为操作系统(OS)。

要启用 PHC 设备，在客户端操作系统中执行以下操作：

1. 将 ptp_kvm 模块设置为在重启后载入。

```
# echo ptp_kvm > /etc/modules-load.d/ptp_kvm.conf
```

2. 将 /dev/ptp0 时钟添加为 chrony 配置的引用：

```
# echo "refclock PHC /dev/ptp0 poll 2" >> /etc/chrony.conf
```

3. 重启 chrony 守护进程：

```
# systemctl restart chronyd
```

4. 要验证 host-guest 时间同步是否已正确配置，请在客户机上使用 chronyc sources 命令。输出应类似于如下：

```
# chronyc sources
210 Number of sources = 1
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
====
#* PHC0                          0   2   377     4    -6ns[   -6ns] +/-  726ns
```

8.2. RED HAT ENTERPRISE LINUX 客户机所需的时间管理参数

对于某些 Red Hat Enterprise Linux 客户机虚拟机，需要正确同步其系统时间需要额外的内核参数。若要设置这些参数，可将这些参数附加到 guest 虚拟机的 /etc/grub2.cfg 文件中的 /kernel 行的末尾。

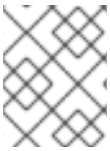
注意

红帽企业 Linux 5.5 及更高版本、红帽企业 Linux 6.0 及更高版本以及红帽企业 Linux 7 使用 kvm-clock 作为默认时钟来源。运行 kvm-clock 避免了对附加内核参数的需求，红帽推荐使用它。

下表列出了 Red Hat Enterprise Linux 的版本以及指定系统中所需的参数。

表 8.1. 内核参数要求

Red Hat Enterprise Linux 版本	其他客户端内核参数
使用 kvm-clock 的 AMD64 和 Intel 64 系统中 7.0 及更新的版本	不需要额外的参数
6.1 及更新的版本在带有 kvm-clock 的 AMD64 和 Intel 64 系统中	不需要额外的参数
带有 kvm-clock 的 AMD64 和 Intel 64 系统上的 6.0	不需要额外的参数
AMD64 和 Intel 64 系统上的 6.0，没有 kvm-clock	notsc lpj=*n*



注意

lpj 参数需要一个数字值，等于客户机虚拟机运行的特定 CPU 的每个*jiffy 值的循环*。如果您不知道这个值，请不要设置 ***lpj*** 参数。

8.3. STEAL TIME ACCOUNTING

steal time 是主机未提供的客户机虚拟机所需的 CPU 时间。当主机在其它位置分配这些资源时（例如，）到另一个客户机时，会发生 steal 时间。

增强时间会在 /proc/stat 的 CPU 时间字段中报告。它由top 和 vmstat 等实用程序自动报告。它显示为 "%st"，或者在"st"列中显示。请注意，它无法关闭。

大量稳定时间表示 CPU 争用，这可以降低客户机性能。为了减轻 CPU 争用，增加客户机的 CPU 优先级或 CPU 配额，或在主机上运行较少的虚拟机。

第 9 章 使用 LIBVIRT 进行网络引导

客户机虚拟机可以在启用 PXE 的情况下引导。PXE 允许客户机虚拟机启动并载入其配置本身。本节演示了使用 libvirt 配置 PXE 客户端的一些基本配置步骤。

本节不涵盖引导镜像或 PXE 服务器的创建。它用于说明如何在私有或桥接网络中配置 libvirt，从而引导启用 PXE 启动的客户机虚拟机。

警告

这些程序仅作为示例提供。在继续操作前，请确保您有足够的备份。

9.1. 准备引导服务器

要执行本章中的步骤，您需要：

- PXE 服务器（DHCP 和 TFTP）- 可以是 libvirt 内部服务器、手动配置 dhcpd 和 tftpd、dnsmasq、一个由 Cobbler 配置的服务器或其他服务器。
- 引导镜像 - 例如，PXELINUX 手动配置或 Cobbler。

9.1.1. 在私有 libvirt 网络中设置 PXE 引导服务器

这个示例使用 *default* 网络。执行以下步骤：

过程 9.1. 配置 PXE 引导服务器

1. 将 PXE 引导镜像和配置放在 /var/lib/tftpboot 中。
2. 使用以下命令：

```
# virsh net-destroy default
# virsh net-edit default
```

3. *编辑默认* 网络的配置文件中的 <ip> 元素，使其包含正确的地址、网络掩码、DHCP 地址范围和引导文件，其中 *BOOT_FILENAME* 代表您用来引导客户机虚拟机的文件名。

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

4. 运行：

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

5. **使用 PXE 引导客户端（请参考 第 9.2 节 “使用 PXE 启动客户机”）。**

9.2. 使用 PXE 启动客户机

本节演示了如何使用 PXE 引导客户机虚拟机。

9.2.1. 使用桥接网络

过程 9.2. 使用 PXE 和桥接网络引导客户端

1. **确保启用桥接功能，以便网络可使用 PXE 引导服务器。**
2. **引导启用了 PXE 启动的客户机虚拟机。您可以使用 virt-install 命令创建启用了 PXE 引导的新虚拟机，如下例所示：**

```
virt-install --pxe --network bridge=breth0 --prompt
```

另外，请确保客户端网络被配置为使用桥接网络，并且 XML 客户机配置文件在 <boot dev='network'/> 元素中有一个 <os> 元素，如下例所示：

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

9.2.2. 使用私有 libvirt 网络

过程 9.3. 使用私有 libvirt 网络

1. **在 libvirt 上配置 PXE 引导，如 第 9.1.1 节 “在私有 libvirt 网络中设置 PXE 引导服务器”所示。**
2. **使用 libvirt 引导客户机虚拟机，并启用 PXE 引导。您可以使用 virt-install 命令使用 PXE 创建/安装新虚拟机：**

```
virt-install --pxe --network network=default --prompt
```

另外，请确保客户端网络被配置为使用您的私有 libvirt 网络，并且 XML 客户机配置文件在 <boot dev='network'/> 元素中有一个 <os> 元素，如下例所示：

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

还要确保客户端虚拟机已连接到私有网络：

```
<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>
```

第 10 章 注册 HYPERVISOR 和虚拟机

Red Hat Enterprise Linux 6 和 7 要求每个客户机虚拟机都映射到特定的管理程序，以确保每个客户机都分配了相同的订阅服务级别。要执行此操作，您需要安装一个订阅代理，该代理会在安装和注册的每个 KVM 管理程序上自动检测所有客户机虚拟机(VM)，进而创建一个位于主机上的映射文件。这个映射文件可确保所有客户机虚拟机都收到以下好处：

- 特定于虚拟系统的订阅可以随时可用，并可应用于所有关联的虚拟机虚拟机。
- 所有可从管理程序继承的订阅优势均可随时可用，并可应用于所有相关的客户机虚拟机。

注意

本章提供的信息仅特定于 Red Hat Enterprise Linux 订阅。如果您有红帽虚拟化订阅或 Red Hat Satellite 订阅，则您还应查阅这些订阅提供的 virt-who 信息。如需红帽订阅管理的更多信息，请参阅《红帽订阅管理 指南》，网址为：

10.1. 在主机物理机器上安装 VIRT-WHO

1. **注册 KVM 管理程序**
   在终端中以 root 用户身份在主机物理计算机上运行 **subscription-manager register [options]** 命令注册 KVM 管理程序。可使用 **# subscription-manager register --help** 菜单获得更多选项。如果您在使用用户名和密码时，请使用 Subscription Manager 应用程序已知的凭证。如果这是您首次订阅，且您没有用户帐户，请联络客户支持。例如，要将虚拟机注册为 'admin' 使用 'secret' 作为密码，您可以发送以下命令：

   ```
   [root@rhel-server ~]# subscription-manager register --username=admin --password=secret --auto-attach
   ```

2. **安装 virt-who 软件包**
   在主机物理机器上运行以下命令来安装 virt-who 软件包：

   ```
   # yum install virt-who
   ```

3. **创建 virt-who 配置文件**
   对于每个管理程序，在 **/etc/virt-who.d/** 目录中添加一个配置文件。文件必须至少包含以下片断：

   ```
   [libvirt]
   type=libvirt
   ```

   有关配置 virt-who 的详情请参考 第 10.1.1 节 “配置 virt-who”。

4. **启动 virt-who 服务**
   在主机物理机器上运行以下命令启动 virt-who 服务：

   ```
   # systemctl start virt-who.service
   # systemctl enable virt-who.service
   ```

5. **确认 virt-who 服务正在收到客户机信息**
   此时，virt-who 服务将从主机收集一组域。检查主机物理计算机上的 **/var/log/rhsm/rhsm.log** 文件，以确认该文件是否包含客户端虚拟机的列表。例如：

```
2015-05-28 12:33:31,424 DEBUG: Libvirt domains found: [{'guestId': '58d59128-cfbb-4f2c-93de-230307db2ce0', 'attributes': {'active': 0, 'virtWhoType': 'libvirt', 'hypervisorType': 'QEMU'}, 'state': 5}]
```

过程 10.1. 在客户门户网站中管理订阅

1. 订阅虚拟机监控程序
   由于虚拟机将获得与虚拟机监控程序相同的订阅好处，因此管理程序具有有效的订阅，且订阅可供虚拟机使用。

   a. 登录到客户门户网站
      在红帽客户门户网站中提供 您的红帽帐户凭证 以进行登录。

   b. 点系统链接
      前往 My Subscriptions 接口的 Systems 部分。

   c. 选择虚拟机监控程序
      在系统页面中，存在所有订阅的系统表。单击虚拟机监控程序的名称（如 localhost.localdomain）。在打开的详情页面中，点 Attach a subscription 并选择所有列出的订阅。单击 Attach Selected。这会将主机的物理订阅附加到管理程序，以便 guest 可从订阅中受益。

2. 订阅客户机虚拟机 - 首次使用
   此步骤适用于拥有新订阅且之前从未订阅客户机虚拟机的用户。如果您要添加虚拟机，请跳过此步骤。要在运行 virt-who 服务的机器上使用分配给管理程序配置文件的订阅，请通过在客户机虚拟机上的终端运行以下命令自动订阅。

   ```
   [root@virt-who ~]# subscription-manager attach --auto
   ```

3. 订阅额外的客户机虚拟机
   如果您首次订阅了虚拟机，请跳过此步骤。如果您要添加额外的虚拟机，请注意，运行此命令不一定会在客户机虚拟机中重新附加相同的订阅。这是因为，删除所有订阅后允许自动附加来解决给定客户机虚拟机需要什么操作，可能会导致与之前使用的不同订阅。这对您的系统可能没有任何影响，但应该了解它。如果您使用手动附加程序附加虚拟机（如下所述），则需要手动附加这些虚拟机，因为自动附加将无法正常工作。使用以下命令首先删除旧客户机的订阅，然后使用自动附加订阅将订阅附加到所有客户机。在客户机虚拟机中运行这些命令。

   ```
   [root@virt-who ~]# subscription-manager remove --all
   [root@virt-who ~]# subscription-manager attach --auto
   ```

4. 确认订阅已附加
   通过在客户机虚拟机中运行以下命令来确认订阅已附加到管理程序中：

   ```
   [root@virt-who ~]# subscription-manager list --consumed
   ```

   将显示类似于如下内容的输出。请注意订阅详情。它应该说"订阅为"当前"。

   ```
   [root@virt-who ~]# subscription-manager list --consumed
   +-------------------------------------------+
     Consumed Subscriptions
   +-------------------------------------------+
   Subscription Name: Awesome OS with unlimited virtual guests
   Provides:   Awesome OS Server Bits
   ```

```
SKU:    awesomeos-virt-unlimited
Contract:   0
Account:   ######### Your account number #####
Serial:   ######### Your serial number ######
Pool ID:   XYZ123
Provides Management:  No
Active:   True
Quantity Used:   1
Service Level:
Service Type:
Status Details:  Subscription is current
Subscription Type:
Starts:   01/01/2015
Ends:    12/31/2015
System Type:   Virtual
```

???
: The ID for the subscription to attach to the system is displayed here. You will need this ID if you need to attach the subscription manually.

???
: Indicates if your subscription is current. If your subscription is not current, an error message appears. One example is Guest has not been reported on any host and is using a temporary unmapped guest subscription. In this case the guest needs to be subscribed. In other cases, use the information as indicated in 第 10.5.2 节 “我有订阅状态错误，我有什么作用？”.

5. **注册其他客户端**
   当您在 hypervisor 中安装新客户机虚拟机时，您必须注册新虚拟机并使用附加到虚拟机监控程序的订阅，方法是在客户机虚拟机中运行以下命令：

```
# subscription-manager register
# subscription-manager attach --auto
# subscription-manager list --consumed
```

10.1.1. 配置 virt-who

virt-who 服务使用以下文件进行配置：

- **/etc/virt-who.conf - 包含常规配置信息，包括检查连接的虚拟机监控程序的间隔。**
- **/etc/virt-who.d/*hypervisor_name*.conf - 包含特定管理程序的配置信息。**

提供了一个基于 web 的向导来生成虚拟机监控程序配置文件和 virt-who.conf 所需的代码片段。要运行向导，请浏览客户门户上的 Red Hat Virtualization Agent(virt-who)Configuration Helper。

在向导的第二页面中，选择以下选项：

- **您的 virt-who 报告在哪里？订阅资产管理器**
- **管理程序类型:libvirt**

按照向导完成配置。如果正确执行配置，则 virt-who 将自动为指定管理程序上的现有和将来的客户机提供所选订阅。

有关虚拟机监控程序配置文件的详情，请查看 **virt-who-config** man page。

10.2. 注册新客户机虚拟机

如果需要在已经注册并运行的主机上创建新的 guest 虚拟机，则必须运行 virt-who 服务。这样可保证 virt-who 服务将 guest 映射到管理程序，以便系统正确注册为虚拟系统。要注册虚拟机，请输入以下命令：

```
[root@virt-server ~]# subscription-manager register --username=admin --password=secret --auto-attach
```

10.3. 删除客户机虚拟机条目

如果客户机虚拟机正在运行，取消注册系统，在客户机上以 root 用户身份在终端窗口中运行以下命令：

```
[root@virt-guest ~]# subscription-manager unregister
```

但是，如果删除了系统，则虚拟服务无法判断是否删除或暂停该服务。在这种情况下，您必须使用以下步骤从服务器端手动删除系统：

1. 登录到 Subscription Manager
   订阅管理器位于 红帽客户门户网站 中。点击屏幕顶部的登录图标，使用您的用户名和密码登录到客户门户网站。

2. 点订阅标签页
   点 **Subscriptions** 选项卡。

3. 点系统链接
   向下滚动页面，再单击 **Systems** 链接。

4. 删除系统
   要删除系统配置文件，请在表中找到指定系统的配置文件，选中其名称旁边的复选框，然后单击 **Delete**。

10.4. 手动安装 VIRT-WHO

本节将介绍如何手动附加由管理程序提供的订阅。

过程 10.2. 如何手动附加订阅

1. 列出订阅信息并查找池 ID
   首先，您需要列出虚拟类型的可用订阅。使用以下命令：

```
[root@server1 ~]# subscription-manager list --avail --match-installed | grep 'Virtual' -B12
Subscription Name: Red Hat Enterprise Linux ES (Basic for Virtualization)
Provides:          Red Hat Beta
                   Oracle Java (for RHEL Server)
                   Red Hat Enterprise Linux Server
SKU:               -------
Pool ID:           XYZ123
Available:         40
Suggested:         1
Service Level:     Basic
```

```
Service Type:      L1-L3
Multi-Entitlement: No
Ends:              01/02/2017
System Type:       Virtual
```

请注意显示的池 ID。在下一步中复制此 ID。

2. 使用池 ID 附加订阅
使用您在上一步中复制的池 ID 运行 attach 命令。将池 ID *XYZ123* 替换为您检索到的池 ID。使用以下命令：

```
[root@server1 ~]# subscription-manager attach --pool=XYZ123

Successfully attached a subscription for: Red Hat Enterprise Linux ES (Basic for
Virtualization)
```

10.5. 对 VIRT-WHO 进行故障排除

10.5.1. 为什么管理程序状态红色？

场景：在服务器端，您将在没有订阅的系统管理程序上部署客户机。之后 24 小时，管理程序将其状态显示为红色。要纠正这种情况，您必须获得该管理程序的订阅。或者，通过订阅将 guest 永久迁移到 hypervisor。

10.5.2. 我有订阅状态错误，我有什么作用？

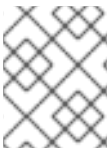
场景：显示以下出错信息：

- 系统没有正确订阅
- Status unknown
- 通过 virt-who（主机/虚拟机映射）将客户机绑定到管理程序.

要查找错误的原因，请打开名为 rhsm.log 的 virt-who 日志文件，该文件位于 /var/log/rhsm/ 目录中。

第 11 章 使用 QEMU 客户机代理和 SPICE 代理增强虚拟化

可以部署 Red Hat Enterprise Linux 中的代理，如 QEMU 客户机代理和 SPICE 代理，以帮助虚拟化工具在您的系统中运行更加最佳。本章介绍了这些代理。



注意

要进一步优化和调整主机和客户机性能，请参阅 Red Hat Enterprise Linux 7 虚拟化调整和优化指南。

11.1. QEMU 客户机代理

QEMU 客户机代理在客户机中运行，并允许主机使用 libvirt 向客户机操作系统发出命令，帮助有 freezing 和 thawing 文件系统等功能。然后，客户端操作系统会异步响应这些命令。在 Red Hat Enterprise Linux 7 中，QEMU 客户机代理软件包 qemu-guest-agent 默认安装。

这部分论述了客户端代理可用的 libvirt 命令和选项。

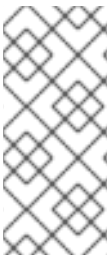
重要

请注意，只有在受信任的客户机运行时，才安全地依赖 QEMU 客户机代理。不受信任的访客可能会恶意忽略或滥用客户机代理协议，虽然存在内置保护机制以防止对主机上的服务攻击进行拒绝，主机需要客户合作才能正常运行。

请注意，QEMU 客户机代理可用于在客户机运行时启用和禁用虚拟 CPU(vCPU)，从而在不使用热插拔和热拔功能的情况下调整 vCPU 数量。如需更多信息，请参阅 第 20.36.6 节 “配置虚拟 CPU 数”。

11.1.1. 设置 QEMU 客户机代理和主机之间的通信

主机机器通过主机和客户机机器之间的 VirtIO 串行连接与 QEMU 客户机代理通信。VirtIO 串行通道通过字符设备驱动程序（通常是 Unix 套接字）连接到主机，客户机则侦听此串行通道。



注意

qemu-guest-agent 不会检测主机是否侦听 VirtIO 串行频道。但是，由于此频道的当前用途是侦听主机到客户机事件，因此通过写入没有监听器的频道来出现问题的客户机虚拟机的概率非常低。另外，qemu-guest-agent 协议包含同步标记，它允许主机物理机器在发出命令时强制客户端虚拟机同步，并且 libvirt 已使用这些标记，因此客户机虚拟机能够安全地丢弃任何早期待处理的未发响应。

11.1.1.1. 在 Linux 客户机上配置 QEMU 客户机代理

可在正在运行或关闭虚拟机上配置 QEMU 客户机代理。如果在正在运行的虚拟客户机中配置了，客户机将立即开始使用客户机代理。如果 guest 关闭，则将在下次引导时启用 QEMU 客户机代理。

virsh 或 virt-manager 可用于配置客户机和 QEMU 客户机代理之间的通信。下面的说明描述了如何在 Linux 客户端中配置 QEMU 客户机代理。

过程 11.1. 在关闭 Linux 客户机中使用 virsh 设置客户机代理与主机之间的通信

1. 关闭虚拟机
   在配置 QEMU 客户机代理前，请确定虚拟机（本例中为 *rhel7*）被关闭：

```
# virsh shutdown rhel7
```

2. **在客户机 XML 配置中添加 QEMU 客户机代理频道**
   编辑客户机的 XML 文件以添加 QEMU 客户机代理详情：

   ```
   # virsh edit rhel7
   ```

   在客户机的 XML 文件中添加以下内容并保存更改：

   ```
   <channel type='unix'>
     <target type='virtio' name='org.qemu.guest_agent.0'/>
   </channel>
   ```

3. **启动虚拟机**

   ```
   # virsh start rhel7
   ```

4. **在客户端中安装 QEMU 客户机代理**
   安装 QEMU 客户机代理（如果尚未在客户机虚拟机中安装）：

   ```
   # yum install qemu-guest-agent
   ```

5. **在客户机中启动 QEMU 客户机代理**
   在客户机中启动 QEMU 客户机代理服务：

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

另外，可以使用以下步骤在运行的客户机中配置 QEMU 客户机代理：

过程 11.2. 在正在运行的 Linux 客户机中设置客户机代理和主机之间的通信

1. **为 QEMU 客户机代理创建 XML 文件**

   ```
   # cat agent.xml
   <channel type='unix'>
     <target type='virtio' name='org.qemu.guest_agent.0'/>
   </channel>
   ```

2. **将 QEMU 客户机代理附加到虚拟机**
   使用以下命令，将 QEMU 客户机代理附加到正在运行的虚拟机（本例中为 *rhel7*）：

   ```
   # virsh attach-device rhel7 agent.xml
   ```

3. **在客户端中安装 QEMU 客户机代理**
   安装 QEMU 客户机代理（如果尚未在客户机虚拟机中安装）：

   ```
   # yum install qemu-guest-agent
   ```

4. **在客户机中启动 QEMU 客户机代理**
   在客户机中启动 QEMU 客户机代理服务：

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

过程 11.3. 使用 virt-manager在 QEMU 客户机代理和主机间设置通信

1. **关闭虚拟机**
   在配置 QEMU 客户机代理前，确保虚拟机已关闭。

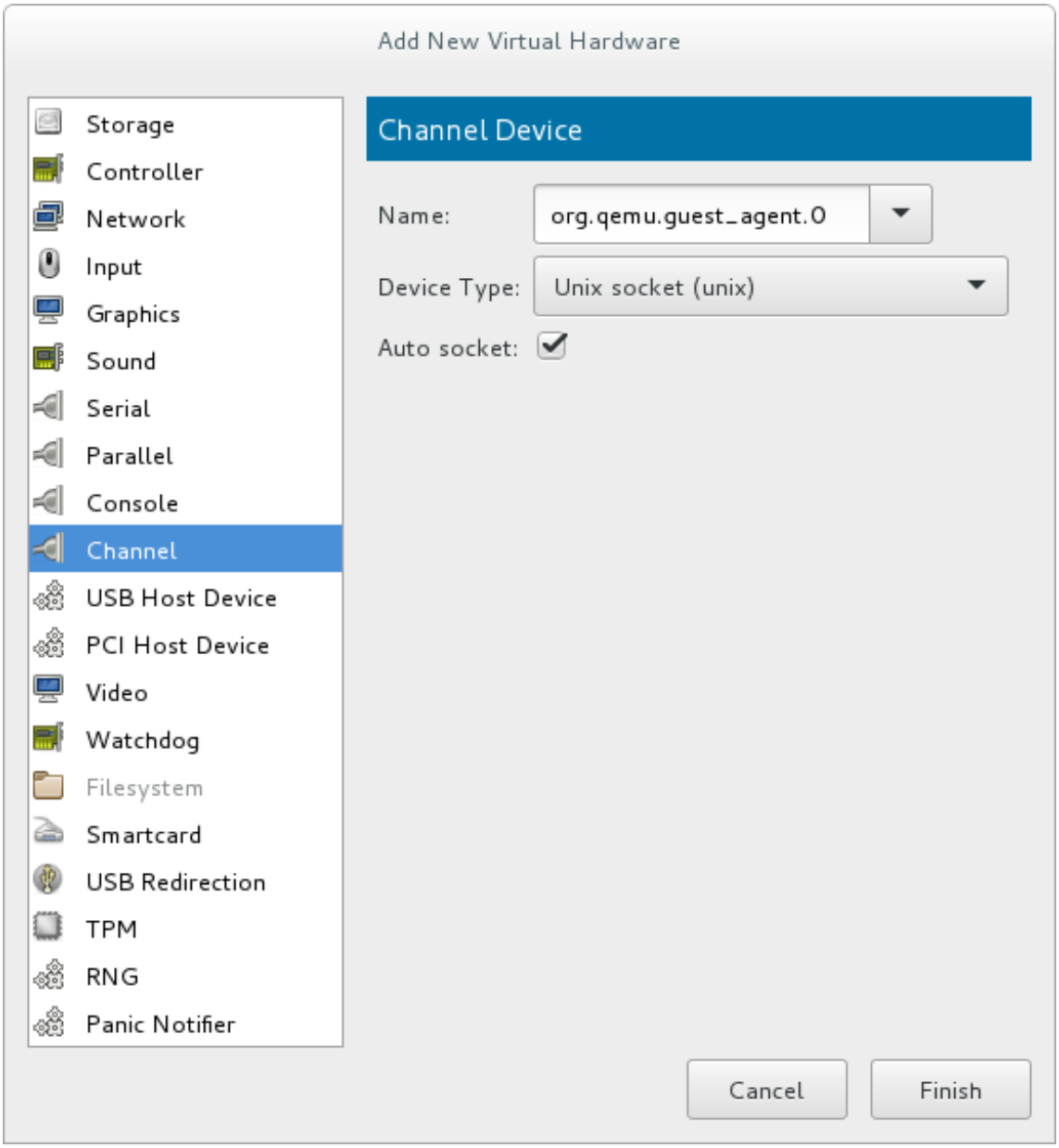
   要关闭虚拟机，请从 Virtual Machine Manager中的虚拟机列表中选择，然后单击菜单栏中的 light 开关图标。

2. **将 QEMU 客户机代理频道添加到客户端**
   单击 guest 窗口顶部的 lightbulb 图标，打开虚拟机的硬件详细信息。

   点击 Add Hardware 按钮，打开 Add New Virtual Hardware 窗口，然后选择 Channel。

   从 Name 下拉列表中选择 QEMU 客户机代理并点 Finish:

   图 11.1. 选择 QEMU 客户机代理频道设备



3. 启动虚拟机

要启动虚拟机，请从 Virtual Machine Manager 中的虚拟机列表中选择，然后点击  。

4. 在客户端中安装 QEMU 客户机代理
使用 virt-manager 打开客户端，如果尚未在客户机虚拟机中安装 QEMU 客户机代理：

```
# yum install qemu-guest-agent
```

5. 在客户机中启动 QEMU 客户机代理
在客户机中启动 QEMU 客户机代理服务：

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

QEMU 客户机代理现在在 *rhel7* 虚拟机上配置。

11.2. 使用带有 LIBVIRT 的 QEMU 客户机代理

安装 QEMU 客户机代理可让各种 libvirt 命令变得更加强大。客户机代理增强了以下 virsh 命令：

- **virsh shutdown --mode=agent** - 此关机方法比 virsh shutdown --mode=acpi 可靠，因为 QEMU 客户机代理使用的 virsh shutdown 可以保证以干净状态关闭合作虚拟机。如果代理不存在，libvirt 必须依赖于注入 ACPI 关闭事件，但有些客户机会忽略该事件，因此不会关闭。

  可与 virsh reboot 相同的语法一起使用。

- **virsh snapshot-create --quiesce** - 允许客户机在创建快照之前将其 I/O 清空到稳定状态，这允许使用快照而无需执行 fsck 或丢失部分数据库事务。通过提供客户机合作，客户机代理可以实现高级别的磁盘内容稳定性。

- **virsh domfsfreeze 和 virsh domfsthaw** - 隔离 guest 文件系统.

- **virsh domfstrim** - 结构 guest 以修剪其文件系统。

- **virsh domtime** - Queries 或 set the guest 的时钟.

- **virsh setvcpus --guest** - Instructs the guest 使 CPU 离线。

- **virsh domifaddr --source 代理** - 通过客户机代理查询客户机操作系统的 IP 地址。

- **virsh domfsinfo** - 显示正在运行的客户端中挂载文件系统的列表。

- **virsh set-user-password** - 在 guest 中设置用户帐户的密码。

11.2.1. 创建客户机磁盘备份

libvirt 可以与 qemu-guest-agent 通信，以确保客户机虚拟机文件系统的快照在内部一致，并可根据需求使用。虚拟客户机系统管理员可编写和安装特定于应用程序的 freeze/thaw hook 脚本。在释放文件系统前，qemu-guest-agent 调用主 hook 脚本（包括在qemu-guest-agent 软件包中）。freezing 进程临时取消激活所有客户机虚拟机应用程序。

快照过程由以下步骤组成：

- 文件系统应用/数据库将工作缓冲区清空到虚拟磁盘，并停止接受客户端连接

- 应用程序将其数据文件变为一致的状态

- 主 hook 脚本返回
- qemu-guest-agent 冻结文件系统，管理堆栈会拍摄快照
- 已确认快照
- 文件系统功能恢复

Thawing 以相反的顺序发生。

要创建客户端文件系统的快照，请运行 **virsh snapshot-create --quiesce --disk-only** 命令（此外，运行 **virsh snapshot-create-as *guest_name* --quiesce --disk-only**，在 第 20.39.2 节 “为当前客户机虚拟机创建快照”中进一步介绍）。

注意

应用程序特定的 hook 脚本可能需要各种 SELinux 权限才能正确运行，就像需要连接到套接字才能与数据库通信时所执行的操作。通常，应该针对这些目的开发并安装本地 SELinux 策略。访问文件系统节点应该开箱即用，在标记为 **/etc/qemu-ga/fsfreeze-hook.d/** 的表行中列出的 **restorecon -FvvR** 命令后。表 11.1 “QEMU 客户机代理软件包内容”

qemu-guest-agent 二进制 RPM 包括以下文件：

表 11.1. QEMU 客户机代理软件包内容

文件名	描述
/usr/lib/systemd/system/qemu-guest-agent.service	用于 QEMU 客户机代理的服务控制脚本（启动/停止）。
/etc/sysconfig/qemu-ga	QEMU 客户机代理的配置文件，因为它由 **/usr/lib/systemd/system/qemu-guest-agent.service** 控制脚本读取。设置记录在 文件中，shell 脚本注释。
/usr/bin/qemu-ga	QEMU 客户机代理二进制文件。
/etc/qemu-ga	hook 脚本的根目录。
/etc/qemu-ga/fsfreeze-hook	主 hook 脚本。这里不需要修改。
/etc/qemu-ga/fsfreeze-hook.d	单个、特定于应用程序的 hook 脚本的目录。客户机系统管理员应手动将 hook 脚本复制到这个目录中，确保它们的正确文件模式位，然后在该目录中运行 **restorecon -FvvR**。
/usr/share/qemu-kvm/qemu-ga/	带有示例脚本的目录（例如，仅用于使用）。此处包含的脚本不会执行。

主 hook 脚本 **/etc/qemu-ga/fsfreeze-hook** 可记录自己的消息，以及特定于应用程序的标准输出和错误消息： **/var/log/qemu-ga/fsfreeze-hook.log**。如需更多信息，请参阅 libvirt 上游网站。

11.3. SPICE 代理

SPICE 代理有助于通过帮助将客户机操作系统与 SPICE 客户端集成，从而更加顺利地运行图形应用程序。

例如，当在 virt-manager 中重新定义窗口大小时，SPICE 代理允许自动 X 会话解析调整客户端解析。SPICE 代理也支持在主机和客户机之间进行复制和粘贴，并防止鼠标光标放大。

有关 SPICE 代理功能的系统特定信息，请参阅 spice-vdagent 软件包的 README 文件。

11.3.1. 设置 SPICE 代理和主机之间的通信

可以在正在运行或关闭虚拟机上配置 SPICE 代理。如果在正在运行的虚拟客户机中配置了，客户机将立即开始使用客户机代理。如果 guest 关闭，则将在下次引导时启用 SPICE 代理。

virsh 或 virt-manager 可用于配置客户机和 SPICE 代理之间的通信。下面的说明描述了如何在 Linux 虚拟机上配置 SPICE 代理。

过程 11.4. 在 Linux 客户机中使用 virsh 设置客户机代理与主机之间的通信

1. 关闭虚拟机
   在配置 SPICE 代理前，请确定虚拟机（本例中为 *rhel7*）被关闭：

   ```
   # virsh shutdown rhel7
   ```

2. 在客户机 XML 配置中添加 SPICE 代理频道
   编辑客户端的 XML 文件以添加 SPICE 代理详情：

   ```
   # virsh edit rhel7
   ```

   在客户机的 XML 文件中添加以下内容并保存更改：

   ```
   <channel type='spicevmc'>
     <target type='virtio' name='com.redhat.spice.0'/>
   </channel>
   ```

3. 启动虚拟机

   ```
   # virsh start rhel7
   ```

4. 在客户端中安装 SPICE 代理
   如果尚未在客户机虚拟机中安装 SPICE 代理：

   ```
   # yum install spice-vdagent
   ```

5. 在客户机中启动 SPICE 代理
   在客户机中启动 SPICE 代理服务：

   ```
   # systemctl start spice-vdagent
   ```

或者，可使用以下步骤在运行的客户机中配置 SPICE 代理：

过程 11.5. 在正在运行的 Linux 客户机中设置 SPICE 代理与主机之间的通信

1. 为 SPICE 代理创建 XML 文件

```
# cat agent.xml
<channel type='spicevmc'>
  <target type='virtio' name='com.redhat.spice.0'/>
</channel>
```

2. **将 SPICE 代理附加到虚拟机**
   用这个命令将 SPICE 代理附加到正在运行的虚拟机（本例中为 *rhel7*）：

```
# virsh attach-device rhel7 agent.xml
```

3. **在客户端中安装 SPICE 代理**
   如果尚未在客户机虚拟机中安装 SPICE 代理：

```
# yum install spice-vdagent
```

4. **在客户机中启动 SPICE 代理**
   在客户机中启动 SPICE 代理服务：

```
# systemctl start spice-vdagent
```

过程 11.6. 使用 virt-manager在 SPICE 代理和主机间设置通信

1. **关闭虚拟机**
   在配置 SPICE 代理前，确保虚拟机已关闭。

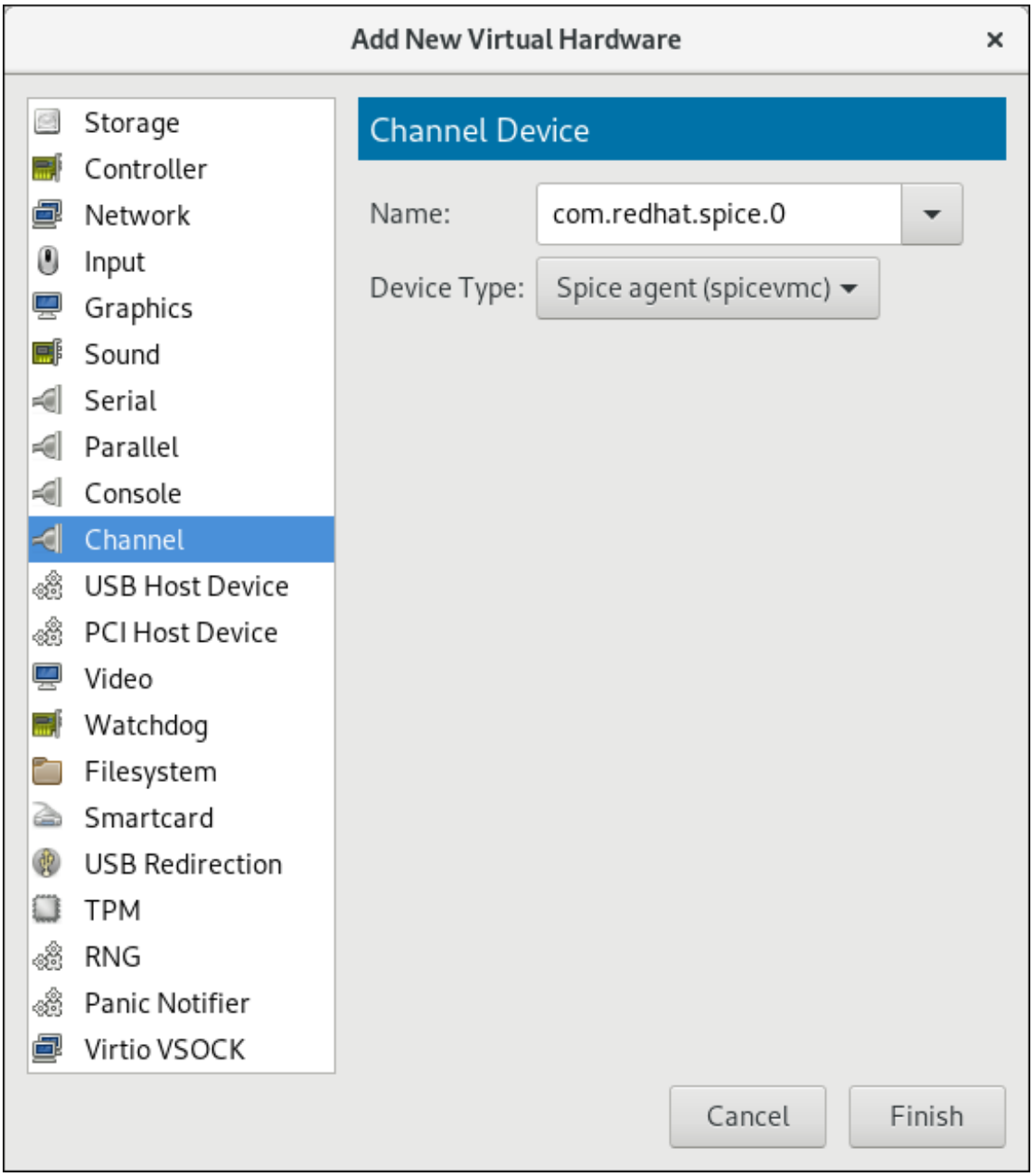
   要关闭虚拟机，请从 Virtual Machine Manager中的虚拟机列表中选择，然后单击菜单栏中的 light 开关图标。

2. **将 SPICE 代理频道添加到客户端**
   单击 guest 窗口顶部的 lightbulb 图标，打开虚拟机的硬件详细信息。

   点击 **Add Hardware** 按钮，打开 **Add New Virtual Hardware** 窗口，然后选择 **Channel**。

   从 **名称** 下拉列表中选择 SPICE 代理，编辑频道地址，然后点 **Finish**:

图 11.2. 选择 SPICE 代理频道设备



3. 启动虚拟机

   要启动虚拟机，请从 Virtual Machine Manager 中的虚拟机列表中选择，然后点击  。

4. 在客户端中安装 SPICE 代理
   使用 virt-manager 打开客户机，并在客户机虚拟机上尚未安装 SPICE 代理：

   ```
   # yum install spice-vdagent
   ```

5. 在客户机中启动 SPICE 代理
   在客户机中启动 SPICE 代理服务：

   ```
   # systemctl start spice-vdagent
   ```

SPICE 代理现在已在 *rhel7* 虚拟机上配置。

第 12 章 嵌套虚拟化

12.1. 概述

从 Red Hat Enterprise Linux 7.5 开始，*嵌套虚拟化* 是 KVM 客户机虚拟机的技术预览。https://access.redhat.com/support/offerings/techpreview/借助此功能，在物理主机（*级别0 或 L0*）*上运行的客户机虚拟机（也称为级别1或 L1*）可以充当虚拟机监控程序，然后创建自己的 guest 虚拟机(*L2*)。

嵌套虚拟化在各种场景中很有用，例如在受限环境中调试虚拟机监控程序，并在有限的物理资源上测试大型虚拟部署。但请注意，在生产环境中不支持嵌套虚拟化，主要用于开发和测试。

嵌套虚拟化依赖于主机虚拟化扩展来实现功能，它不应该使用 QEMU Tiny Code Generator(TCG)模拟（在 Red Hat Enterprise Linux 不支持）虚拟环境中运行客户机。

12.2. 设置

按照以下步骤启用、配置和启动使用嵌套虚拟化：

1. 启用 ：此功能默认为禁用。要启用它，请在 L0 主机物理机上使用以下步骤。

   对于 Intel：

   a. 检查主机系统中是否有嵌套虚拟化可用。

      ```
      $ cat /sys/module/kvm_intel/parameters/nested
      ```

      如果这个命令返回 Y 或 1，则代表启用了该功能。

      如果命令返回 0 或 N，请使用 *ii* 和 *iii* 的步骤。

   b. 卸载 kvm_intel 模块：

      ```
      # modprobe -r kvm_intel
      ```

   c. 激活嵌套功能：

      ```
      # modprobe kvm_intel nested=1
      ```

   d. 现在，嵌套功能只有在下一次重启 L0 主机时才会启用。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
      ```

   AMD：

   a. 检查系统中是否有嵌套虚拟化可用：

      ```
      $ cat /sys/module/kvm_amd/parameters/nested
      ```

      如果这个命令返回 Y 或 1，则代表启用了该功能。

如果命令返回 0 或 N，请使用 *ii* 和 *iii* 的步骤。

b. **卸载 kvm_amd 模块**

```
# modprobe -r kvm_amd
```

c. **激活嵌套功能**

```
# modprobe kvm_amd nested=1
```

d. **现在，嵌套功能只有在下一次重启 L0 主机时才会启用。要永久启用它，请在 /etc/modprobe.d/kvm.conf 文件中添加以下行：**

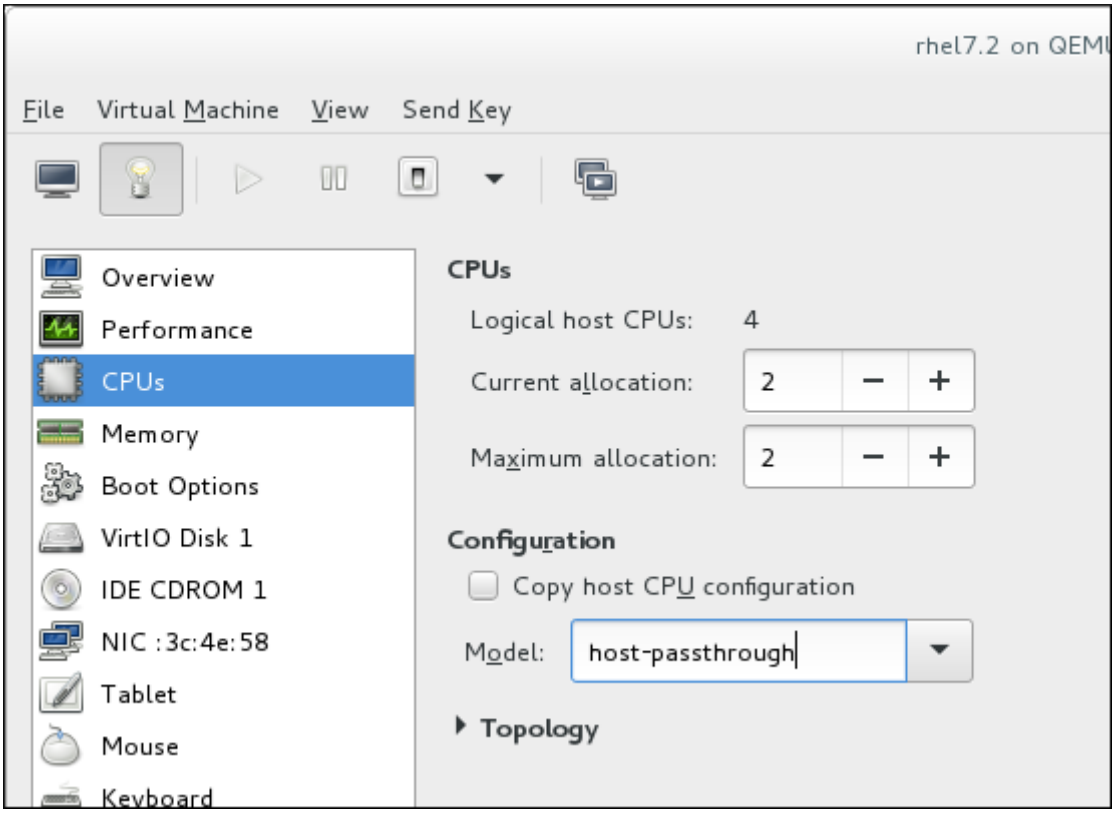
```
options kvm_amd nested=1
```

2. **使用以下方法之一为嵌套虚拟化配置 L1 虚拟机：**

virt-manager

a. **打开所需 guest 的 GUI 并单击 *Show Virtual Hardware Details* 图标。**

b. 选择 *Processor* **菜单，并在 *Configuration* 部分中键入 host-passthrough** in the*Model* **字段（不要使用下拉菜单），然后单击" *应用* "。**



[D]

域 XML

在客户机的域 XML 文件中添加以下行：

```
<cpu mode='host-passthrough'/>
```

如果客户机的 XML 配置文件已经包含 <cpu> 元素，重写它。

3. 要开始使用 嵌套虚拟化，请在 L1 客户端中安装 L2 客户机。要做到这一点，请按照安装 L1 客户端时相同的步骤 - 请参阅 第 3 章 *创建虚拟机* 了解更多信息。

12.3. 限制和限制

- 强烈建议在 L0 主机和 L1 客户端中运行 Red Hat Enterprise Linux 7.2 或更高版本。L2 虚拟机可以包含红帽支持的任何 guest 系统。
- 不支持迁移 L1 或 L2 客户机。
- 不支持将 L2 虚拟机用作虚拟机监控程序，并创建 L3 虚拟机。
- L1 管理程序不能使用主机上的全部功能。例如，L1 管理程序无法使用 IOMMU/VT-d 或 APICv。
- 要使用嵌套虚拟化，主机 CPU 必须具有所需的功能标记。要确定 L0 和 L1 管理程序是否已正确设置，请在 L0 和 L1 上使用 **cat /proc/cpuinfo** 命令，并确保在两个 hypervisor 上为相应的 CPU 列出了以下标记：
  - 对于 Intel - vmx (Hardware Virtualization)和 ept （扩展页表）
  - AMD - svm （等同于 vmx）和 npt （等同于 ept）

部分 II. 管理

这部分涵盖了与虚拟机管理相关的主题，并解释了虚拟化功能（如虚拟网络、存储和 PCI 分配）的工作方式。这部分还提供有关使用 qemu-img、virt-manager 和 virsh 工具管理的设备和客户机虚拟机管理的说明。

第 13 章 为虚拟机管理存储

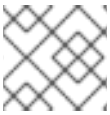
本章提供有关虚拟机存储的信息。虚拟存储从分配给虚拟机连接的物理存储抽象化。存储使用半虚拟化或模拟块设备驱动程序附加到虚拟机。

13.1. 存储概念

存储池是 客户端虚拟机供客户机使用而设置的存储数量。存储池被分成一个 *存储卷*。每个存储卷都以客户机总线上的块设备形式分配给客户机虚拟机。

存储池和卷使用 libvirt 管理。使用 libvirt 的远程协议，可以管理客户机虚拟机生命周期的所有方面，以及客户机虚拟机所需的资源的配置。这些操作可以在远程主机上执行。因此，使用 libvirt 等管理应用程序可以允许用户执行为客户机虚拟机配置主机物理机器所需的所有任务。这包括分配资源、运行客户机虚拟机、将其关闭和取消分配资源，而无需 shell 访问或任何其他控制频道。

libvirt API 可用于查询存储池中的卷列表，或者获取存储池中容量、分配和可用存储的信息。可能会查询存储池中的存储卷以获取分配和容量等信息，稀疏卷可能会有所不同。

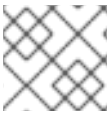


注意

有关稀疏卷的更多信息，请参阅 虚拟化入门指南。

对于支持它的存储池，可以使用 libvirt API 来创建、克隆、调整大小和删除存储卷。API 也可用于将数据上传到存储卷，从存储卷下载数据，或者从存储卷擦除数据。

启动存储池后，可以使用存储池名称和存储卷名称而非域 XML 中卷的主机路径，将存储卷分配给客户机。



注意

有关域 XML 的详情，请参考 第 23 章 *操作域 XML*。

存储池可以被停止（销毁）。这会删除数据的抽象，但保持数据不受影响。

例如，使用 **mount -t nfs nfs.example.com:/path/to/share /path/to/data** 的 NFS 服务器。负责管理存储管理员在虚拟化主机上定义 NFS 存储池，以描述导出的服务器路径和客户端目标路径。这将允许 libvirt 在启动 libvirt 时或者根据需要在 libvirt 运行时自动执行挂载。NFS 服务器导出目录的文件在 NFS 存储池中被列为存储卷。

当存储卷添加到客户端时，管理员不需要为卷添加目标路径。他只需要根据名称添加存储池和存储卷。因此，如果目标客户端路径改变，它不会影响虚拟机。

启动存储池时，libvirt 将共享挂载到指定目录中，就像系统管理员登录并执行 **nfs.example.com:/path/to/share /vmdata** 一样。如果存储池被配置为 autostart，则 libvirt 可确保将 NFS 共享磁盘挂载到 libvirt 启动时指定的目录中。

启动存储池后，NFS 共享磁盘中的文件报告为存储卷，且可以使用 libvirt API 查询存储卷的路径。然后可将存储卷的路径复制到客户机虚拟机 XML 定义的部分，该部分描述了客户机虚拟机块设备的源存储。对于 NFS，使用 libvirt API 的应用程序可以在存储池中创建和删除存储卷（NFS 共享中的文件）到池大小的限制（共享的存储容量）。

并非所有存储池类型都支持创建和删除卷。停止存储池(pool-destroy)会撤销启动操作，在这种情况下，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。详情请查看 **man virsh**。

过程 13.1. 创建并分配存储

此流程提供了对为虚拟机创建和分配存储所需的步骤的高级了解。

1. 创建存储池
   从可用存储介质创建一个或多个存储池。如需更多信息，请参阅 第 13.2 节 “使用存储池”。
2. 创建存储卷
   从可用存储池创建一个或多个存储卷。如需更多信息，请参阅 第 13.3 节 “使用存储卷”。
3. 为虚拟机分配存储设备。
   将从存储卷中提取的一个或多个存储设备分配给客户机虚拟机。如需更多信息，请参阅 第 13.3.6 节 “在客户机中添加存储设备”。

13.2. 使用存储池

这部分提供有关将存储池与虚拟机搭配使用的信息。它提供了 概念的信息，以及使用 virsh 命令和虚拟机管理器 创建、配置 和删除 存储池的详细说明。

13.2.1. 存储池概念

存储池是由 libvirt 管理的文件、目录或存储设备，用于为虚拟机提供存储。存储池被分成存储虚拟机镜像或作为额外存储附加到虚拟机的存储卷。多个虚拟机可以共享同一存储池，从而更好地分配存储资源。

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。本地存储池对不需要迁移或大量虚拟机的部署非常有用。本地存储池可能不适用于许多生产环境，因为它们无法用于实时迁移。

联网的（共享）存储池

联网的存储池包括使用标准协议通过网络共享的存储设备。当使用 virt-manager 在主机间迁移虚拟机时，需要联网的存储，但在迁移 virsh 时是可选的。

有关迁移虚拟机的更多信息，请参阅 第 15 章 *KVM 迁移*。

以下是 Red Hat Enterprise Linux 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
- glusterfs 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 SCSI 设备的基于 vHBA 的存储池

以下是 Red Hat Enterprise Linux 不支持的 libvirt 存储池类型列表：

- 基于多路径的存储池
- 基于 RBD 的存储池
- 基于 Sheepdog 的存储池
- 基于 Vstorage 的存储池
- 基于 ZFS 的存储池

注意

一些不受支持的存储池类型会出现在 Virtual Machine Manager 界面中。但是，它们不应被使用。

13.2.2. 创建存储池

这部分介绍了使用 virsh 和 Virtual Machine Manager 创建存储池的一般说明。使用 virsh 可让您指定所有参数，而使用虚拟机管理器 提供了创建更简单的存储池的图形方法。

13.2.2.1. 使用 virsh 创建存储池

注意

本节演示了创建基于分区的存储池作为示例。

过程 13.2. 使用 virsh 创建存储池

1. 阅读建议并确保满足所有先决条件
   对于某些存储池，本指南建议您使用某些实践。另外，某些类型的存储池有一些先决条件。要查看建议和先决条件，如果有，请参阅 第 13.2.3 节 “存储池特定”。

2. 定义存储池
   存储池可以是持久性或临时的。主机系统重启后，持久性存储池会保留下来。临时存储池仅在主机重启前存在。

   执行以下操作之一：

   - 使用 XML 文件定义存储池。

     a.创建包含新设备所需的存储池信息的临时 XML 文件。

     XML 文件必须包含根据存储池类型的特定字段。如需更多信息，请参阅 第 13.2.3 节 “存储池特定”。

     下面显示了一个存储池定义 XML 文件示例。在本例中，该文件被保存到 ~/guest_images.xml

     ```
     <pool type='fs'>
       <name>guest_images_fs</name>
       <source>
         <device path='/dev/sdc1'/>
       </source>
     ```

```
  <target>
    <path>/guest_images</path>
  </target>
</pool>
```

b.使用 **virsh pool-define** 命令创建持久性存储池或 **virsh pool-create** 命令，以创建并启动临时存储池。

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

或

```
# virsh pool-create ~/guest_images.xml
Pool created from guest_images_fs
```

c.删除步骤中创建的 XML 文件。

- 使用 **virsh pool-define-as** 命令创建持久性存储池或 **virsh pool-create-as** 命令，以创建临时存储池。

  以下示例从 **/guest_images** 目录中创建一个永久、基于文件系统的存储池，映射到 **/dev/sdc1**。

```
# virsh pool-define-as guest_images_fs fs - - /dev/sdc1 - "/guest_images"
Pool guest_images_fs defined
```

或

```
# virsh pool-create-as guest_images_fs fs - - /dev/sdc1 - "/guest_images"
Pool guest_images_fs created
```

注意

当使用 **virsh** 接口时，命令中的选项名称是可选的。如果没有使用选项名称，请将横线用于不需要指定的字段。

3. 验证是否已创建池
   使用 **virsh pool-list --all** 列出所有现有存储池。

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images_fs      inactive   no
```

4. 定义存储池目标路径
   使用 **virsh pool-build** 命令为预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，并定义数据的格式。然后使用 **virsh pool-list** 命令，确保列出存储池。

```
# 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 ..
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images_fs      inactive   no
```

注意

构建目标路径只适用于基于磁盘、基于文件系统以及逻辑存储池。如果 libvirt 检测到源存储设备的数据格式与所选存储池类型的不同，构建将失败，除非指定了 覆盖 选项。

5. 启动存储池
   使用 virsh pool-start 命令准备源设备以供使用。

   执行的操作取决于存储池类型。例如，对于基于文件系统的存储池，virsh pool-start 命令会挂载文件系统。对于基于 LVM 的存储池，virsh pool-start 命令可激活卷组 usng thevgchange 命令。

   然后，使用 virsh pool-list 命令来确保 存储池处于活动状态。

```
# virsh pool-start guest_images_fs
Pool guest_images_fs started
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
guest_images_fs      active     no
```

注意

virsh pool-start 命令只适用于持久性存储池。临时存储池创建时会自动启动。

6. 启用自动启动（可选）
   默认情况下，使用 virsh 定义的存储池不会被设置为在每次 libvirtd 启动时自动启动。您可以使用 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

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

   现在，每次 libvirtd 启动时都会自动启动存储池。

7. 验证存储池

验证存储池是否已正确创建，报告的大小与预期相同，状态报告为 **running**。验证文件系统的目标路径中存在"lost+found"目录，表示挂载了该设备。

```
# virsh pool-info guest_images_fs
Name:           guest_images_fs
UUID:           c7466869-e82a-a66c-2187-dc9d6f0877d0
State:          running
Persistent:     yes
Autostart:      yes
Capacity:       458.39 GB
Allocation:     197.91 MB
Available:      458.20 GB
# mount | grep /guest_images
/dev/sdc1 on /guest_images type ext4 (rw)
# ls -la /guest_images
total 24
drwxr-xr-x.  3 root root  4096 May 31 19:47 .
dr-xr-xr-x. 25 root root  4096 May 31 19:38 ..
drwx------.  2 root root 16384 May 31 14:18 lost+found
```

13.2.2.2. 使用虚拟机管理器创建存储池

注意

本节演示了创建基于磁盘的存储池作为示例。

过程 13.3. 使用虚拟机管理器创建存储池

1. 准备创建存储池的中
   这将因不同类型的存储池而异。详情请查看 第 13.2.3 节 “存储池特定”。

   在本例中，您可能需要使用 *GUID 分区表* 重新标记磁盘。

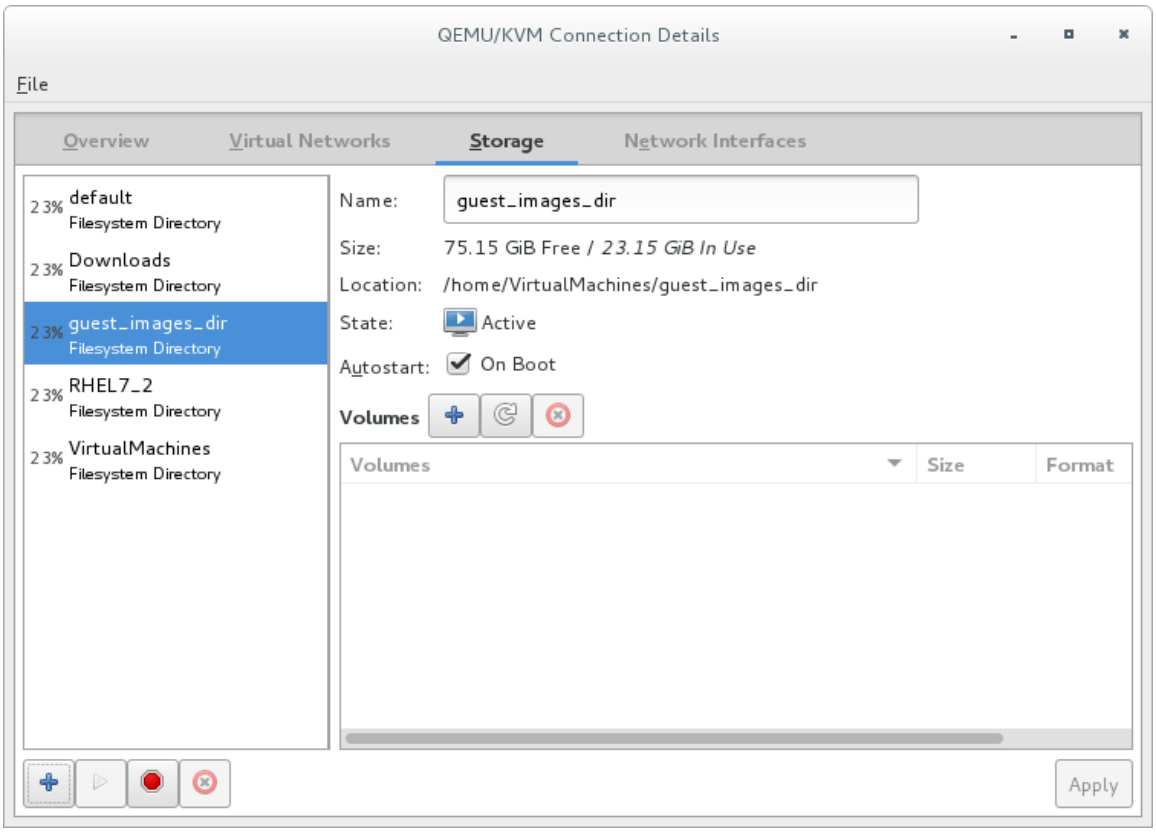
2. 打开存储设置

   a. 在 Virtual Machine Manager 中，选择您要配置的主机连接。

      打开 **Edit** 菜单，再选择 **Connection Details**。

   b. 点 **Connection Details** 窗口中的 **Storage** 选项卡。

图 13.1. Storage 标签页



3. 创建新存储池

注意

使用 虚拟机管理器，您只能创建持久性存储池。临时存储池只能使用 virsh 创建。

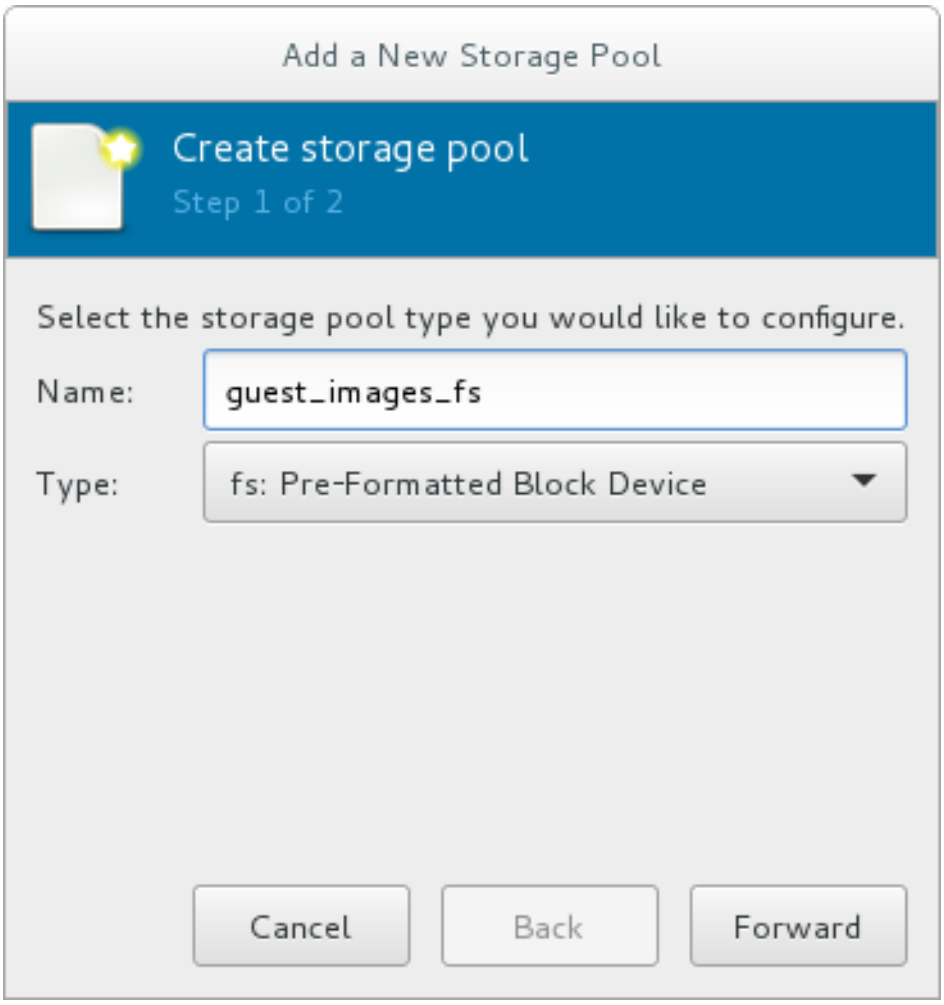
a. 添加新存储池（第 1 部分）

点击窗口底部的  按钮。此时会出现 Add a New Storage Pool 向导。

输入存储池的 Name。这个示例使用名称 *guest_images_fs*。

从 Type 下拉列表中，选择要创建的存储池类型。这个示例使用 fs：预先填充块设备。

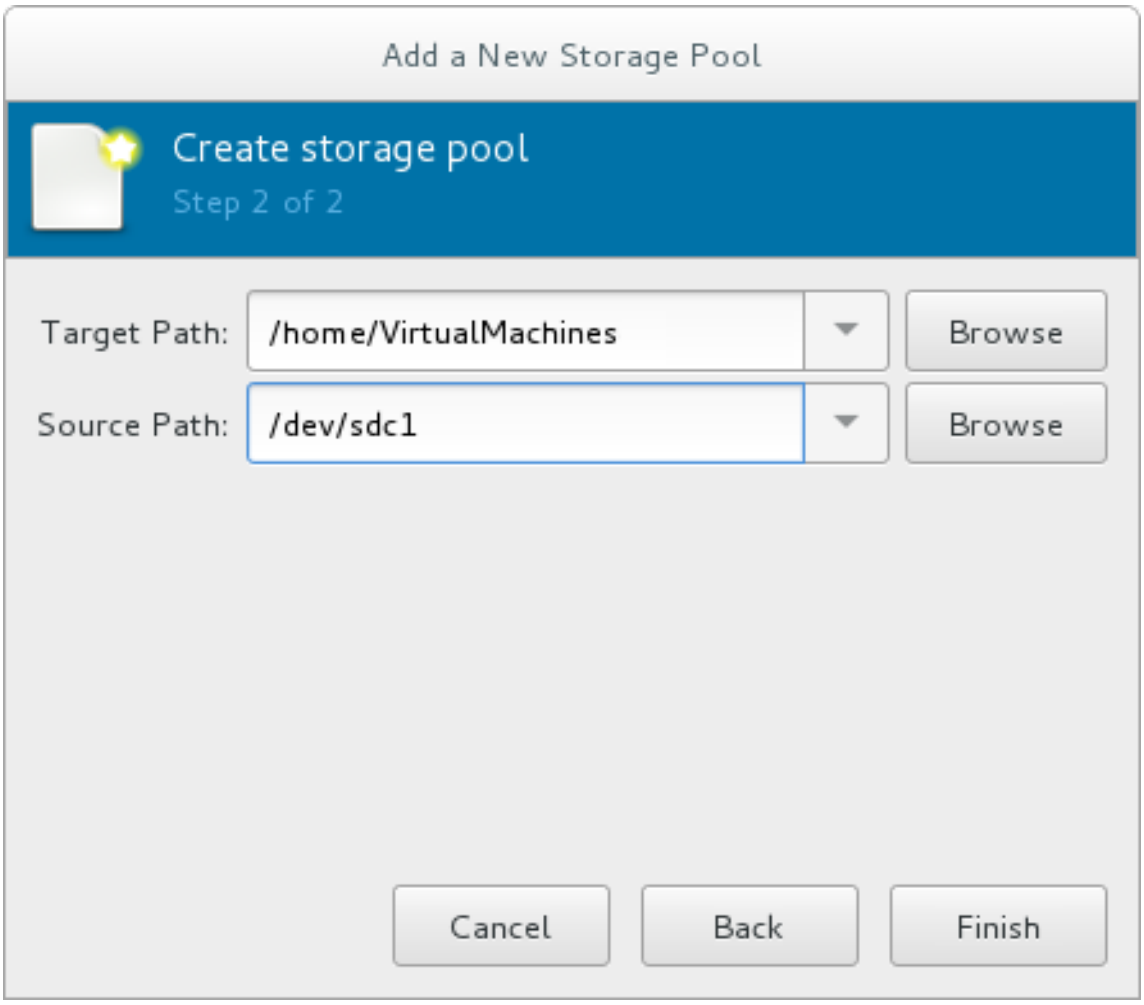
图 13.2. 存储池名称和类型



单击 "转发 "按钮继续。

b. **添加新池（第 2 部分）**

图 13.3. 存储池路径



使用相关参数配置存储池。有关每种存储池的参数的详情，请参考 第 13.2.3 节 “存储池特定”。

对于某些类型的存储池，对话框中会出现一个 Build Pool 复选框。如果要从存储构建存储池，请选中 Build Pool 复选框。

验证详细信息，然后单击 Finish 按钮以创建存储池。

13.2.3. 存储池特定

本节提供了与每种类型的存储池相关的信息，包括先决条件、参数和其他信息。它包括以下主题：

- 第 13.2.3.1 节 “基于目录的存储池”
- 第 13.2.3.2 节 “基于磁盘的存储池”
- 第 13.2.3.3 节 “基于文件系统的存储池”
- 第 13.2.3.4 节 “基于 glusterfs 的存储池”
- 第 13.2.3.5 节 “基于 iSCSI 的存储池”
- 第 13.2.3.6 节 “基于 LVM 的存储池”
- 第 13.2.3.7 节 “基于 NFS 的存储池”

- 第 13.2.3.8 节 “使用 SCSI 设备使用基于 vHBA 的存储池”

13.2.3.1. 基于目录的存储池

参数
下表提供了 XML 文件、virsh pool-define-as 命令和 Virtual Machine Manager 应用程序所需的参数列表，用于创建基于目录的存储池。

表 13.1. 基于目录的存储池参数

描述	XML	pool-define-as	虚拟机管理器
存储池的类型	<pool type='dir'>	[type] 目录	dir：文件系统目录
存储池的名称	<name>*name*</name>	[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>	目标 *path_to_pool*	目标路径

如果您使用 virsh 创建存储池，请通过验证池是否已创建来继续。

示例
以下是基于 /guest_images 目录的存储池的 XML 文件示例：

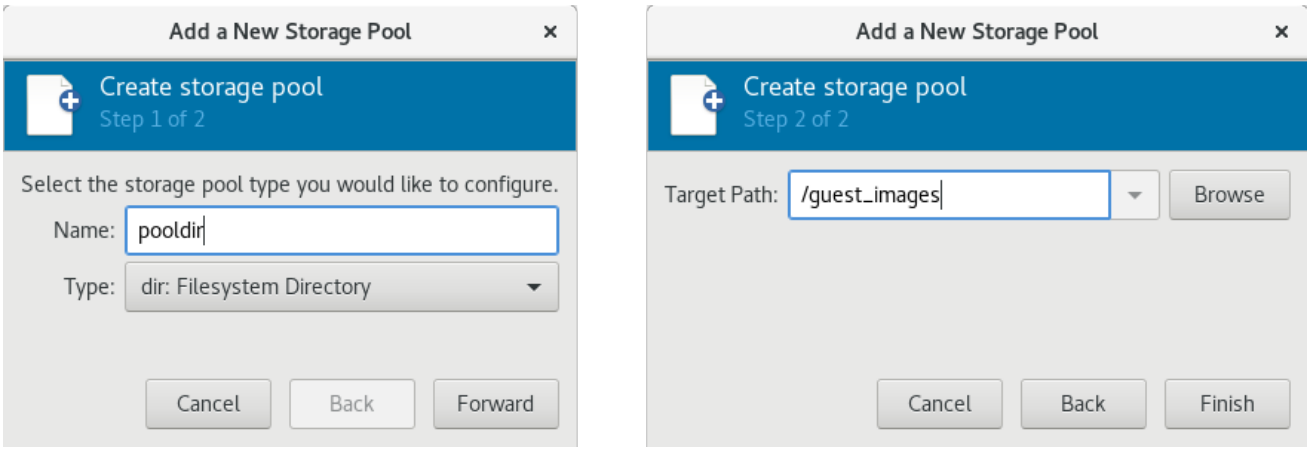
```
<pool type='dir'>
  <name>dirpool</name>
  <target>
    <path>/guest_images</path>
  </target>
</pool>
```

以下是基于 /guest_images 目录创建存储池的命令示例：

```
# virsh pool-define-as dirpool dir --target "/guest_images"
Pool FS_directory defined
```

下图显示了 虚拟机管理器 Add a New Storage Pool 对话框的示例，它用于创建基于/guest_images 目录的存储池：

图 13.4. 添加新基于目录的存储池示例



13.2.3.2. 基于磁盘的存储池

建议
在创建基于磁盘的存储池前请注意以下几点：

- **根据所使用的 libvirt 版本，将磁盘专用于存储池，可以重新格式化并擦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。**
- **不应向 guest 授予对整个磁盘或块设备（例如 /dev/sdb）的写入权限。使用分区（例如 /dev/sdb1）或 LVM 卷。**

  如果您将整个块设备传递给客户端，客户机可能会对它进行分区，或者创建自己的 LVM 组。这可能导致主机物理机器检测到这些分区或 LVM 组并导致错误。

先决条件

注意

只有在您不运行 virsh pool-build 命令时才需要本节中的步骤。

在主机磁盘上可创建基于磁盘的存储池前，必须使用 *GUID 分区表* (GPT)磁盘标签重新标记磁盘。GPT 磁盘标签允许在每个设备上创建最多 128 个分区。

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

重新标记磁盘后，继续使用 定义存储池来创建存储池。

参数
下表提供了 XML 文件、virsh pool-define-as 命令和 Virtual Machine Manager 应用程序所需的参数列表，用于创建基于磁盘的存储池。

表 13.2. 基于磁盘的存储池参数

描述	XML	pool-define-as	虚拟机管理器
存储池的类型	<pool type='disk'>	[type] 磁盘	磁盘：物理磁盘设备
存储池的名称	<name>*name*</name>	[name] *name*	Name
指定存储设备的路径。例如： /dev/sdb	<source> <device path=/dev/*sdb*/> <source>	source-dev *path_to_disk*	源路径
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>	目标 *path_to_pool*	目标路径

如果您使用 virsh 创建存储池， 请继续 **定义存储池。**

示例
以下是基于磁盘存储池的 XML 文件示例：

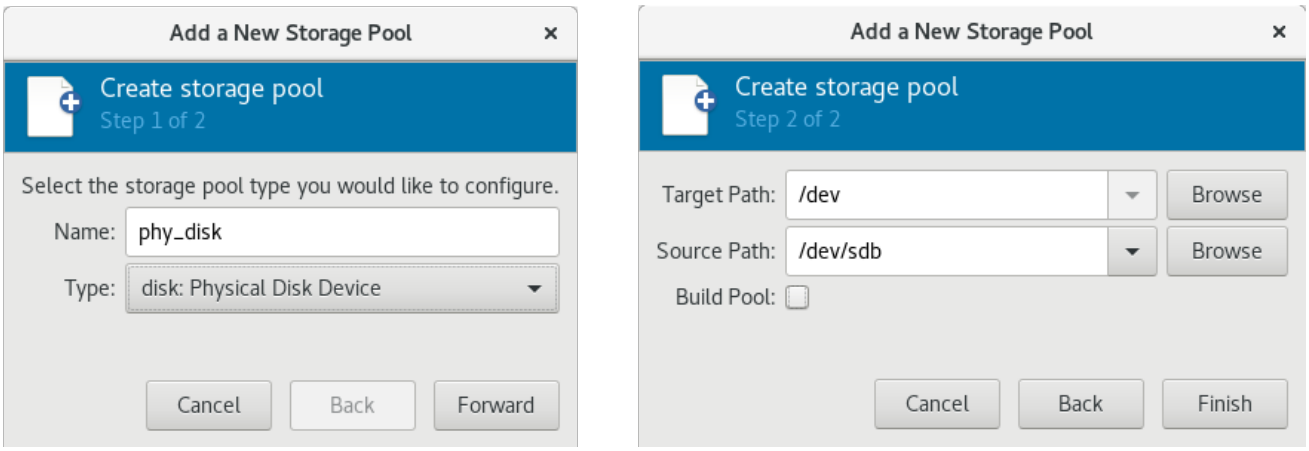
```
<pool type='disk'>
  <name>phy_disk</name>
  <source>
    <device path='/dev/sdb'/>
    <format type='gpt'/>
  </source>
  <target>
    <path>/dev</path>
  </target>
</pool>
```

以下是创建基于磁盘存储池的命令示例：

```
# virsh pool-define-as phy_disk disk --source-format=gpt --source-dev=/dev/sdb --target /dev
Pool phy_disk defined
```

下图显示了虚拟机 XML 配置虚拟机管理器 来添加 New Storage Pool 对话框的例子，用于创建基于磁盘的存储池：

图 13.5. 添加新的基于磁盘的存储池示例



13.2.3.3. 基于文件系统的存储池

建议
不要使用本节中的步骤分配整个磁盘作为存储池（如 /dev/sdb）。不应为 guest 授予对整个磁盘或块设备的写入权限。这个方法只应该被用来将分区（例如 /dev/sdb1）分配给存储池。

先决条件

注意

只有在您不运行 virsh pool-build 命令时才需要。

要从分区创建存储池，请将文件系统挂载到 ext4。

```
# mkfs.ext4 /dev/sdc1
```

格式化文件系统后，继续使用 定义存储池来继续创建存储池。

参数
下表提供了 XML 文件、virsh pool-define-as 命令和 Virtual Machine Manager 应用程序所需的参数列表，用于从分区创建基于文件系统的存储池。

表 13.3. 基于文件系统的存储池参数

描述	XML	pool-define-as	虚拟机管理器
存储池的类型	<pool type='fs'>	[type] fs	fs:预先填充块设备
存储池的名称	<name>*name*</name>	[name] *name*	Name
指定分区的路径。例如：/dev/sdc1	<source> <device path='*source_path*' />	[source] *path_to_partition*	源路径

描述	XML	pool-define-as	虚拟机管理器
文件系统类型，例如 ext4	<format type='*fs_type*' /></source>	[source 格式] *FS-format*	不适用
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>	[target] *path_to_pool*	目标路径

如果您使用 virsh 创建存储池， 请继续**验证存储池是否已创建。**

示例
以下是基于文件系统的存储池的 XML 文件示例：

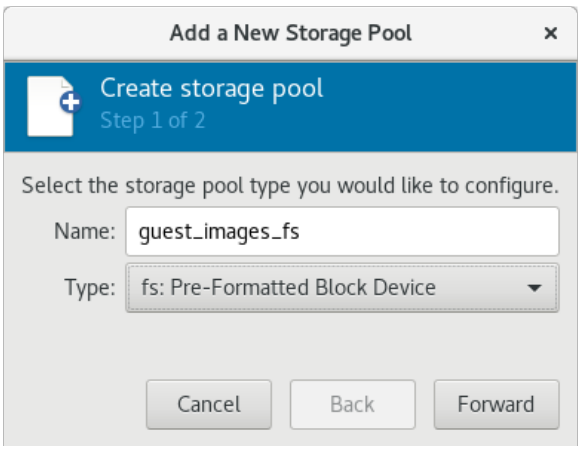
```
<pool type='fs'>
  <name>guest_images_fs</name>
  <source>
    <device path='/dev/sdc1'/>
    <format type='auto'/>
  </source>
  <target>
    <path>/guest_images</path>
  </target>
</pool>
```

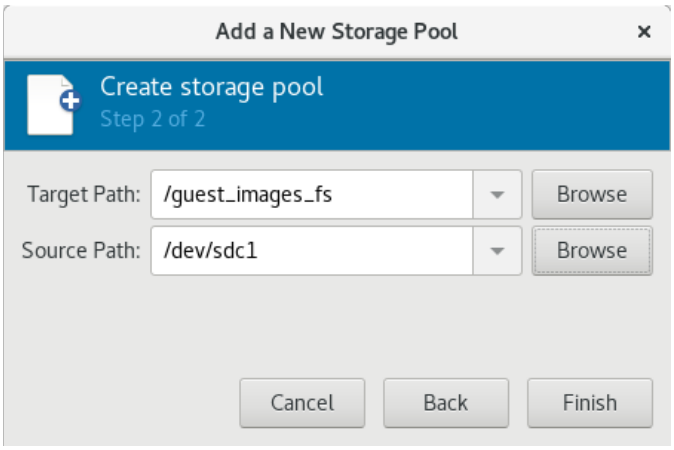
以下是创建基于分区的存储池的命令示例：

```
# virsh pool-define-as guest_images_fs fs --source-dev /dev/sdc1 --target /guest_images
Pool guest_images_fs defined
```

下图显示了虚拟机 XML 配置虚拟机管理器 来添加 New Storage Pool 对话框的例子，用于创建基于文件系统的存储池：

图 13.6. 添加一个基于文件系统的存储池示例





13.2.3.4. 基于 glusterfs 的存储池

建议
GlusterFS 是一个用户空间文件系统，在用户空间(FUSE)中使用文件系统。

先决条件
在在主机上创建基于 GlusterFS 的存储池前，必须准备好 Gluster 服务器。

过程 13.4. 准备 Gluster 服务器

1. **使用以下命令列出 Gluster 服务器的状态来获取 Gluster 服务器的 IP 地址：**

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

Task Status of Volume gluster-vol1
------------------------------------------------------------------------------
There are no active volume tasks
```

2. **如果没有安装，请安装 glusterfs-fuse 软件包。**

3. **如果没有启用，请启用 virt_use_fusefs 布尔值。检查是否已启用。**

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

确保安装并启用所需的软件包后，继续创建存储池，继续创建存储池，并 定义了存储池。

参数
下表提供了 XML 文件、virsh pool-define-as 命令和 Virtual Machine Manager 应用程序所需的参数列表，用于创建基于 GlusterFS 的存储池。

表 13.4. glusterfs 基于存储池参数

描述	XML	pool-define-as	虚拟机管理器
存储池的类型	<pool type='gluster'>	[type] gluster	Gluster：Gluster Filesystem
存储池的名称	<name>*name*</name>	[name] *name*	Name
Gluster 服务器的主机名或 IP 地址	<source> <hostname='*hostname*' />	source-host *hostname*	主机名

描述	XML	pool-define-as	虚拟机管理器
Gluster 服务器的名称	<name='*Gluster-name*' />	source-name *Gluster-name*	源名称
用于存储池的 Gluster 服务器上的路径	<dir path='*Gluster-path*' /> </source>	source-path *Gluster-path*	源路径

如果您使用 virsh 创建存储池， 请继续**验证存储池是否已创建。**

示例
以下是基于 GlusterFS 的存储池的 XML 文件示例：

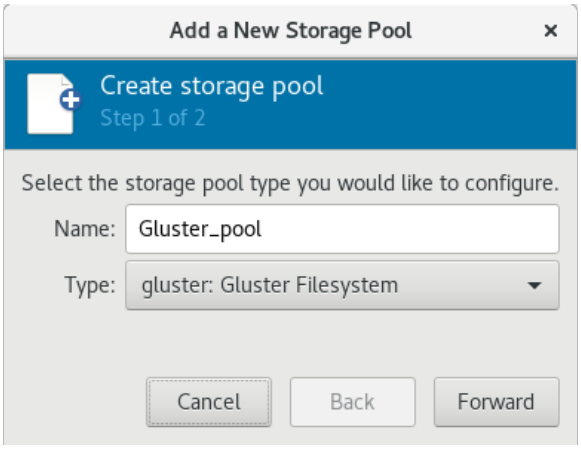
```
<pool type='gluster'>
  <name>Gluster_pool</name>
  <source>
    <host name='111.222.111.222'/>
    <dir path='/'/>
    <name>gluster-vol1</name>
  </source>
</pool>
```

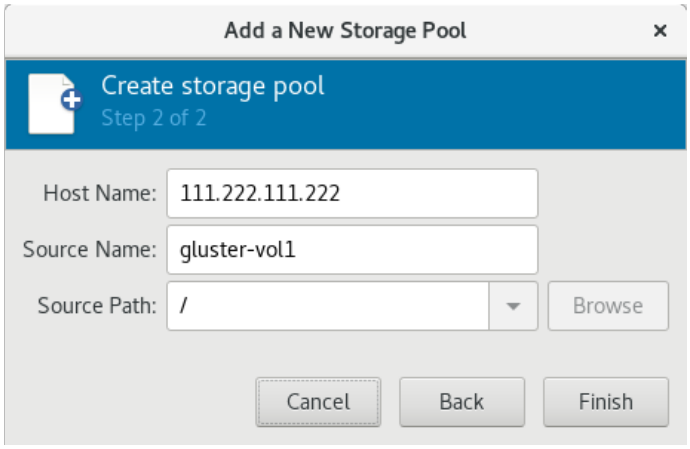
以下是创建基于 GlusterFS 的存储池的命令示例：

```
# pool-define-as --name Gluster_pool --type gluster --source-host 111.222.111.222 --source-name
gluster-vol1 --source-path /
Pool Gluster_pool defined
```

下图显示了虚拟机 XML 配置虚拟机管理器 添加 New Storage Pool 对话框的例子，用于创建基于 GlusterFS 的存储池：

图 13.7. 添加新的基于 GlusterFS 的存储池示例





13.2.3.5. 基于 iSCSI 的存储池

建议
Internet Small Computer System Interface（iSCSI）是用于共享存储设备的网络协议。iSCSI 通过 IP 层使用 SCSI 指令连接到目标（存储服务器）。

使用基于 iSCSI 的设备存储客户机虚拟机允许更灵活的存储选项，比如将 iSCSI 用作块存储设备。iSCSI 设备使用 Linux-IO(LIO)目标。这是 Linux 的多协议 SCSI 目标。除了 iSCSI 外，LIO 还支持通过以太网光纤通道和光纤通道(FCoE)。

先决条件
在创建基于 iSCSI 的存储池前，必须创建 iSCSI 目标。iSCSI 目标使用 targetcli 软件包创建，该软件包提供了用于创建软件支持的 iSCSI 目标的命令集。

过程 13.5. 创建 iSCSI 目标

1. 安装 targetcli 软件包

```
# yum install targetcli
```

2. 启动 **targetcli** 命令集

```
# targetcli
```

3. 创建存储对象
   使用存储池创建三个存储对象。

   a. 创建块存储对象

      i. 导航到 **/backstores/block** 目录。

      ii. 运行 **create** 命令。

```
# create [block-name][filepath]
```

      例如：

```
# create block1 dev=/dev/sdb1
```

   b. 创建 fileio 对象

      i. 进入 **/fileio** 目录。

      ii. 运行 **create** 命令。

```
# create [fileio-name][image-name] [image-size]
```

      例如：

```
# create fileio1 /foo.img 50M
```

   c. 创建 ramdisk 对象

      i. 导航到 **/ramdisk** 目录。

      ii. 运行 **create** 命令。

```
# create [ramdisk-name] [ramdisk-size]
```

例如：

```
# create ramdisk1 1M
```

d. 记录这一步中创建的磁盘名称。稍后将使用它们。

4. 创建 iSCSI 目标

a. 导航到 /iscsi 目录。

b. 以两种方式之一创建目标：

- 运行 create 命令（不带参数）。

  iSCSI 限定名称(IQN)会自动生成。

- 运行 create 命令并指定 IQN 和服务器。例如：

```
# create iqn.2010-05.com.example.server1:iscsirhel7guest
```

5. 定义门户 IP 地址
要通过 iSCSI 导出块存储，必须先配置门户、LUN 和 *访问控制列表* ACL。

该门户包括目标监控器的 IP 地址和 TCP，以及它连接到的启动器。iSCSI 使用端口 3260。默认配置了这个端口。

连接到端口 3260：

a. 进入 /tpg 目录。

b. 运行以下命令：

```
# portals/ create
```

此命令提供侦听端口 3260 的所有可用 IP 地址。

如果只使用单个 IP 地址侦听端口 3260，请将 IP 地址添加到 命令的末尾。例如：

```
# portals/ create 143.22.16.33
```

6. 配置 LUN，并将存储对象分配到光纤
此步骤使用创建存储对象 中创建的 存储对象。

a. 导航到 定义门户 IP 地址 中创建的 TPG的 lun 目录。例如：

```
# iscsi>iqn.iqn.2010-05.com.example.server1:iscsirhel7guest
```

b. 将第一个 LUN 分配给 ramdisk。例如：

```
# create /backstores/ramdisk/ramdisk1
```

c. **将第二个 LUN 分配给块磁盘。例如：**

```
# create /backstores/block/block1
```

d. **将第三个 LUN 分配给 fileio 磁盘。例如：**

```
# create /backstores/fileio/fileio1
```

e. **列出生成的 LUN。**

```
/iscsi/iqn.20...csirhel7guest ls

o- tgp1 ..........[enabled, auth]
  o- acls..........[0 ACL]
  o- luns..........[3 LUNs]
  | o- lun0..........[ramdisk/ramdisk1]
  | o- lun1..........[block/block1 (dev/vdb1)]
  | o- lun2..........[fileio/file1 (foo.img)]
  o- portals..........[1 Portal]
    o- IP-ADDRESS:3260..........[OK]
```

7. 为每个启动器创建 ACL
在启动器连接时启用身份验证。您还可以将指定的 LUN 重新变为指定的非联网器。目标和启动器具有唯一名称。iSCSI 启动器使用 IQN。

a. **使用启动器名称查找 iSCSI initiator 的 IQN。例如：**

```
# cat /etc/iscsi/initiator2.iscsi
InitiatorName=create iqn.2010-05.com.example.server1:iscsirhel7guest
```

此 IQN 用于创建 ACL。

b. **进入 acls 目录。**

c. **通过执行以下操作之一创建 ACL：**

- **通过运行 create 命令（不带参数）为所有 LUN 和启动器创建 ACLS。**

```
# create
```

- **为特定 LUN 和启动器创建 ACL，运行指定 iSCSI intiator 的 IQN 的 create 命令。例如：**

```
# create iqn.2010-05.com.example.server1:888
```

- **配置内核目标，以为所有启动器使用单个用户 ID 和密码。**

```
# set auth userid=user_ID
# set auth password=password
# set attribute authentication=1
# set attribute generate_node_acls=1
```

完成此步骤后，通过 保护存储池来继续。

8. **保存配置**
   通过覆盖之前的引导设置使配置持久。

```
# saveconfig
```

9. **启用服务**
   要在下一次启动时应用保存的设置，请启用该服务。

```
# systemctl enable target.service
```

可选流程
在创建基于 iSCSI 的存储池前，您可以使用 iSCSI 目标执行多个可选步骤。

过程 13.6. 在 RAID 阵列中配置逻辑卷

1. **创建 RAID5 阵列**
   有关创建 RAID5 阵列的详情，请参考 *Red Hat Enterprise Linux 7 存储管理指南*。

2. **在 RAID5 阵列中创建 LVM 逻辑卷**
   有关在 RAID5 阵列中创建 LVM 逻辑卷的详情，请参考 *Red Hat Enterprise Linux 7 逻辑卷管理器管理指南*。

过程 13.7. 测试发现性

- **确保新的 iSCSI 设备可被发现。**

```
# iscsiadm --mode discovery --type sendtargets --portal server1.example.com
143.22.16.33:3260,1 iqn.2010-05.com.example.server1:iscsirhel7guest
```

过程 13.8. 测试设备附加

1. **附加新的 iSCSI 设备**
   连接新设备(*iqn.2010-05.com.example.server1:iscsirhel7guest*)，以确定是否可以附加该设备。

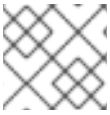
```
# iscsiadm -d2 -m node --login
scsiadm: Max file limits 1024 1024

Logging in to [iface: default, target: iqn.2010-05.com.example.server1:iscsirhel7guest, portal:
143.22.16.33,3260]
Login to [iface: default, target: iqn.2010-05.com.example.server1:iscsirhel7guest, portal:
143.22.16.33,3260] successful.
```

2. **分离设备**

```
# iscsiadm -d2 -m node --logout
scsiadm: Max file limits 1024 1024

Logging out of session [sid: 2, target: iqn.2010-05.com.example.server1:iscsirhel7guest,
portal: 143.22.16.33,3260
Logout of [sid: 2, target: iqn.2010-05.com.example.server1:iscsirhel7guest, portal:
143.22.16.33,3260] successful.
```

过程 13.9. 为 iSCSI 存储池使用 libvirt secret

过程 15.9. 为 iSCSI 存储池使用 libvirt secret



注意

如果在创建 iSCSI 目标时定义了 *user_ID 和密码*，则需要这个过程。

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。这可以在定义池之前或之后配置，但必须启动池才能使身份验证设置生效。

1. **创建 libvirt secret 文件**
   创建一个带有质询身份验证协议(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. **定义 secret**

```
# virsh secret-define secret.xml
```

3. **验证 UUID**

```
# virsh secret-list
UUID                                   Usage
--------------------------------------------------------------------------------
2d7891af-20be-4e5e-af83-190e8a922360  iscsi iscsirhel7secret
```

4. **为 UID 分配 secret**
   使用以下命令，将密码分配给上一步输出中的 UUID。这样可保证 CHAP 用户名和密码位于由 libvirt 控制的 secret 列表中。

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

5. **在存储池中添加身份验证条目**
   使用 virsh edit 修改存储池 XML 文件中的<source> 条目，再添加一个 <auth> 元素，指定 *验证类型*、*用户名* 和 *secret 使用*。

   例如：

```
<pool type='iscsi'>
  <name>iscsirhel7pool</name>
    <source>
      <host name='192.168.122.1'/>
      <device path='iqn.2010-05.com.example.server1:iscsirhel7guest'/>
      <auth type='chap' username='redhat'>
        <secret usage='iscsirhel7secret'/>
      </auth>
    </source>
  <target>
```

```
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

注意

<auth> **子元素存在于客户机 XML 的*<pool>* 和 *<disk>* 元素的不同位置。对于 *<pool>*，*<auth>* 在 *<source>* 元素中指定，这描述了查找池源的位置，因为身份验证是某些池源（iSCSI 和 RBD）的属性。对于 *<disk>*，这是域的子元素，对 iSCSI 或 RBD 磁盘的身份验证是磁盘的属性。**

另外，磁盘的 *<auth>* 子元素与存储池的不同。

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

6. **激活更改**
   必须启动存储池才能激活这些更改。

   - **如果存储池尚未启动，请按照使用 virsh 创建存储池中的步骤来定义和启动 存储池。**
   - **如果池已启动，输入以下命令停止并重启存储池：**

```
# virsh pool-destroy iscsirhel7pool
# virsh pool-start iscsirhel7pool
```

参数
下表提供了 XML 文件、virsh pool-define-as 命令和 Virtual Machine Manager 应用程序所需的参数列表，用于创建基于 iSCSI 的存储池。

表 13.5. 基于 iSCSI 的存储池参数

描述	XML	pool-define-as	虚拟机管理器
存储池的类型	<pool type='iscsi'>	[type] iscsi	iSCSI：iSCSI 目标
存储池的名称	<name>*name*</name>	[name] *name*	Name
主机的名称。	<source> <host name='*hostname*' />	source-host *hostname*	主机名
iSCSI IQN。	<device path="*iSCSI_IQN*" /> </source>	source-dev *iSCSI_IQN*	源 IQN

描述	XML	pool-define-as	虚拟机管理器
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>	目标 *path_to_pool*	目标路径
（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>	*请参见以下注释。*	启动器 IQN

注意

iSCSI initiator 的 IQN 可使用 virsh find-storage-pool-sources-as iscsi 命令确定。

如果您使用 virsh 创建存储池， 请继续 **验证存储池是否已创建。**

示例
以下是基于 iSCSI 的存储池的 XML 文件示例：

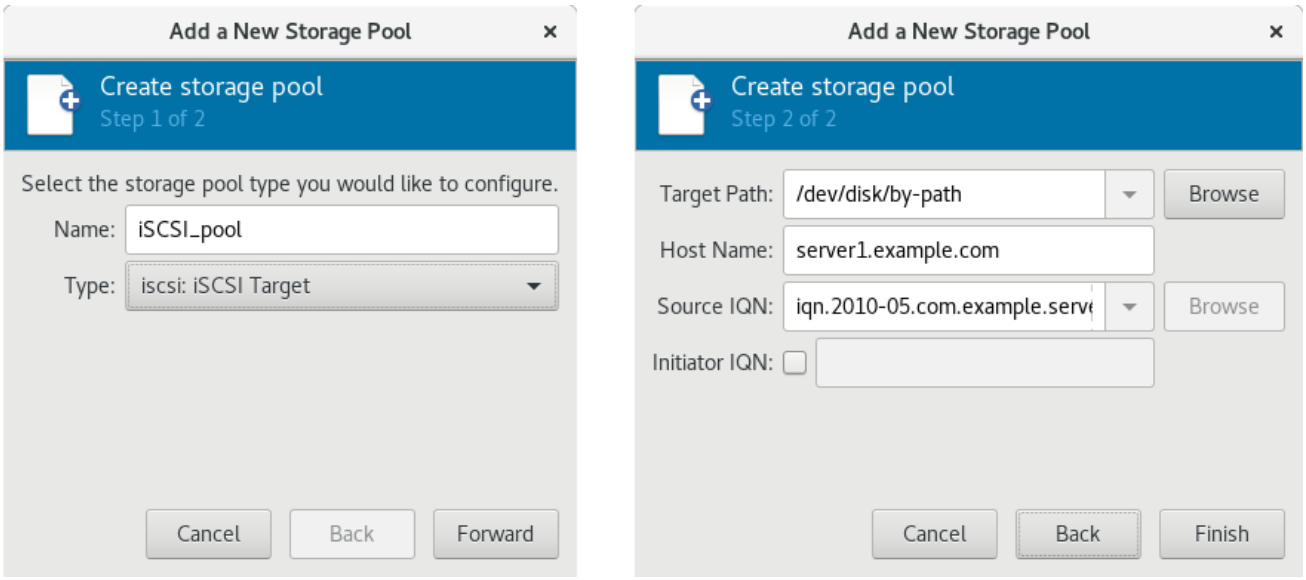
```
<pool type='iscsi'>
  <name>iSCSI_pool</name>
  <source>
    <host name='server1.example.com'/>
    <device path='iqn.2010-05.com.example.server1:iscsirhel7guest'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

以下是创建基于 iSCSI 的存储池的命令示例：

```
# virsh pool-define-as --name iSCSI_pool --type iscsi --source-host server1.example.com --source-
dev iqn.2010-05.com.example.server1:iscsirhel7guest --target /dev/disk/by-path
Pool iSCSI_pool defined
```

下图显示了虚拟机 XML 配置虚拟机管理器 来添加 New Storage Pool 对话框的例子，用于创建基于 iSCSI 的存储池：

图 13.8. 添加新的基于 iSCSI 的存储池示例



13.2.3.6. 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。您可以使用逻辑卷管理器命令或 virsh 命令创建卷组。要使用 virsh 接口管理卷组，请使用 virsh 命令创建卷组。

  有关卷组的详情，请查看 *Red Hat Enterprise Linux 逻辑卷管理器管理指南*。
- 基于 LVM 的存储池需要一个完整磁盘分区。如果用这些步骤激活新分区或设备，分区将被格式化并删除所有数据。如果使用主机的现有卷组(VG)，则不会删除任何内容。建议您在完成以下步骤前备份存储设备。

  有关创建 LVM 卷组的详情，请参考 *Red Hat Enterprise Linux Logical Volume Manager Administration Guide*。
- 如果您在现有 VG 上创建基于 LVM 的存储池，则不应运行 pool-build 命令。

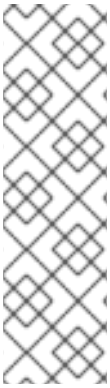
确保准备好 VG 后，继续使用 定义存储池来继续创建存储池。

参数
下表提供了 XML 文件、virsh pool-define-as 命令和 Virtual Machine Manager 应用程序所需的参数列表，用于创建基于 LVM 的存储池。

表 13.6. 基于 LVM 的存储池参数

描述	XML	pool-define-as	虚拟机管理器
存储池的类型	<pool type='logical'>	[type] logical	逻辑：LVM 卷组

描述	XML	pool-define-as	虚拟机管理器
存储池的名称	<name>*name*</name>	[name] *name*	Name
存储池设备的路径	<source> <device path='*device_path*' />	source-dev *device_path*	源路径
卷组名称	<name='*VG-name*' />	source-name *VG-name*	源路径
虚拟组格式	<format type='lvm2' /> </source>	source-format lvm2	不适用
目标路径	<target> <path='*target-path*' /> </target>	target *target-path*	目标路径



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>
```

如果您使用 virsh 创建存储池，请继续验证存储池是否已创建。

示例
以下是基于 LVM 的存储池的 XML 文件示例：

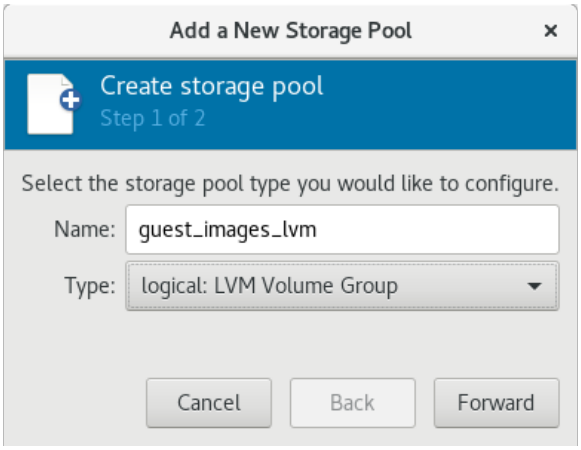
```
<pool type='logical'>
  <name>guest_images_lvm</name>
  <source>
    <device path='/dev/sdc'/>
    <name>libvirt_lvm</name>
    <format type='lvm2'/>
  </source>
  <target>
    <path>/dev/libvirt_lvm</path>
  </target>
</pool>
```

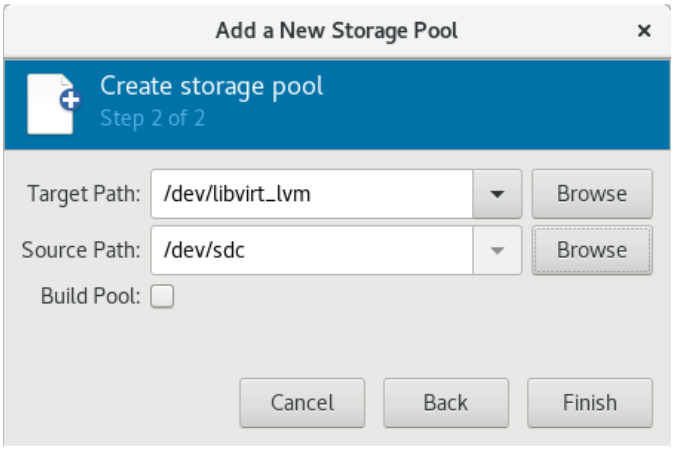
以下是创建基于 LVM 的存储池的命令示例：

```
# virsh pool-define-as guest_images_lvm logical --source-dev=/dev/sdc --source-name libvirt_lvm --
target /dev/libvirt_lvm
Pool guest_images_lvm defined
```

下图显示了虚拟机 XML 配置虚拟机管理器 来添加 New Storage Pool 对话框的例子，用于创建基于 LVM 的存储池：

图 13.9. 添加一个基于 LVM 的新存储池示例





13.2.3.7. 基于 NFS 的存储池

先决条件
要创建基于网络文件系统(NFS)的存储池，主机机器应该已经配置了一个 NFS 服务器。有关 NFS 的详情，请查看 *Red Hat Enterprise Linux Storage Administration Guide*。

确保正确配置了 NFS 服务器后，使用 定义存储池来继续创建存储池。

参数
下表提供了 XML 文件、**virsh pool-define-as** 命令和 Virtual Machine Manager 应用程序所需的参数列表，用于创建基于 NFS 的存储池。

表 13.7. 基于 NFS 的存储池参数

描述	XML	pool-define-as	虚拟机管理器
存储池的类型	<pool type='netfs'>	[type] netfs	NETFS：网络导出的目录
存储池的名称	<name>*name*</name>	[name] *name*	Name
挂载点所在的 NFS 服务器的主机名称。这可以是主机名或 IP 地址。	<source> <host name='*host_name*' />	source-host *host_name*	主机名
NFS 服务器中使用的目录	<dir path='*source_path*' /> </source>	source-path *source_path*	源路径

描述	XML	pool-define-as	虚拟机管理器
指定目标的路径。这将是用于存储池的路径。	<target> <path>/*target_path*</path> </target>	target *target_path*	目标路径

如果您使用 virsh 创建存储池， 请继续 **验证存储池是否已创建。**

示例
以下是基于 NFS 的存储池的 XML 文件示例：

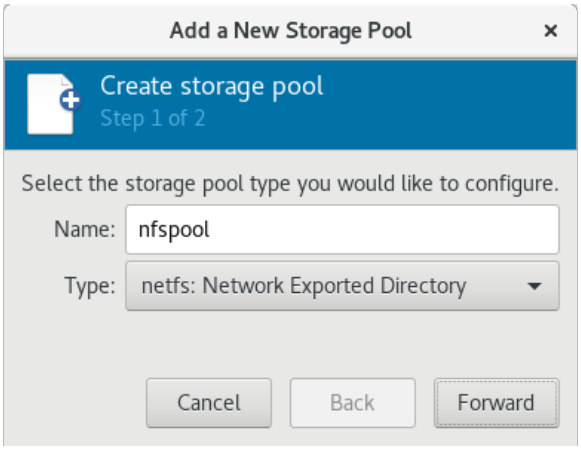
```
<pool type='netfs'>
  <name>nfspool</name>
  <source>
    <host name='localhost'/>
    <dir path='/home/net_mount'/>
  </source>
  <target>
    <path>/var/lib/libvirt/images/nfspool</path>
  </target>
</pool>
```

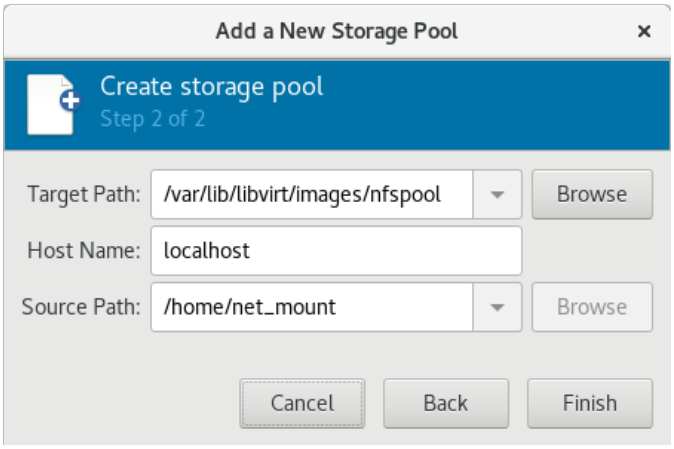
以下是创建基于 NFS 的存储池的命令示例：

```
# virsh pool-define-as nfspool netfs --source-host localhost --source-path /home/net_mount --target
/var/lib/libvirt/images/nfspool
Pool nfspool defined
```

下图显示了虚拟机 XML 配置虚拟机管理器 来添加 New Storage Pool 对话框的例子，用于创建基于 NFS 的存储池：

图 13.10. 添加新的基于 NFS 的存储池示例





13.2.3.8. 使用 SCSI 设备使用基于 vHBA 的存储池

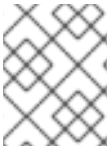
注意

您不能使用 Virtual Machine Manager使用 SCSI 设备创建基于 vHBA 的存储池。

建议
N_Port ID 虚拟化(NPIV)是一种软件技术，允许共享单一物理光纤通道主机总线适配器(HBA)。这样，多个虚拟机可以从多个物理主机查看相同的存储，因而能轻松查看存储的迁移路径。因此，只要指定了正确的存储路径，迁移不需要创建或复制存储。

在虚拟化中，*虚拟主机总线适配器*或 *vHBA* 控制虚拟机的逻辑单元号(LUN)。要使主机在多个 KVM 虚拟机间共享一个光纤通道设备路径，必须为每个虚拟机创建一个 vHBA。多个 KVM 客户机不得使用单个 vHBA。

NPIV 的每个 vHBA 都由其父 HBA 以及自己的 World Wide Node Name(WWN)和 World Wide Port Name(WWPN)标识。存储的路径由 WWNN 和 WWPN 值决定。父 HBA 可以定义为 **scsi_host#**或 WWN/WWPN 对。



注意

如果父 HBA 定义为 **scsi_host#**，且硬件添加到主机计算机，scsi_host#分配可能会改变。因此，建议您使用 WWNN/WWPN 对定义父 HBA。

建议您基于 vHBA 定义 libvirt 存储池，因为这会保留 vHBA 配置。

使用 libvirt 存储池有两个主要优点：

- libvirt 代码可使用 virsh 命令输出轻松查找 LUN 路径。
- 虚拟机迁移只需要在目标机器上定义和启动具有相同 vHBA 名称的存储池。要做到这一点，在虚拟机 XML 配置中必须指定 vHBA LUN、libvirt 存储池和卷名称。例如，请参阅 第 13.2.3.8 节 “使用 SCSI 设备使用基于 vHBA 的存储池”。

注意

在创建 vHBA 前，建议您在主机 LUN 中配置存储阵列(SAN)侧 zoning，以提供客户机之间的隔离，并防止数据崩溃的可能性。

要创建持久的 vHBA 配置，首先使用以下格式创建一个 libvirt 'scsi' 存储池 XML 文件。当创建一个使用同一物理 HBA 中的存储池的 vHBA 时，建议为 <path> 值使用一个稳定的位置，比如系统中的 /dev/disk/by-{path|id|uuid|label} 位置之一。

当创建多个使用同一物理 HBA 中的存储池的 vHBA 时，<path> 字段的值只能是 /dev/，否则存储池卷只对其中一个 vHBA 可见，主机中的设备无法通过 NPIV 配置公开给多个客户机。

有关 <path> 和 <target> 中的元素的更多信息，请参阅 上游 libvirt 文档。

先决条件
在使用 SCSI 设备创建基于 vHBA 的存储池前，先创建一个 vHBA：

过程 13.10. 创建 vHBA

1. 在主机系统中找到 HBA
   要在主机系统中找到 HBA，请使用 **virsh nodedev-list --cap vports** 命令。

   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. 检查 HBA 的详情
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 的最大数量。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>
```

在这个示例中，**<max_vports>** 值显示可在 HBA 配置中使用总计 127 个虚拟端口。**<vports>** 值显示当前使用的虚拟端口数。这些值在创建 vHBA 后更新。

3. 创建 vHBA 主机设备
为 vHBA 主机创建类似如下的 XML 文件。在本例中，该文件名为 *vhba_host3.xml*。

这个示例使用 **scsi_host*3*** 来描述父 vHBA。

```
# cat vhba_host3.xml
<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。

```
# cat vhba_host3.xml
<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 过程 13.10, “创建 vHBA” 中所示的 HBA 详情中的值匹配。

<parent> 字段指定要与这个 vHBA 设备关联的 HBA 设备。<device> 标签中的详情在下一步中使用来为主机创建新 vHBA 设备。有关 nodedev XML 格式的详情，请查看 libvirt 上游页面。

4. **在 vHBA 主机设备中创建新 vHBA**
   要基于 *vhba_host3* 创建 vHBA，请使用 virsh nodedev-create 命令：

```
# virsh nodedev-create vhba_host3.xml
Node device scsi_host5 created from vhba_host3.xml
```

5. **验证 vHBA**
   使用 virsh nodedev-dumpxml 命令验证新的 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>
```

验证 vHBA 后，继续使用 定义存储池 的存储池。

参数
下表提供了 XML 文件、virsh pool-define-as 命令和 Virtual Machine Manager 应用程序所需的参数列表，用于创建基于 vHBA 的存储池。

表 13.8. 基于 vHBA 的存储池参数

描述	XML	pool-define-as
存储池的类型	<pool type='scsi'>	scsi

描述	XML	pool-define-as
存储池的名称	<name>*name*</name>	--adapter-name *name*
vHBA 的标识符。**父** 属性为可选。	<source> <adapter type='fc_host' [parent=*parent_scsi_device*] wwnn='*WWNN*' wwpn='*WWPN*' /> </source>	[--adapter-parent *parent*] --adapter-wwnn *wwnn* --adapter-wpnn *wwpn*
指定目标的路径。这将是用于存储池的路径。	<target> <path>*target_path*</path> </target>	目标 *path_to_pool*

重要

当 <path> 字段是 /dev/ 时，libvirt 为卷设备路径生成唯一的短设备路径。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客户机中列出同一个卷。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

parent 属性可以在 <adapter> 字段中使用它来识别不同路径可使用 NPIV LUN 的物理 HBA 父父项。此字段 scsi_host*N* 与 vports 和 max_vports 属性相结合，以完成父项识别。parent、parent_wwnn、parent_wwpn 或 parent_fabric_wwn 属性提供在主机重启后使用相同的 HBA 的不同保证程度。

- **如果没有指定 parent，libvirt 将使用支持 NPIV 的第一个 scsi_host*N* 适配器。**
- **如果只指定 parent，则在配置中添加额外的 SCSI 主机适配器时会出现问题。**
- **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，在主机重启同一光纤中的 HBA 后，无论使用的 scsi_host*N* 是什么，都会选择同一光纤中的 HBA。**

如果您使用 virsh 创建存储池，请继续验证存储池是否已创建。

示例
以下是基于 vHBA 的存储池的 XML 文件示例。第一个示例是 HBA 中唯一存储池的存储池示例。第二个示例是存储池之一，它是使用多个存储池之一，它使用单个 vHBA 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' wwnn='5001a4a93526d0a1' wwpn='5001a4ace3ee047d'/>
  </source>
  <target>
```

```
    <path>/dev/disk/by-path</path>
  </target>
</pool>
```

```
<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>
```

以下是创建基于 vHBA 的存储池的命令示例：

```
# virsh pool-define-as vhbapool_host3 scsi --adapter-parent scsi_host3 --adapter-wwnn
5001a4a93526d0a1 --adapter-wwpn 5001a4ace3ee047d --target /dev/disk/by-path
Pool vhbapool_host3 defined
```

注意

virsh 命令不提供定义 parent_wwnn、parent_wwpn 或 parent_fabric_wwn 属性的方法。

配置虚拟机以使用 vHBA LUN
为 vHBA 创建存储池后，必须将 vHBA LUN 添加到虚拟机配置中。

1. 在虚拟机 XML 的虚拟机中创建一个磁盘卷。
2. 在 storage pool 参数中指定 storage volume 和 <source>。

以下示例显示了：

```
<disk type='volume' device='disk'>
  <driver name='qemu' type='raw'/>
  <source pool='vhbapool_host3' volume='unit:0:4:0'/>
  <target dev='hda' bus='ide'/>
</disk>
```

指定 lun 设备而不是 disk，请参阅以下示例：

```
<disk type='volume' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw'/>
  <source pool='vhbapool_host3' volume='unit:0:4:0' mode='host'/>
  <target dev='sda' bus='scsi'/>
  <shareable />
</disk>
```

有关在客户机中添加基于 SCSI LUN 的存储的 XML 配置示例，请参考 第 13.3.6.3 节 “在客户机中添加基于 SCSI LUN 的存储”。

请注意，为了确保在硬件故障时确保成功连接到 LUN，建议您编辑 fast_io_fail_tmo 和 dev_loss_tmo 选项。如需更多信息，请参阅在硬件故障后重新连接到公开的 LUN。

13.2.4. 删除存储池

您可以使用 virsh 或 Virtual Machine Manager 删除存储池。

13.2.4.1. 删除存储池的先决条件

为了避免对使用您要删除的存储池的其他客户机虚拟机造成负面影响，建议您停止存储池并释放它所使用的任何资源。

13.2.4.2. 使用 virsh 删除存储池

1. 列出定义的存储池：

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

2. 停止您要删除的存储池。

```
# virsh pool-destroy guest_images_disk
```

3. （可选）对于某些类型的存储池，您可以选择删除存储池所在的目录：

```
# virsh pool-delete guest_images_disk
```

4. 删除存储池的定义。

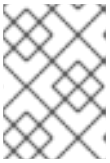
```
# virsh pool-undefine guest_images_disk
```

5. 确认池未定义：

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

13.2.4.3. 使用虚拟机管理器删除存储池

1. 在 Connection Details 窗口的 Storage 选项卡中，选择您要删除的存储池列表。
2. 点 Storage 窗口底部的  。这会停止存储池，并释放其正在使用的任何资源。
3. 点  。



注意

只有在存储池停止时才会启用  图标。

存储池已删除。

13.3. 使用存储卷

本节提供有关使用存储卷的信息。它提供了概念的信息，以及使用 virsh 命令和虚拟机管理器创建、编辑和删除存储卷的详细信息。

13.3.1. 存储卷概念

存储池被分成一个 *存储卷*。存储卷是物理分区的抽象、LVM 逻辑卷、基于文件的磁盘镜像，以及由 libvirt 处理的其他存储类型。无论底层硬件如何，存储卷都以本地存储设备的形式呈现给客户机虚拟机。

注意

下面的部分不包含 virsh 提供的所有可能命令和参数，用来管理存储卷>。如需更多信息，请参阅 第 20.30 节 “存储卷命令”。

在主机机器上，存储卷的名称及其派生到的存储池的标识符。在 virsh 命令行中，格式为 ***--pool*** *storage_pool volume_name*。

例如，*guest_images* 池中名为 *firstimage* 的卷。

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

有关其他参数和参数，请参阅 第 20.34 节 “列出卷信息”。

13.3.2. 创建存储卷

这部分介绍了使用 **virsh** 和 Virtual Machine Manager 从存储池创建存储卷的一般说明。创建存储卷后，您可以向 客户机添加存储设备。

13.3.2.1. 使用 virsh 创建存储卷

执行以下操作之一：

- 使用 XML 文件定义存储卷。

  a.创建包含新设备所需的存储卷信息的临时 XML 文件。

  XML 文件必须包含包括以下内容在内的具体字段：

  - Name - 存储卷的名称。

- allocation - 存储卷的总存储分配。
- capacity - 存储卷的逻辑容量。如果卷是稀疏的，则这个值可能与分配值 不同。
- Target - 主机系统中的存储卷的路径，以及它的权限和标签（可选）。

下面显示了一个存储卷定义 XML 文件示例。在本例中，该文件被保存到 ~/guest_volume.xml

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

b.使用 virsh vol-create 命令创建基于 XML 文件的存储卷。

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

c.删除步骤中创建的 XML 文件。

- 使用 virsh vol-create-as 命令创建存储卷。

```
# virsh vol-create-as guest_images_dir volume1 20GB --allocation 0
```

- 使用 virsh vol-clone 命令克隆现有的存储卷。virsh vol-clone 命令必须指定存储池，其中包含要克隆的存储卷以及新创建的存储卷的名称。

```
# virsh vol-clone --pool guest_images_dir volume1 clone1
```

13.3.2.2. 使用虚拟机管理器创建存储卷

过程 13.11. 使用虚拟机管理器创建存储卷

1. 打开存储设置

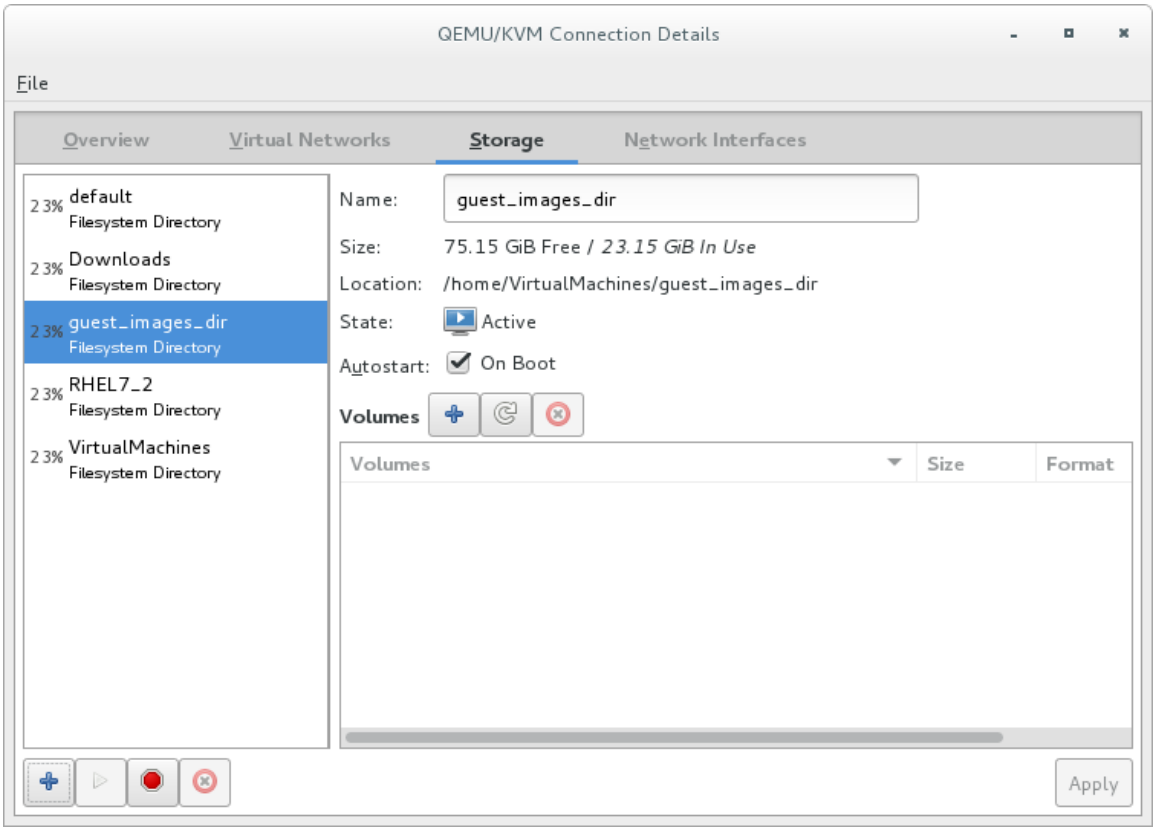
   a.

   在 Virtual Machine Manager 中，打开 Edit 菜单，然后选择 Connection Details。

   b.

   点 Connection Details 窗口中的 Storage 选项卡。

   图 13.11. Storage 标签页



   Connection Details 窗口左侧的窗格会显示一个存储池列表。

2. 选择您要在其中创建存储卷的存储池

   在存储池列表中，点击您要在其中创建存储卷的存储池。

   在所选存储池上配置的任何存储卷都出现在窗口底部的 卷 窗格中。

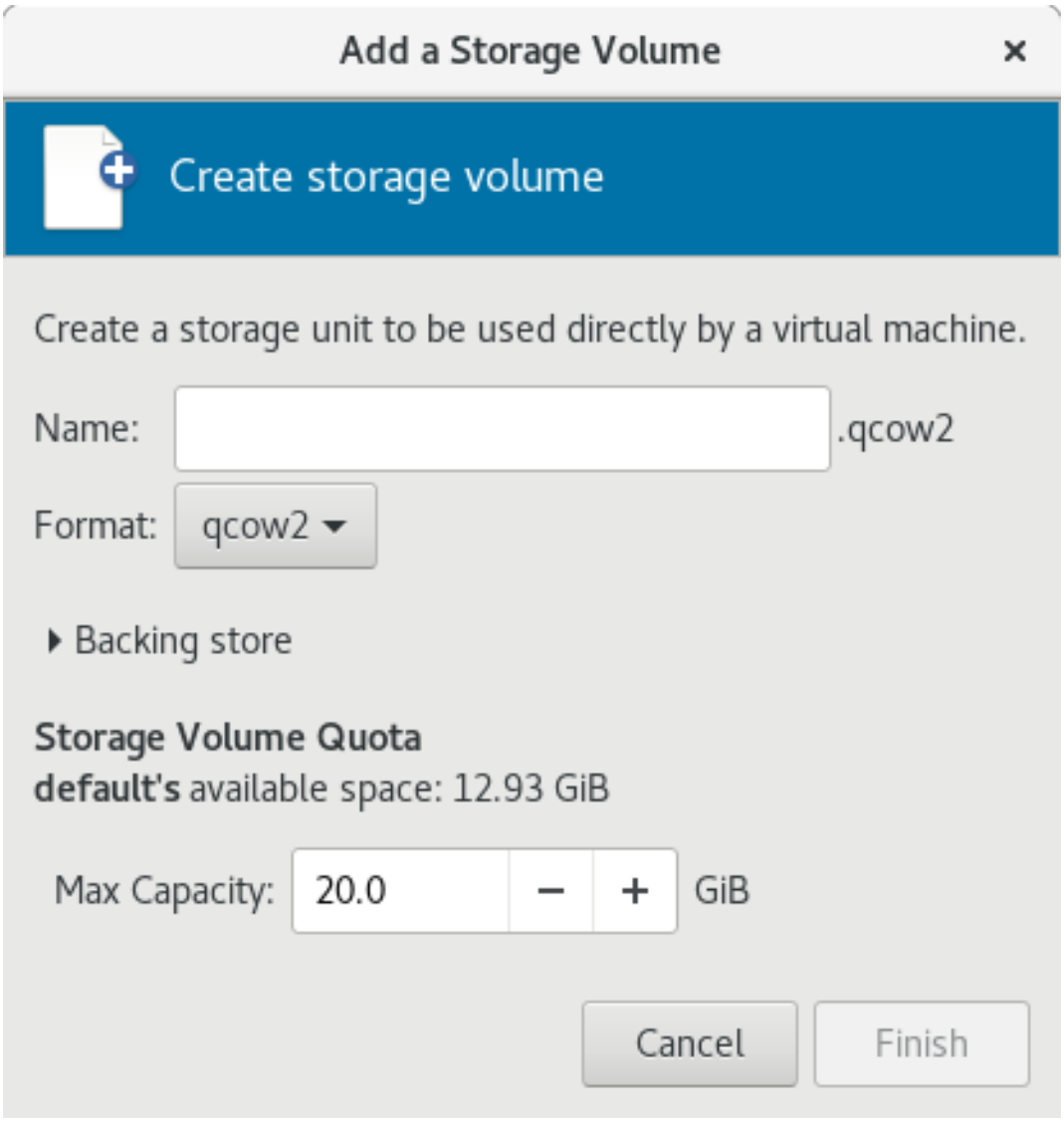
3. 添加新存储卷

   点击 卷 列表上方的

   按钮。此时会出现 Add a Storage Volume 对话框。

图 13.12. 创建存储卷



4. **配置存储卷**

   使用以下参数配置存储卷：

   - **在 Name 字段中输入存储池的名称。**
   - **从 Format 列表中选择存储卷的格式。**
   - **在 Max Capacity 项中输入存储卷的最大大小。**

5. **完成创建过程**

   点 Finish。Add a Storage Volume 对话框关闭，存储卷会出现在 卷 列表中。

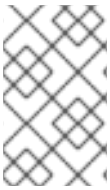
13.3.3. 查看存储卷

您可以从存储池创建多个存储卷。您还可以使用 **virsh vol-list** 命令列出存储池中的存储卷。在以下示例中，***guest_images_disk*** 包含三个卷。

```
# virsh vol-create-as guest_images_disk volume1 8G
Vol volume1 created

# virsh vol-create-as guest_images_disk volume2 8G
Vol volume2 created

# virsh vol-create-as guest_images_disk volume3 8G
Vol volume3 created

# virsh vol-list guest_images_disk
Name                 Path
-----------------------------------------
volume1              /home/VirtualMachines/guest_images_dir/volume1
volume2              /home/VirtualMachines/guest_images_dir/volume2
volume3              /home/VirtualMachines/guest_images_dir/volume3
```

13.3.4. 管理数据

本节提供有关管理存储卷上的数据的信息。



注意

某些类型的存储卷不支持所有数据管理命令。具体信息请查看以下部分。

13.3.4.1. 隐藏存储卷

为确保无法访问存储卷中的数据，可以使用 **virsh vol-wipe** 命令来擦除存储卷。

使用 **virsh vol-wipe** 命令来擦除存储卷：

```
# virsh vol-wipe new-vol vdisk
```

默认情况下，数据会被零覆盖。但是，可以指定许多不同的方法来唤醒存储卷。有关 **virsh vol-wipe** 命令的所有选项的详细信息，请参阅 第 20.32 节 “删除存储卷的内容”。

13.3.4.2. 将数据上传到存储卷

您可以使用 virsh vol-upload 命令将数据从指定的本地文件上传到存储卷。

```
# virsh vol-upload --pool pool-or-uuid --offset bytes --length bytes vol-name-or-key-or-path local-file
```

以下是 virsh vol-upload 主选项：

- **--pool *pool-or-uuid* - 卷所处的存储池的名称或 UUID。**
- ***vol-name-or-key-path* - 要上传的卷的名称或密钥或者路径。**
- **--offset *字节* 在存储卷开始写入数据的位置。**
- **--length *length* - 要上传的数据量的上限。**

注意

如果 *local-file* 大于指定的 --length，则会出现错误。

例 13.1. 将数据上传到存储卷

```
# virsh vol-upload sde1 /tmp/data500m.empty disk-pool
```

在本例中，sde1 是 disk-pool 存储池中的一个卷。/tmp/data500m.empty 中的数据被复制到 sde1。

13.3.4.3. 将数据下载到存储卷

您可以使用 virsh vol-download 命令从存储卷下载数据到指定的本地文件。

```
# vol-download --pool pool-or-uuid --offset bytes --length bytes vol-name-or-key-or-path local-file
```

以下是 virsh vol-download 的主要选项：

- **--pool *pool-or-uuid* - 卷所在的存储池的名称或 UUID。**
- ***vol-name-or-key-path* - 要下载的卷的名称、密钥或路径。**
- **--offset - 在存储卷中开始读取数据的位置。**
- **--length *length* - 要下载的数据量的上限。**

例 13.2. 从存储卷下载数据

```
# virsh vol-download sde1 /tmp/data-sde1.tmp disk-pool
```

在本例中，sde1 是 disk-pool 存储池中的一个卷。sde1 中的数据已下载到 /tmp/data-sde1.tmp 中。

13.3.4.4. 调整存储卷大小

您可以使用 vol-resize 命令重新定义指定存储卷的容量。

```
# virsh vol-resize --pool pool-or-uuid vol-name-or-path pool-or-uuid capacity --allocate --delta --shrink
```

***容量* 以字节为单位表示。命令需要 --pool *pool-or-uuid*，这是卷所在存储池的名称或 UUID。此命令还需要 *vol-name-or-key-or-path*、名称、密钥或卷路径才能调整大小。**

除非指定了 --allocate，否则新容量可能是稀疏的。通常，容量是新大小，但如果存在 --delta，则会将其添加到现有大小中。除非出现 --shrink，尝试缩小卷将失败。

请注意，除非提供了 --shrink 且不需要负数值，否则 *容量* 为负数。*容量* 是一个缩放整数，如果没有后缀，则默认为 字节。另外，请注意这个命令只能安全地供活跃客户端使用的存储卷。请参阅 第 20.13.3 节 “更改客户机虚拟机块设备的大小” 以获取实时大小。

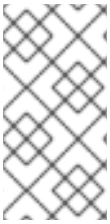
例 13.3. 重新定义存储卷大小

例如，如果您创建了 50M 存储卷，您可以使用以下命令将其大小调整为 100M：

```
# virsh vol-resize --pool disk-pool sde1 100M
```

13.3.5. 删除存储卷

您可以使用 virsh 或 虚拟机管理器删除存储卷。



注意

为了避免对使用您要删除的存储卷的虚拟机虚拟机造成负面影响，建议您使用它释放任何资源。

13.3.5.1. 使用 virsh 删除存储卷

使用 virsh vol-delete 命令删除存储卷。该命令必须指定存储卷的名称或路径以及提取存储卷的存储池。

以下示例从 *guest_images_dir* 存储池中删除 *volume_name* 存储卷：

```
# virsh vol-delete volume_name --pool guest_images_dir
  vol volume_name deleted
```

13.3.5.2. 使用虚拟机管理器删除存储卷

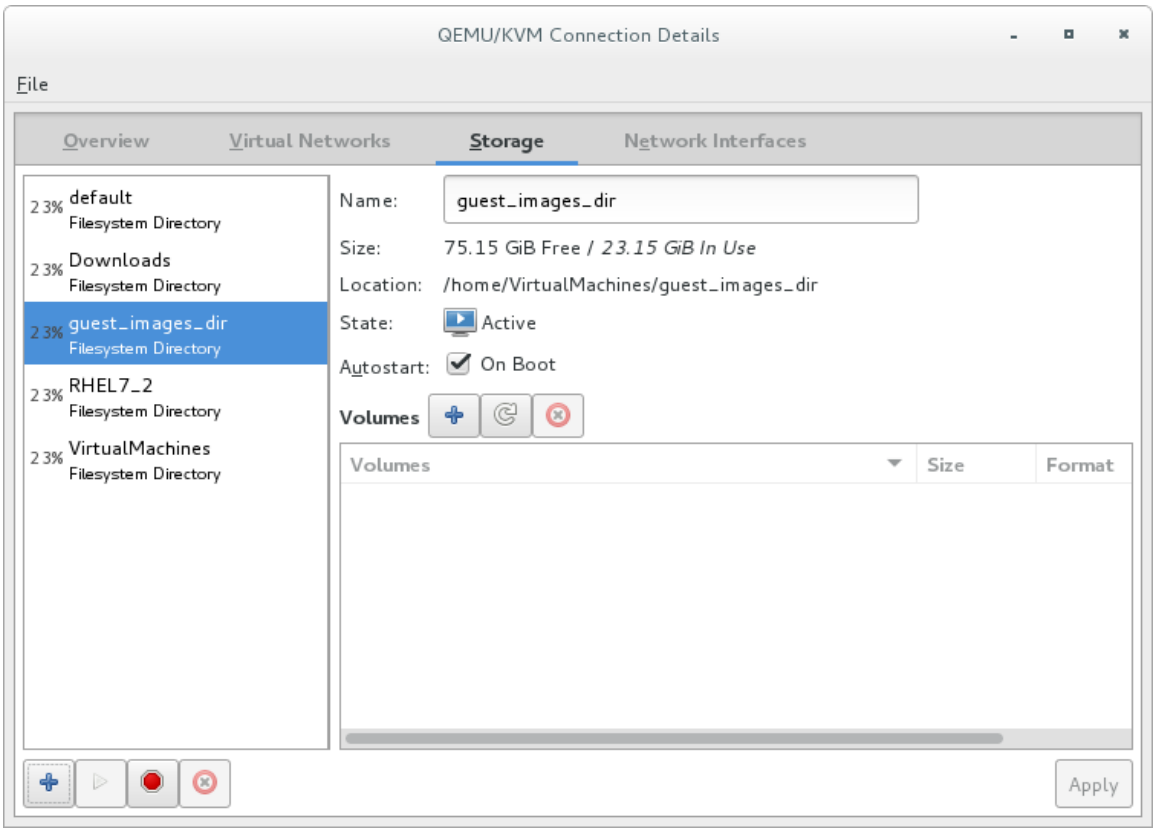
过程 13.12. 使用虚拟机管理器删除存储卷

1. 打开存储设置

   a. 在 Virtual Machine Manager 中，打开 Edit 菜单，然后选择 Connection Details。

   b. 点 Connection Details 窗口中的 Storage 选项卡。

图 13.13. Storage 标签页



Connection Details 窗口左侧的窗格会显示一个存储池列表。

2. 选择您要删除的存储卷

a. 在存储池列表中，点击提取存储卷的存储池。

在所选存储池上配置的存储卷列表显示在窗口底部的 卷 窗格中。

b. 选择您要删除的存储卷。

3. 删除存储卷

a. 点击

按钮（移动 卷 列表）。此时会出现确认对话框。

b. 单击 "是"所选存储卷被删除。

13.3.6. 在客户机中添加存储设备

您可以使用 virsh 或 Virtual Machine Manager 在客户机虚拟机中添加存储设备。

13.3.6.1. 使用 virsh 在客户机中添加存储设备

要将存储设备添加到客户机虚拟机，请使用 attach-disk 命令。可以在 XML 文件或命令行中指定有关要添加的磁盘信息的参数。

以下是包含存储定义的 XML 文件示例。

```
<disk type='file' device='disk>'>
  <driver name='qemu' type='raw' cache='none'/>
  <source file='/var/lib/libvirt/images/FileName.img'/>
  <target dev='vdb' bus='virtio'/>
</disk>
```

以下命令使用名为 NewStorage.xml 的 XML 文件将磁盘附加到 *Guest1*。

```
# virsh attach-disk --config Guest1 ~/NewStorage.xml
```

以下命令将磁盘附加到 *Guest1*，而不使用 xml 文件。

```
# virsh attach-disk --config Guest1 --source /var/lib/libvirt/images/FileName.img --target vdb
```

13.3.6.2. 使用虚拟机管理器在客户机中添加存储设备

您可以将存储卷添加到客户机虚拟机，或者创建或添加默认存储设备到客户机虚拟机。

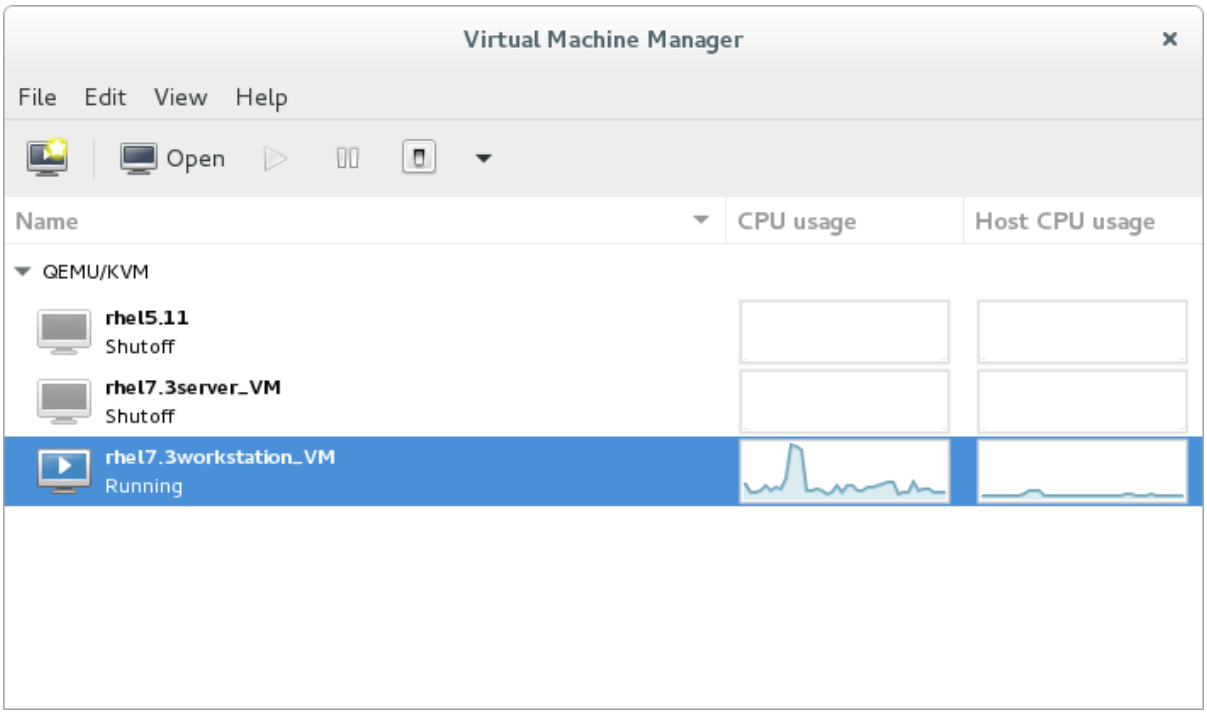
13.3.6.2.1. 在客户机中添加存储卷

在客户机虚拟机中添加存储卷：

1. 在虚拟机硬件详情窗口中打开虚拟机管理器

   以 root 用户身份执行 virt-manager 命令打开 virt-manager，或打开 Applications → System Tools → Virtual Machine Manager。

图 13.14. Virtual Machine Manager 窗口



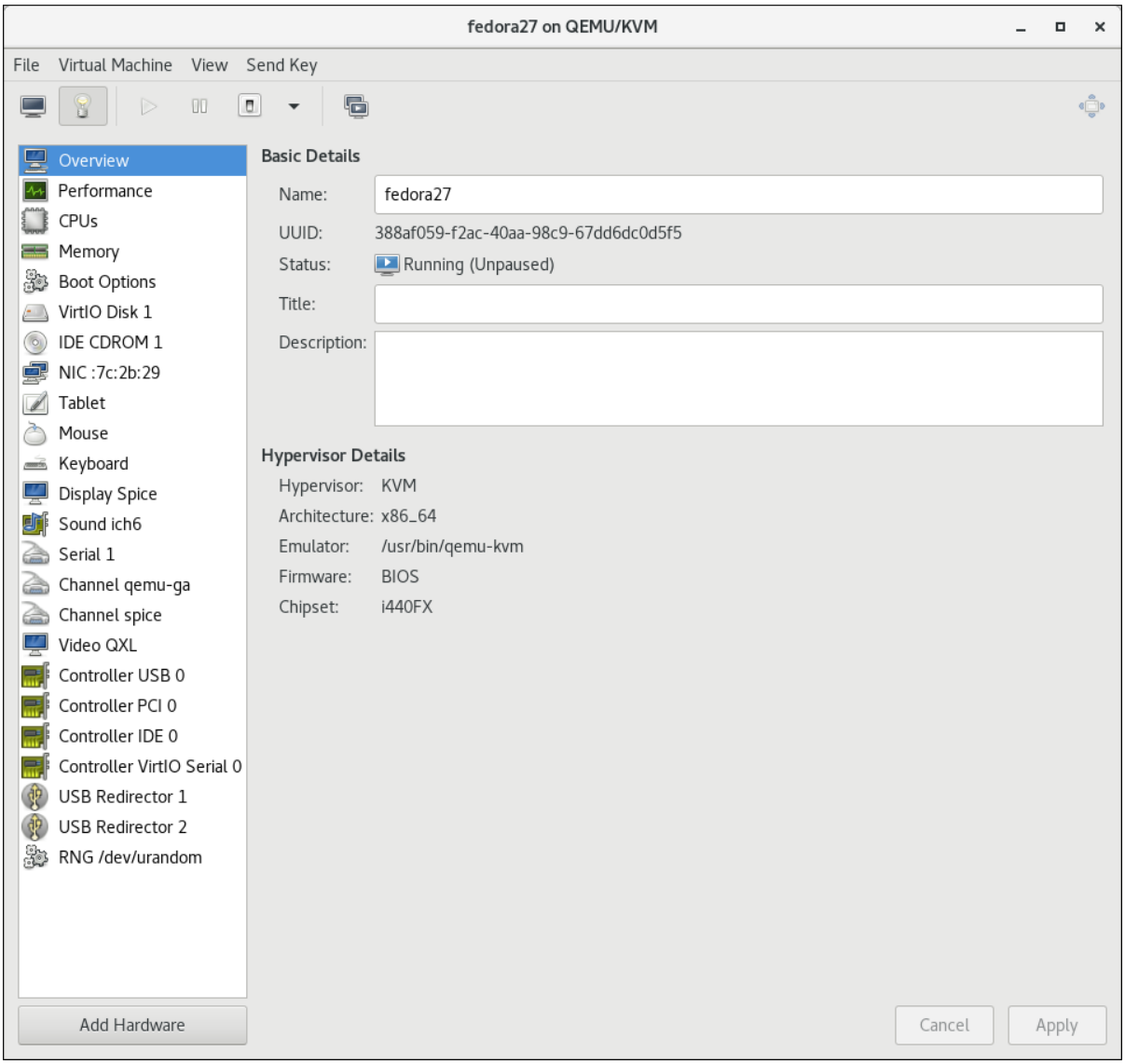
选择您要添加存储卷的客户机虚拟机。

点 Open。此时会打开 Virtual Machine 窗口。

点

。此时会出现硬件详细信息窗口。

图 13.15. 硬件详情窗口

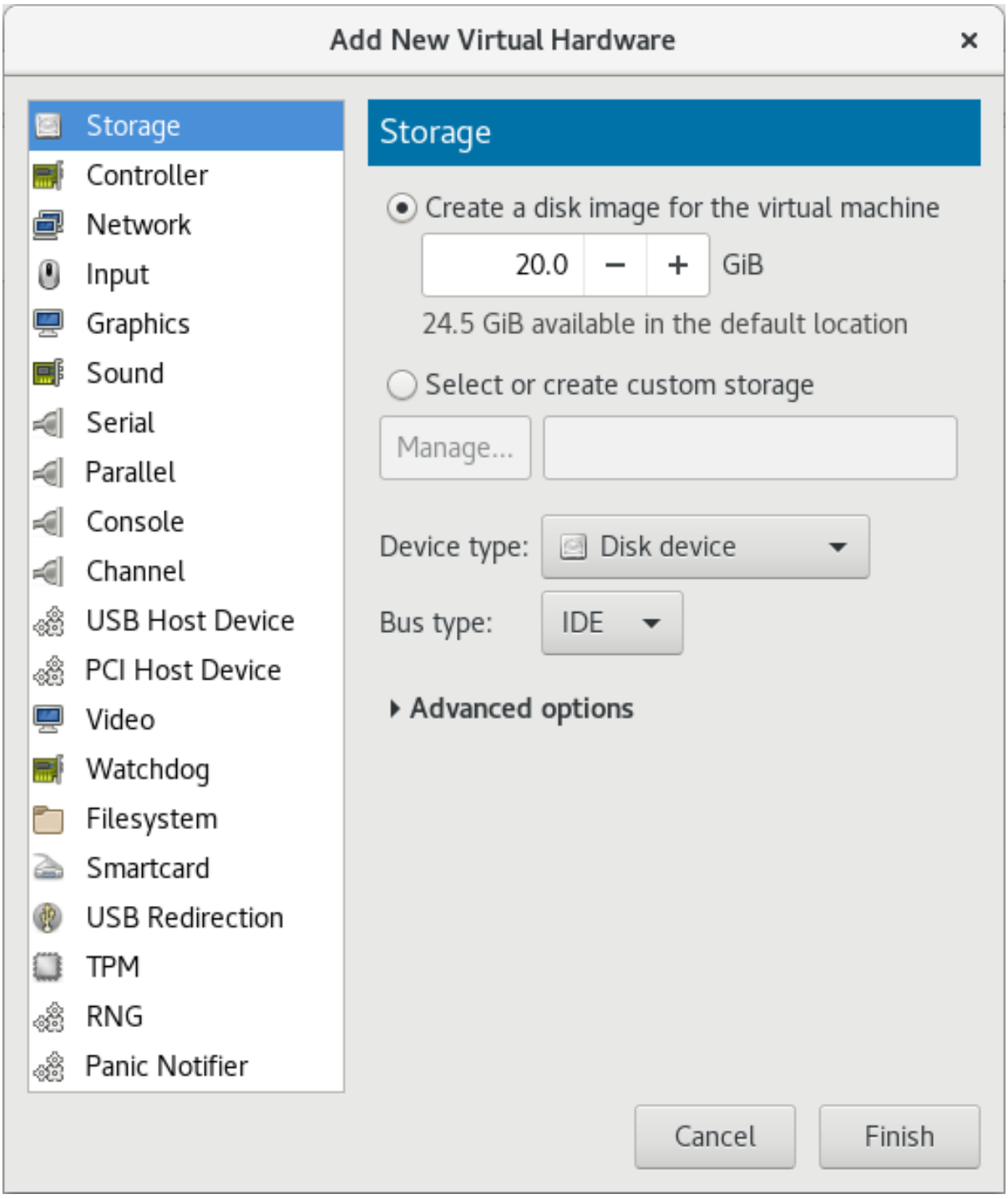


2. **打开 Add New Virtual Hardware 窗口**

   单击 Add Hardware。此时会出现 Add New Virtual Hardware 窗口。

   确定在硬件类型窗格中选择了 Storage。

图 13.16. Add New Virtual Hardware 窗口

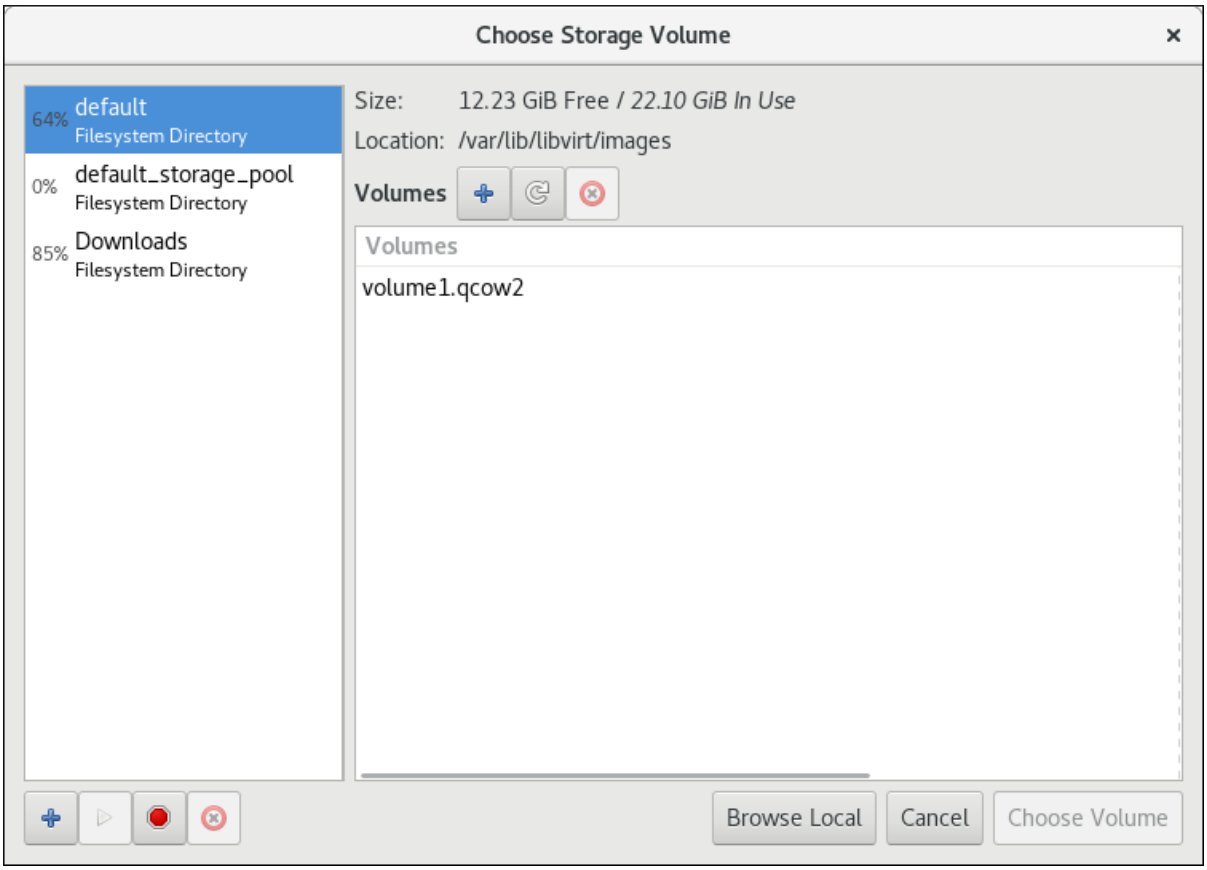


3. 查看存储卷列表

   选择选择或创建自定义存储选项 按钮。

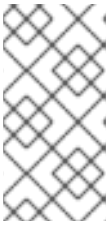
   单击 **Manage**。此时会出现 **Choose Storage Volume** 对话框。

图 13.17. Select Storage Volume 窗口



4. 选择存储卷

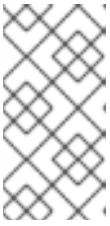
从 Select Storage Volume 窗口左侧的列表中选择一个存储池。所选存储池中的存储卷列表会出现在 Volumes 列表中。



注意

您可以从 Select Storage Volume 窗口中创建存储池。如需更多信息，请参阅 第 13.2.2.2 节 “使用虚拟机管理器创建存储池”。

从 Volumes 列表中选择存储卷。



注意

您可以从 Select Storage Volume 窗口中创建存储卷。如需更多信息，请参阅 第 13.3.2.2 节 “使用虚拟机管理器创建存储卷”。

单击 Choose Volume。Select Storage Volume 窗口关闭。

5. **配置存储卷**

从设备类型列表中选择 设备类型。可用的类型有：磁盘设备、Floppy 设备和 LUN 直通.

从 Bus 类型 列表中选择总线类型。可用的总线类型取决于所选的设备类型。

从 Cache 模式列表中选择缓存模式。可用的缓存模式有：Hypervisor default, none, writethrough, writeback, directsync, insecure

点 Finish。Add New Virtual Hardware 窗口关闭。

13.3.6.2.2. 在客户机中添加默认存储

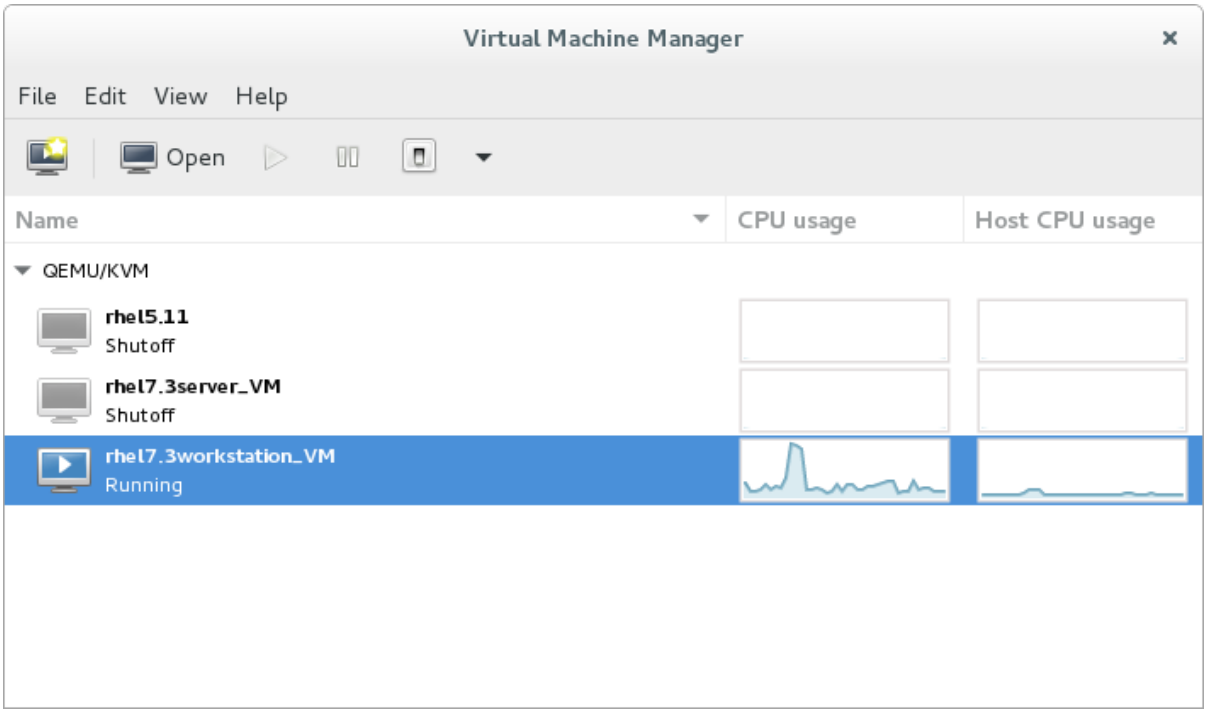
默认的存储池是 /var/lib/libvirt/images/ 目录中的基于文件的镜像。

在客户机虚拟机中添加默认存储：

1. **在虚拟机硬件详情窗口中打开虚拟机管理器**

以 root 用户身份执行 virt-manager 命令打开 virt-manager，或打开 Applications → System Tools → Virtual Machine Manager。

图 13.18. Virtual Machine Manager 窗口



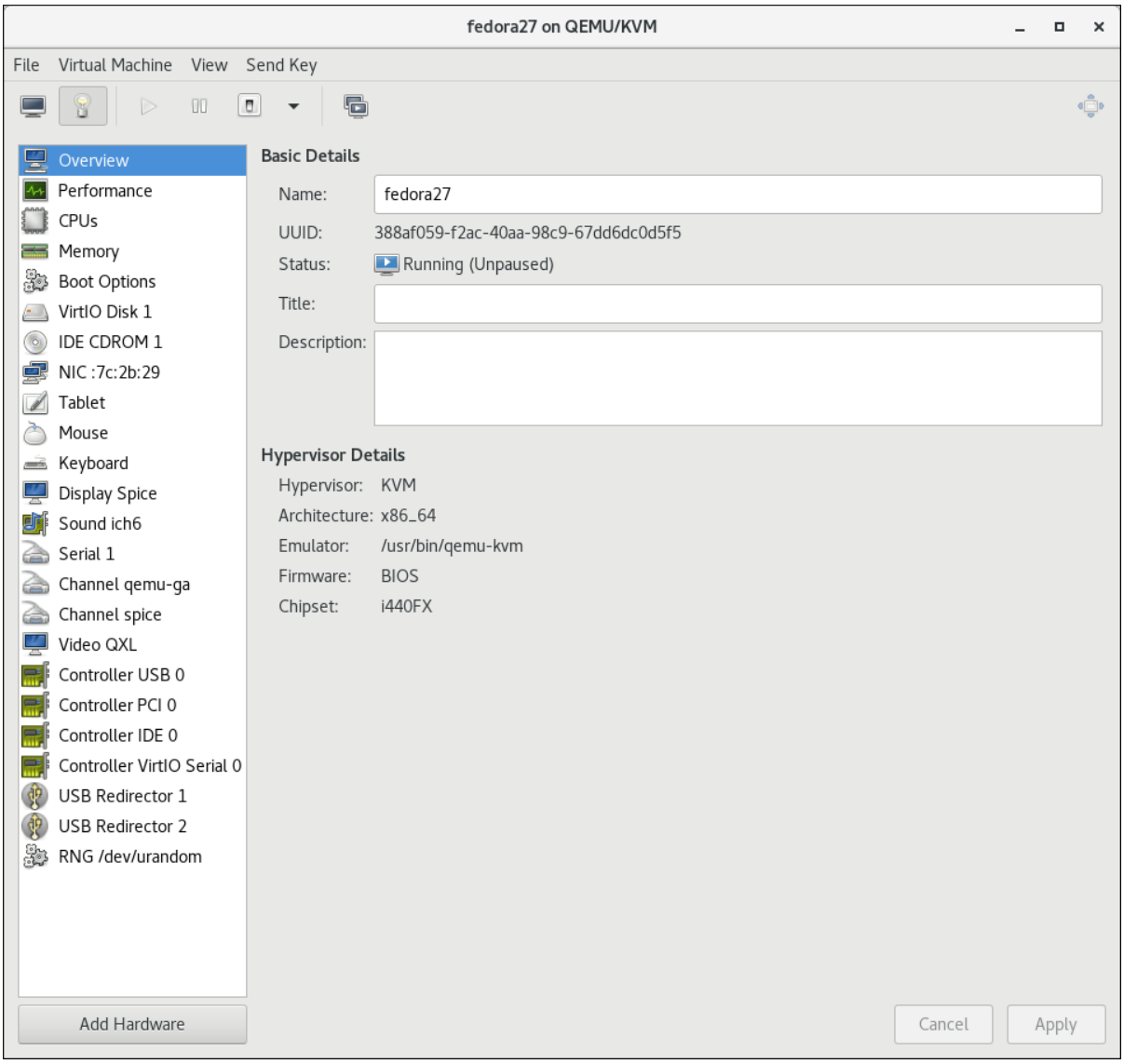
选择您要添加存储卷的客户机虚拟机。

点 Open。此时会打开 Virtual Machine 窗口。

点

。此时会出现硬件详细信息窗口。

图 13.19. 硬件详情窗口

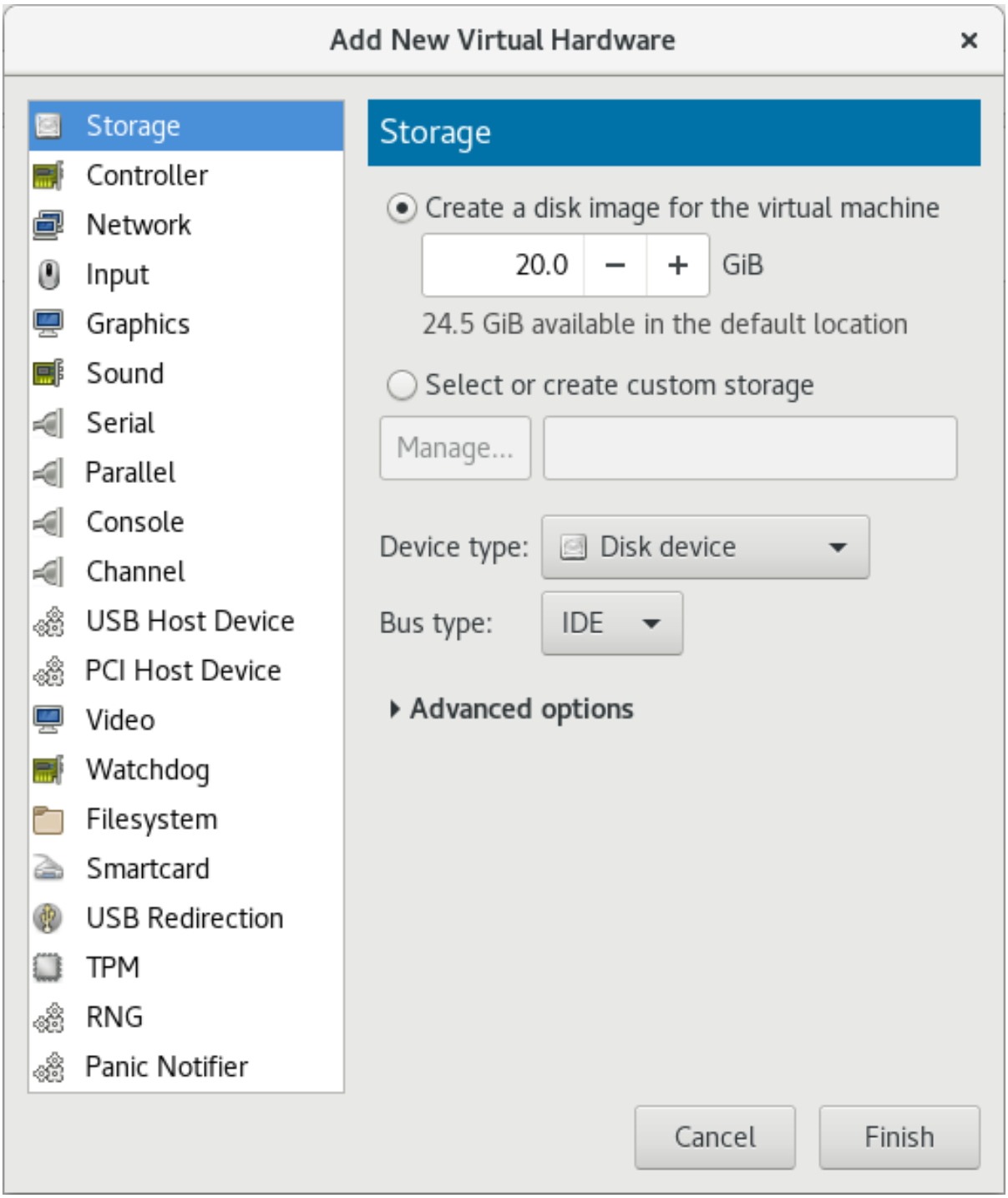


2. 打开 Add New Virtual Hardware 窗口

单击 Add Hardware。此时会出现 Add New Virtual Hardware 窗口。

确定在硬件类型窗格中选择了 **Storage**。

图 13.20. Add New Virtual Hardware 窗口



3. 为客户机创建一个磁盘

   确保为 **虚拟机**选项创建磁盘镜像。

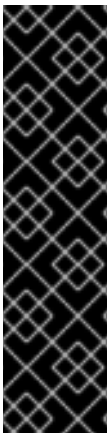
   在为虚拟机选项按钮创建磁盘镜像下面的文本框中输入要创建的磁盘大小。

点 Finish。Add New Virtual Hardware 窗口关闭。

13.3.6.3. 在客户机中添加基于 SCSI LUN 的存储

可以通过多种方式向客户机公开主机 SCSI LUN。将 SCSI LUN 公开给客户机提供了在客户机上直接向 LUN 执行 SCSI 命令的能力。这可很有用，作为在客户机间共享 LUN 的方法，并在主机之间共享光纤通道存储。

有关基于 SCSI LUN 的存储的详情，请参考使用 SCSI 设备 的基于 vHBA 的存储池。



重要

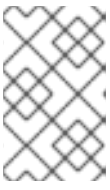
可选的 *sgio* 属性控制是否为 device='lun' 磁盘过滤非特权 SCSI Generical I/O(SG_IO)命令。*sgio* 属性可以指定为 'filtered' 或 'unfiltered'，但必须设置为 'unfiltered' 以允许 SG_IO ioctl 命令在持久预留中通过客户端传递命令。

除了设置 sgio='unfiltered' 外，必须将 *<shareable>* 元素设置为在客户机间共享 LUN。如果没有指定 *sgio* 属性，则默认为 'filtered'。

<disk> XML 属性 device='lun' 可用于以下客户机磁盘配置：

- type='block' for <source dev='/dev/disk/by-{path|id|uuid|label}'/>

```
<disk type='block' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw'/>
  <source dev='/dev/disk/by-path/pci-0000\:04\:00.1-fc-0x203400a0b85ad1d7-lun-0'/>
  <target dev='sda' bus='scsi'/>
  <shareable/>
</disk>
```



注意

<source> 设备名称中的冒号前的反斜杠是必需的。

- type='network' for <source protocol='iscsi'... />

```
<disk type='network' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw'/>
  <source protocol='iscsi' name='iqn.2013-07.com.example:iscsi-net-pool/1'>
    <host name='example.com' port='3260'/>
    <auth username='myuser'>
      <secret type='iscsi' usage='libvirtiscsi'/>
    </auth>
  </source>
  <target dev='sda' bus='scsi'/>
  <shareable/>
</disk>
```

- **当使用 iSCSI 或 NPIV/vHBA 源池作为 SCSI 源池时，type='volume'。**

  以下示例 XML 显示使用 iSCSI 源池（名为 *iscsi-net-pool*）作为 SCSI 源池的客户机：

```
<disk type='volume' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw'/>
  <source pool='iscsi-net-pool' volume='unit:0:0:1' mode='host'/>
  <target dev='sda' bus='scsi'/>
  <shareable/>
</disk>
```

注意

<source> 标签中的 mode= 选项是可选的，但如果使用，则必须设置为 'host'，而不是 'direct'。当设置为 "host" 时，libvirt 将找到本地主机上设备的路径。当设置为 "直接" 时，libvirt 将使用源池的来源主机数据生成到设备的路径。

上面示例中的 iSCSI 池(*iscsi-net-pool*)将具有类似如下的配置：

```
# virsh pool-dumpxml iscsi-net-pool
<pool type='iscsi'>
  <name>iscsi-net-pool</name>
  <capacity unit='bytes'>11274289152</capacity>
  <allocation unit='bytes'>11274289152</allocation>
  <available unit='bytes'>0</available>
  <source>
    <host name='192.168.122.1' port='3260'/>
    <device path='iqn.2013-12.com.example:iscsi-chap-netpool'/>
    <auth type='chap' username='redhat'>
      <secret usage='libvirtiscsi'/>
    </auth>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
```

```
    <permissions>
     <mode>0755</mode>
    </permissions>
  </target>
</pool>
```

要验证 iSCSI 源池中的可用 LUN 的详情，请输入以下命令：

```
# virsh vol-list iscsi-net-pool
Name                 Path
------------------------------------------------------------------------------
unit:0:0:1           /dev/disk/by-path/ip-192.168.122.1:3260-iscsi-iqn.2013-
12.com.example:iscsi-chap-netpool-lun-1
unit:0:0:2           /dev/disk/by-path/ip-192.168.122.1:3260-iscsi-iqn.2013-
12.com.example:iscsi-chap-netpool-lun-2
```

- 当使用 NPIV/vHBA 源池作为 SCSI 源池时，type='volume'。

以下示例 XML 显示使用 NPIV/vHBA 源池（名为 *vhbapool_host3*）作为 SCSI 源池的客户机：

```
<disk type='volume' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw'/>
  <source pool='vhbapool_host3' volume='unit:0:1:0'/>
  <target dev='sda' bus='scsi'/>
  <shareable/>
</disk>
```

上例中的 NPIV/vHBA 池(*vhbapool_host3*)将具有类似的配置：

```
# virsh pool-dumpxml vhbapool_host3
<pool type='scsi'>
  <name>vhbapool_host3</name>
  <capacity unit='bytes'>0</capacity>
  <allocation unit='bytes'>0</allocation>
  <available unit='bytes'>0</available>
  <source>
    <adapter type='fc_host' parent='scsi_host3' managed='yes' wwnn='5001a4a93526d0a1'
wwpn='5001a4ace3ee045d'/>
  </source>
  <target>
    <path>/dev/disk/by-path</path>
    <permissions>
      <mode>0700</mode>
      <owner>0</owner>
      <group>0</group>
```

```
    </permissions>
  </target>
</pool>
```

要验证 vHBA 上可用 LUN 的详情，请输入以下命令：

```
# virsh vol-list vhbapool_host3
Name                 Path
------------------------------------------------------------------------------
unit:0:0:0           /dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016044602198-lun-0
unit:0:1:0           /dev/disk/by-path/pci-0000:10:00.0-fc-0x5006016844602198-lun-0
```

有关在 SCSI 设备中使用 NPIV vHBA 的详情，请参考 第 13.2.3.8 节 “使用 SCSI 设备使用基于 vHBA 的存储池”。

以下流程演示了在客户机中添加基于 SCSI LUN 的存储设备的示例。以上 *<disk device='lun'>* 客户机磁盘配置都可以使用这个方法附加。根据您的环境替换配置。

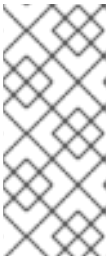
过程 13.13. 将基于 SCSI LUN 的存储附加到客户端

1. 通过在新文件中写入 <disk> 元素来创建设备文件，并使用 XML 扩展（本例中为 *sda.xml*）保存此文件：

```
# cat sda.xml
<disk type='volume' device='lun' sgio='unfiltered'>
  <driver name='qemu' type='raw'/>
  <source pool='vhbapool_host3' volume='unit:0:1:0'/>
  <target dev='sda' bus='scsi'/>
  <shareable/>
</disk>
```

2. 将 *sda.xml* 中创建的设备与您的虚拟客户机相关联（例如：

```
# virsh attach-device --config Guest1 ~/sda.xml
```

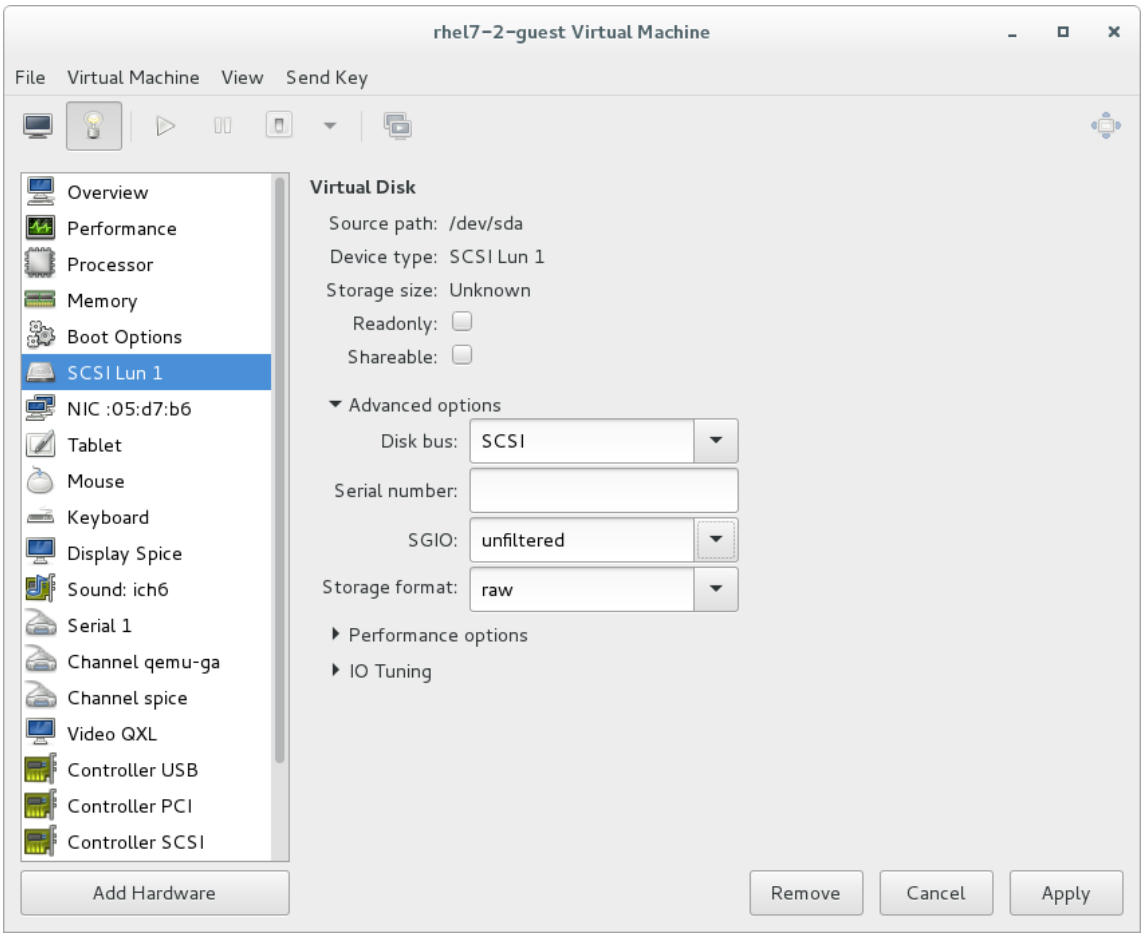


注意

使用 --config 选项运行 virsh attach-device 命令，需要 guest reboot 将设备永久添加到客户端。或者，可以使用 --persistent 选项而不是 --config，它也可用于将设备热插到客户机中。

另外，可使用 virt-manager 在客户机中附加或配置基于 SCSI LUN 的存储。要使用 virt-manager 进行配置，请点击 Add Hardware 按钮并添加带有所需参数的虚拟磁盘，或者从此窗口中更改现有 SCSI LUN 设备的设置。在 Red Hat Enterprise Linux 7.2 及更高版本中，SGIO 值也可以在 virt-manager 中配置：

图 13.21. 使用 virt-manager 配置 SCSI LUN 存储



硬件失败后重新连接到公开的 LUN

如果因为硬件失败（如主机总线适配器），与公开的光纤通道(FC)LUN 的连接会丢失，则客户机上公开的 LUN 也可能会继续显示为故障，即使硬件故障也是如此。要防止这种情况，请编辑 dev_loss_tmo 和 fast_io_fail_tmo 内核选项：

- dev_loss_tmo 控制 SCSI 层在将 SCSI 设备标记为失败前等待的时间。要防止超时，建议将 选项设置为最大值，即 2147483647。

- fast_io_fail_tmo 控制 SCSI 层在失败的 SCSI 设备失败后等待多久，然后再返回 I/O。为确保内核不忽略 dev_loss_tmo，请将这个选项的值设置为小于 dev_loss_tmo 的值的任意数值。

要修改 dev_loss_tmo 和 fast_io_fail 的值，请执行以下操作之一：

- 编辑 /etc/multipath.conf 文件，并在 defaults 部分中设置值：

```
defaults {
...
fast_io_fail_tmo     20
dev_loss_tmo    infinity
}
```

- 在 FC 主机或远程端口级别设置 dev_loss_tmo 和 fast_io_fail，例如：

```
# echo 20 > /sys/devices/pci0000:00/0000:00:06.0/0000:13:00.0/host1/rport-1:0-0/fc_remote_ports/rport-1:0-0/fast_io_fail_tmo
# echo 2147483647 > /sys/devices/pci0000:00/0000:00:06.0/0000:13:00.0/host1/rport-1:0-0/fc_remote_ports/rport-1:0-0/dev_loss_tmo
```

要验证 dev_loss_tmo 和 fast_io_fail 的新值是否活跃，请使用以下命令：

```
# find /sys -name dev_loss_tmo -print -exec cat {} \;
```

如果正确设置了参数，输出将类似如下，使用适当的设备或设备而不是 pci0000:00/0000:00:06.0/0000:13:00.0/host1/rport-1:0-0/fc_remote_ports/rport-1:0-0:

```
# find /sys -name dev_loss_tmo -print -exec cat {} \;
...
/sys/devices/pci0000:00/0000:00:06.0/0000:13:00.0/host1/rport-1:0-0/fc_remote_ports/rport-1:0-0/dev_loss_tmo
2147483647
...
```

13.3.6.4. 在客户机虚拟机中管理存储控制器

与 virtio 磁盘不同，SCSI 设备需要在客户机虚拟机中存在控制器。本节详细介绍了创建虚拟 SCSI 控制器（也称为"主机总线适配器"或 HBA）以及向客户机虚拟机添加 SCSI 存储所需的步骤。

过程 13.14. 创建虚拟 SCSI 控制器

1. 显示客户机虚拟机(Guest1)的配置，并查找已存在的 SCSI 控制器：

```
# virsh dumpxml Guest1 | grep controller.*scsi
```

如果存在设备控制器，命令将输出一个或多个类似如下的行：

```
<controller type='scsi' model='virtio-scsi' index='0'/>
```

2. 如果上一步没有显示设备控制器，请在一个新文件中创建一个描述，并使用以下步骤将其添加到虚拟机中：

   a. 通过在新文件中写入 **<controller>** 元素来创建设备控制器，并使用 **XML** 扩展名保存此文件。**virtio-scsi-controller.xml**，例如：

   ```
   <controller type='scsi' model='virtio-scsi'/>
   ```

   b. 将您在 **virtio-scsi-controller.xml** 中创建的设备控制器与您的虚拟客户机（例如，**Guest1**）关联：

   ```
   # virsh attach-device --config Guest1 ~/virtio-scsi-controller.xml
   ```

   在本例中，**--config** 选项的行为与磁盘的作用相同。如需更多信息，请参阅 第 13.3.6 节 “在客户机中添加存储设备”。

3. 添加新的 SCSI 磁盘或 CD-ROM。可使用 第 13.3.6 节 “在客户机中添加存储设备” 中的方法添加新磁盘。要创建 SCSI 磁盘，请指定以 **sd** 开头的目标设备名称。

注意

每个控制器的支持限制为 1024 virtio-scsi 磁盘，但该主机中的其他可用资源（如文件描述符）可能会用较少的磁盘耗尽。

如需更多信息，请参阅以下 Red Hat Enterprise Linux 6 白皮书：红帽企业 Linux 内核虚拟机下一代存储接口：virtio-scsi.

```
# virsh attach-disk Guest1 /var/lib/libvirt/images/FileName.img sdb --cache none
```

根据客户机虚拟机中的驱动程序版本，运行的客户机虚拟机可能不会立即检测到新磁盘。按照《Red Hat Enterprise Linux 存储管理指南》中的步骤操作。

13.3.7. 从客户机中删除存储设备

您可以使用 virsh 或 Virtual Machine Manager 从虚拟机中删除存储设备。

13.3.7.1. 使用 virsh 从虚拟机中删除存储

以下示例从 Guest1 虚拟机中删除 vdb 存储卷：

```
# virsh detach-disk Guest1 vdb
```

13.3.7.2. 使用虚拟机管理器从虚拟机中删除存储

过程 13.15. 使用虚拟机管理器从虚拟机中删除存储

使用虚拟机管理器从客户机虚拟机中删除存储：

1. 在虚拟机硬件详情窗口中打开虚拟机管理器

   以 root 用户身份执行 virt-manager 命令打开 virt-manager，或打开 Applications → System Tools → Virtual Machine Manager。

   选择您要从中删除存储设备的客户机虚拟机。

   点 Open。此时会打开 Virtual Machine 窗口。

   点  。此时会出现硬件详细信息窗口。

2. 从客户机虚拟机中删除存储

   从硬件详细信息窗格左侧的硬件列表中选择存储设备。

单击 **Remove。此时会出现确认对话框。**

单击 **"是"存储已从客户机虚拟机中删除。**

第 14 章 使用 QEMU-IMG

qemu-img 命令行工具用于格式化、修改和验证 KVM. qemu-img 选项和用法所使用的各种文件系统，后续部分将突出显示。

警告

不要使用 qemu-img 修改正在运行的虚拟机或任何其他进程所使用的镜像。这可能会销毁镜像。另外，请注意，查询被其他进程修改的镜像可能会遇到不一致的状态。

14.1. 检查磁盘镜像

要在磁盘镜像上执行一致性检查，其文件名为 imgname。

```
# qemu-img check [-f format] imgname
```

注意

只有所选格式组支持一致性检查。其中包括 qcow2、vdi、vhdx、vmdk 和 qed。

14.2. 将更改提交至镜像

使用 qemu-img commit 命令，将指定镜像文件(imgname)中记录的任何更改提交到文件的基础镜像。（可选）指定文件格式类型(fmt)。

```
# qemu-img commit [-f fmt] [-t cache] imgname
```

14.3. 镜像比较

将两个指定的镜像文件的内容（imgname1 和 img name2）与 qemu-img compare 命令进行比较。（可选）指定文件格式类型(fmt)。镜像可以有不同的格式和设置。

默认情况下，如果较大的镜像在另一个镜像后面只包含这个区域中的未分配或者零个扇区，则具有不

同大小的镜像被视为相同。另外，如果一个镜像中没有分配任何扇区，且在另一个镜像中仅包含零字节，则会尽可能高。如果您指定了 **-s** 选项，如果镜像大小不同，或者某个扇区在一个镜像中被分配，则镜像不会被视为相同。

```
# qemu-img compare [-f fmt] [-F fmt] [-p] [-s] [-q] imgname1 imgname2
```

qemu-img 比较 命令退出了以下退出代码之一：

- **0 -** 镜像是相同的
- **1 -** 镜像是不同的
- **2 -** 打开其中一个镜像时出错
- **3 -** 检查扇区分配时出错
- **4 -** 读取数据时出错

14.4. 映射镜像

使用 **qemu-img map** 命令，您可以转储指定镜像文件的元数据(**imgname**)及其后备文件链。转储显示(**imgname**)中每个扇区的分配状态，其最高的文件在后备文件链中分配。（可选）指定文件格式类型(**fmt**)。

```
# qemu-img map [-f fmt] [--output=fmt] imgname
```

输出格式有两种，即 人类可读的格式 和 **json** 格式：

14.4.1. 人类 格式

默认格式（人）只转储文件的非零分配部分。输出中标识了可从其中读取数据和文件中偏移的文件。每行都包括四个字段。下面是一个输出的示例：

```
Offset          Length          Mapped to       File
0               0x20000         0x50000         /tmp/overlay.qcow2
0x100000        0x10000         0x95380000      /tmp/backing.qcow2
```

第一行表示，从镜像偏移 0 开始从偏移 0 开始 0x20000 (131072)字节，位于 tmp/overlay.qcow2（以原始格式打开）从偏移 0x50000 (327680)开始。如果指定 人 格式，则压缩、加密或不以原始格式提供的数据会导致错误。

注意

文件名可以包括换行符。因此，在脚本中无法以 人类可读格式 解析输出。

14.4.2. json 格式

如果指定了 json 选项，输出会以 JSON 格式返回数组字典。除了 人工 选项提供的信息外，输出还包括以下信息：

- data - 显示扇区是否包含数据的布尔值字段
- 零 - 显示数据是否已知为零的布尔值字段
- 深度 - 后备 文件名的深度

注意

当指定 json 选项时，偏移 字段是可选的。

有关 qemu-img map 命令和附加选项的详情，请查看相关的 man page。

14.5. 镜像(MENDING)

修正镜像文件相关镜像格式的选项。（可选）指定文件格式类型(fmt)。

```
# qemu-img amend [-p] [-f fmt] [-t cache] -o options filename
```

注意

此操作仅支持 qcow2 文件格式。

14.6. 将现有镜像转换为另一个格式

convert 选项用于将一个可识别的镜像格式转换为另一个镜像格式。有关接受格式列表，请参阅 第 14.12 节 “支持的 qemu-img 格式”。

```
# qemu-img convert [-c] [-p] [-f fmt] [-t cache] [-O output_fmt] [-o options] [-S sparse_size] filename
output_filename
```

p 参数显示命令的进度（可选，而不是每个命令）和 -S 标志允许创建 稀疏文件，该文件包含在磁盘镜像中。所有目的中的稀疏文件（即，标准文件）都类似标准文件，但物理块只能包含零（也就是说没有）。当操作系统看到此文件时，它会将其视为存在，它会占用实际的磁盘空间，即使实际并不会承担任何文件。这在为客户机虚拟机创建磁盘时特别有用，因为这表明磁盘占用的磁盘空间比它更多的磁盘空间要多。例如，如果您在磁盘镜像上设置 -S 设置为 50Gb，那么您的 10Gb 磁盘空间将显示 60Gb，即使实际使用了 10Gb。

使用格式 output_format 将磁盘镜像 文件名 转换为磁盘镜像 output_filename。磁盘镜像可以选择使用 -c 选项压缩，或通过设置 -o encryption 来使用 -o 选项加密。请注意，可用的选项有 -o 参数与所选格式有所不同。

只有 qcow2 和 qcow2 格式支持加密或压缩。qcow2 加密使用带有安全 128 位键的 AES 格式。qcow2 压缩为只读，因此如果压缩的扇区从 qcow2 格式转换，它会将新格式写成未压缩数据。

使用可增大的格式（如 qcow 或 cow ）时，镜像转换也可用于获得较小的镜像。检测到空白扇区，并从目标镜像中抑制。

14.7. 创建并格式化新镜像或设备

创建 大小 的新磁盘镜像 文件名，格式为。

```
# qemu-img create [-f format] [-o options] filename [size]
```

-

如果使用 -o backing_file=filename 指定基础镜像，则该镜像将只记录自身和基础镜像之间的区别。除非使用 commit 命令，否则不会修改 后备文件。在这种情况下不需要指定任何大小。

14.8. 显示镜像信息

info 参数显示有关磁盘映像 文件名 的信息。info 选项的格式如下：

```
# qemu-img info [-f format] filename
```

此命令通常用于发现磁盘上保留的大小，这些大小可能与显示的大小不同。如果快照存储在磁盘镜像中，它们也会显示它们。例如，此命令将显示块设备中 qcow2 镜像所占用的空间量。这可以通过运行 qemu-img 来完成。您可以使用 qemu-img check 命令来检查所使用的镜像是否与 qemu-img info 命令的输出相匹配。

```
# qemu-img info /dev/vg-90.100-sluo/lv-90-100-sluo
image: /dev/vg-90.100-sluo/lv-90-100-sluo
file format: qcow2
virtual size: 20G (21474836480 bytes)
disk size: 0
cluster_size: 65536
```

14.9. 重新标记镜像的备份文件

qemu-img rebase 更改镜像的备份文件。

```
# qemu-img rebase [-f fmt] [-t cache] [-p] [-u] -b backing_file [-F backing_fmt] filename
```

后备文件被改为 backing_file，如果 文件名 格式支持该功能，支持的文件格式将更改为 backing_format。

注意

只有 qcow2 格式支持更改后备文件(rebase)。

rebase 可操作的模式 有两种，即 安全和 不安全。

默认使用 安全模式 并执行实际重基操作。新的后备文件可能与旧文件不同，qemu-img rebase 命令

将注意，使客户机虚拟机可见 的文件名 内容保持不变。为了实现此目的，在对 后备_file 和旧备份文件更改前，任何不同的集群均将合并到 文件名 中。

请注意，安全模式 是昂贵的操作，与转换镜像相当。需要旧的备份文件才能成功完成。

如果 -u 选项传递给 qemu-img rebase，则使用 不安全 模式。在这个模式中，仅更改后备文件名和 文件名 格式，而无需对文件内容执行任何检查。确保正确指定新的后备文件，或者镜像的 guestvisible 内容将损坏。

此模式可用于重命名或移动后备文件。它可在不需要访问的旧备份文件的情况下使用。例如，它可用于修复已移动或重命名备份文件的镜像。

14.10. 重新大小磁盘镜像

更改磁盘镜像 文件名，就好像创建 大小为 一样。同一方向只能调整 raw 格式的镜像，而 qcow2 镜像则可增大，但不能增大，但无法缩小。

使用以下内容将磁盘镜像 文件名 的大小设置为 size 字节：

```
# qemu-img resize filename size
```

您还可以调整与磁盘镜像当前大小相关的大小。要指定相对于当前大小的大小，请为要增大的字节数添加前缀，或者 - 为磁盘镜像的大小减小到这个字节数。通过添加单元后缀，您可以使用 KB(K)、兆字节(M)、千兆字节(G)或 TB(T)设置镜像大小。

```
# qemu-img resize filename [+|-]size[K|M|G|T]
```

警告

在使用这个命令缩小磁盘镜像前，您必须在虚拟机本身中使用文件系统和分区工具来减少分配的文件系统和分区大小。否则将导致数据丢失。

在使用这个命令增加磁盘镜像后，您必须使用虚拟机中的文件系统和分区工具来实际使用该设备中的新空间。

14.11. 列出、创建、应用和删除快照

使用与 qemu-img snapshot 命令不同的参数，您可以列出、应用、创建或删除指定映像的现有快照（快照）（文件名）。

```
# qemu-img snapshot [ -l | -a snapshot | -c snapshot | -d snapshot ] filename
```

接受的参数如下：

- -L 列出与指定磁盘镜像关联的所有快照。
- apply 选项 -a 将磁盘镜像（文件名）恢复到之前保存 的快照 的状态。
- -c 创建映像的快照（快照）（文件名）。
- -d 删除指定的快照。

14.12. 支持的 QEMU-IMG 格式

当在任何 qemu-img 命令中指定格式时，可以使用以下格式类型：

- Raw 磁盘镜像格式（默认）。这可以是基于文件的速度最快的格式。如果您的文件系统支持漏洞（例如 ext2 或 ext3），则只有写入的扇区会保留空间。使用 qemu-img info 获取镜像使用

的实际大小或 ls -ls on Unix/Linux。虽然 Raw 镜像提供最佳性能，但只有 Raw 镜像只提供非常基本的功能。例如，没有快照可用。

- qcow2 - QEMU 镜像格式，具有最佳功能集，最常用的格式。使用它具有可选的 AES 加密、基于 zlib 的压缩、支持多个虚拟机快照和较小的镜像，它们对不支持漏洞的文件系统很有用。请注意，这种丰富的功能集的性能成本如下。

  虽然只有上述格式可以在客户机虚拟机或主机物理机器中运行，但 qemu-img 也会识别并支持下列格式，以便从 原始、或 qcow2 格式转换。通常会自动检测到镜像的格式。除了将这些格式转换为 raw 或 qcow2 外，它们还可以从 raw 或 qcow2 转换回原始格式。请注意，Red Hat Enterprise Linux 7 提供的 qcow2 版本为 1.1。与之前版本的 Red Hat Enterprise Linux 提供的格式为 0.10。您可以将镜像文件恢复到以前的 qcow2 版本。要了解您使用的版本，请运行 qemu-img info qcow2 [imagefilename.img] 命令。要更改 qcow 版本，请参阅 第 23.19.2 节 “设置目标元素”。

- Bochs - 磁盘映像格式.

- cloop - Linux Compressed Loop 镜像，对于仅在 Knoppix CD-ROMs 中重复使用直接压缩 CD-ROM 镜像。

- COW - User Mode Linux Copy On Write image format。cow 格式包含在与之前的版本兼容。

- dmg - Mac 磁盘镜像格式。

- nbd - 网络块设备。

- Parallels - Parallels 虚拟化磁盘镜像格式。

- QCOW - Old QEMU 镜像格式。仅用于与旧版本兼容。

- qed - Old QEMU 镜像格式。仅用于与旧版本兼容。

- VDI - Oracle VM VirtualBox 硬盘镜像格式。

- ***VHD X - Microsoft Hyper-V 虚拟硬盘-X 磁盘映像格式.***

- ***VMDK - VMware 3 和 4 兼容镜像格式。***

- ***vvfat - 虚拟 VFAT 磁盘镜像格式。***

第 15 章 KVM 迁移

本章论述了将客户机虚拟机从一个主机物理机器迁移到另一台主机。迁移 guest 可能会因为虚拟机在虚拟环境中运行，而不是直接在硬件上运行。

15.1. 迁移定义和优点

迁移工作是将客户机虚拟机内存和任何虚拟设备的状态发送到目标主机物理机器。建议使用共享、联网的存储来存储要迁移的客户机镜像。另外，还建议在迁移虚拟机时使用 libvirt 管理 的存储池 进行共享存储。

可通过 实时 （运行）和 非实时迁移 （下线）客户机执行迁移。

在 实时迁移 中，客户机虚拟机将继续在源主机上运行，而客户机的内存页面传输到目标主机计算机。在迁移过程中，KVM 会监控已传输页面中的任何更改，并在所有初始页面都已转移后开始转移这些更改。KVM 还估计迁移期间的传输速度，因此当剩余的数据传输量达到一定可配置的时间段（默认为 10ms），KVM 暂停原始客户机虚拟机，传输剩余的数据，并在目标主机物理机器上恢复相同的客户机虚拟机。

相反，非实时迁移 （离线迁移）会挂起客户机虚拟机，然后将客户机内存复制到目标主机。然后，客户机会在目标主机上恢复，并且释放了源主机机器中使用的客户机的内存。完成此类迁移所需的时间取决于网络带宽和延迟。如果网络遇到大量使用或低带宽，迁移将花费更长的时间。请注意，如果原始客户机虚拟机修改页的速度比 KVM 可以将其传送到目标主机物理机器更快，则必须使用离线迁移，因为实时迁移永远不会完成。

迁移可用于：

负载均衡

如果 guest 虚拟机能够被过度加载，或者另一台主机机器被过度使用，则可以将客户机虚拟机移动到使用较低的主机机器中。

硬件独立

当您需要升级、添加或删除主机物理机器上时，您可以安全地将客户机虚拟机重新定位到其他主机物理机器。这意味着客户机虚拟机在改进硬件时不会遇到任何停机时间。

节能

虚拟机可以重新分发到其他主机的物理计算机，因此可以关闭未卸载的主机系统，从而在低使用

阶段节省能源并降低成本。

地理迁移

虚拟机可以移到另一个位置，以降低延迟或者因为其他原因需要。

15.2. 迁移要求和限制

在使用 KVM 迁移前，请确定您的系统满足迁移的要求，并了解其限制。

迁移要求

- 使用下列协议之一在共享存储中安装客户机虚拟机：
  - 基于 Fibre Channel 的 LUN
  - iSCSI
  - NFS
  - GFS2
  - SCSI RDMA 协议(SCSI RCP)：Infiniband 和 10GbE iWARP 适配器中使用的块导出协议
- 确保 libvirtd 服务已启用并在运行。

```
# systemctl enable libvirtd.service
# systemctl restart libvirtd.service
```

- 有效迁移的能力取决于 /etc/libvirt/libvirtd.conf 文件中的参数设置。要编辑此文件，请使用以下步骤：

过程 15.1. 配置 libvirtd.conf

1. 打开 libvirtd.conf 需要以 root 身份运行该命令：

```
# vim /etc/libvirt/libvirtd.conf
```

2. 根据需要更改参数并保存文件。

3. 重启 libvirtd 服务：

```
# systemctl restart libvirtd
```

- 应该根据情况检查迁移平台和版本 表 15.1 “实时迁移兼容性”

- 使用单独的系统导出共享存储介质。存储不应位于用于迁移的两个主机物理机器上。

- 共享存储必须在源和目标系统上挂载在同一位置。挂载的目录名称必须相同。虽然可以使用不同路径保留镜像，但不推荐这样做。请注意，如果您打算使用 virt-manager 执行迁移，则路径名称必须相同。如果要使用 virsh 执行迁移，可以在 --xml 选项或 pre-hooks 中使用不同的网络配置和挂载目录 。有关 pre-hooks 的更多信息，请参阅 libvirt 上游文档，以及 XML 选项的详情，请参考 第 23 章 操作域 XML。

- 当在公共 bridge+tap 网络中的现有客户机虚拟机上尝试迁移时，源和目标主机必须位于同一网络中。否则，迁移后，客户机虚拟机网络将不会操作。

迁移限制

- 在 Red Hat Enterprise Linux 中使用基于 KVM 的虚拟化技术时，客户机虚拟机迁移有以下限制：

  - 指向迁移 - 必须手动操作到指定来自原始虚拟机监控程序的目标管理程序

  - 没有验证或回滚可用

- 只能手动确定目标

- 无法在 Red Hat Enterprise Linux 7™ 上执行存储迁移，但您可以在客户机虚拟机关闭时迁移存储。在 Red Hat Virtualization™ 中提供了实时存储迁移。详情请致电您的服务代表。

注意

如果您要将带有 virtio 设备的客户机机器迁移到其中，请确保将任意平台上的 virtio 设备上的向量设置为 32 个或更少。详情请查看 第 23.17 节 “Devices”。

15.3. 实时迁移和 RED HAT ENTERPRISE LINUX 版本兼容性

实时迁移在 表 15.1 “实时迁移兼容性” 所示被支持：

表 15.1. 实时迁移兼容性

迁移方法	发行类型	示例	实时迁移支持	备注
向前	主发行版本	6.5+ → 7.x	完全支持	应该报告任何问题
向后	主发行版本	7.x → 6.y	不支持	
向前	次发行版本	7.x → 7.y (7.0 → 7.1)	完全支持	应该报告任何问题
向后	次发行版本	7.y → 7.x (7.1 → 7.0)	完全支持	应该报告任何问题

迁移故障排除

- 迁移协议的问题 - 如果向后兼容以"未知部分错误"结尾，重复迁移过程可修复该问题，因为它可能是一个瞬态错误。如果不是，请报告问题。

- 音频设备问题 - 从 Red Hat Enterprise Linux 6.x 迁移到 Red Hat Enterprise Linux 7.y 时，es1370 音频卡不再被支持。改为使用 ac97 音频卡。

- 网卡的问题 - 当从 Red Hat Enterprise Linux 6.x 迁移到 Red Hat Enterprise Linux 7.y 时，不再支持 pcnet 和 ne2k_pci 网卡。改为使用 virtio-net 网络设备。

配置网络存储

配置共享存储并在共享存储上安装客户机虚拟机。

或者，使用 NFS 示例 第 15.4 节 “共享存储示例：用于简单迁移的 NFS”

15.4. 共享存储示例：用于简单迁移的 NFS

重要

这个示例使用 NFS 与其他 KVM 主机物理机器共享客户机虚拟机镜像。虽然大型安装不实际，但介绍仅演示迁移技巧。请勿使用此示例来迁移或运行多个客户机虚拟机。另外，还需要启用 **synch** 参数。这是正确导出 NFS 存储所需要的。

对于大型部署来说，iSCSI 存储是更好的选择。有关配置详情请参考 第 13.2.3.5 节 “基于 iSCSI 的存储池”。

有关配置 NFS、打开 IP 表和配置防火墙的详情，请参考 Red Hat Linux Storage Administration Guide。

请确定 NFS 文件锁定没有被使用，因为在 KVM 中不支持它。

1. 导出 libvirt 镜像目录

   迁移需要存储位于独立于迁移目标系统的系统中。在这个独立系统中，通过将默认镜像目录添加到 /etc/exports 文件中来导出存储：

   ```
   /var/lib/libvirt/images *.example.com(rw,no_root_squash,sync)
   ```

   根据您的环境要求更改 **hostname** 参数。

2. 启动 NFS

   a. 如果 NFS 软件包尚未安装，则安装它们：

   ```
   # yum install nfs-utils
   ```

b.

确保打开 iptables 中 NFS 的端口（例如 2049），并将 NFS 添加到 /etc/hosts.allow 文件中。

c.

启动 NFS 服务：

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

3. 将共享存储挂载到源和目的地

在迁移源和目标系统中挂载 /var/lib/libvirt/images 目录：

```
# mount storage_host:/var/lib/libvirt/images /var/lib/libvirt/images
```

警告

为源主机物理机器选择哪个目录都必须与目标主机物理计算机上的目录完全相同。这适用于所有共享存储。该目录必须相同，或使用 virt-manager 迁移会失败。

15.5. 使用 VIRSH 进行实时 KVM 迁移

可以使用 virsh 命令将客户机虚拟机迁移到另一台主机物理计算机。migrate 命令接受以下格式的参数：

```
# virsh migrate --live GuestName DestinationURL
```

请注意，不需要实时迁移时可以消除 --live 选项。其它选项在 第 15.5.2 节 “virsh migrate 命令的附加选项” 中列出。

GuestName 参数表示您要迁移的客户机虚拟机的名称。

DestinationURL 参数是目标主机物理机器的连接 URL。目标系统必须运行相同的 Red Hat Enterprise Linux 版本，它们使用相同的管理程序，并且运行 libvirt。

注意

常规迁移的 DestinationURL 参数和 peer2peer 迁移有不同的语义：

- *正常迁移： DestinationURL 是目标主机物理机器的 URL，如源客户机虚拟机所示。*
- *peer2peer 迁移：DestinationURL 是目标主机物理计算机的 URL，如源主机物理计算机所示。*

输入命令后，系统将提示您输入目标系统的 root 密码。

重要

名称解析必须同时处理（源和目标），才能成功迁移。每个产品都必须能够找到其他内容。请确定您可以 ping 一边，以检查名称解析是否正常工作。

示例：使用 virsh 实时迁移

这个示例从 host1.example.com 迁移到 host2.example.com。更改您的环境的主机物理机器名称。本例迁移了一个名为 guest1-rhel6-64 的虚拟机。

这个示例假设您已经完全配置了共享存储，并满足所有先决条件（以下列出： 迁移要求）。

1. *验证客户机虚拟机正在运行*

   在源系统中 host1.example.com，验证 guest1-rhel6-64 正在运行：

```
[root@host1 ~]# virsh list
Id Name                 State
----------------------------------
 10 guest1-rhel6-64     running
```

2. *迁移客户端虚拟机*

   执行以下命令，将 guest 虚拟机实时迁移到目标 host2.example.com。在目标 URL 的末尾附加 /system 以便告知 libvirt 您需要完全访问。

```
# virsh migrate --live guest1-rhel7-64 qemu+ssh://host2.example.com/system
```

输入命令后，系统将提示您输入目标系统的 root 密码。

3. **Wait**

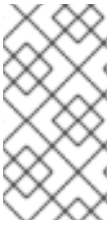
根据负载和客户机虚拟机大小，迁移可能需要一些时间。virsh 只报告错误。客户机虚拟机继续在源主机物理机器中运行，直到完全迁移为止。

4. 验证客户机虚拟机已到达目标主机

从目标系统 host2.example.com，验证 guest1-rhel7-64 正在运行：

```
[root@host2 ~]# virsh list
 Id Name                 State
----------------------------------
 10 guest1-rhel7-64      running
```

实时迁移现已完成。



注意

libvirt 支持各种网络方法，包括 TLS/SSL、UNIX 套接字、SSH 和未加密 TCP。有关使用其它方法的详情请参考 第 18 章 客户机的远程管理。

注意

使用以下命令可迁移非运行的客户机虚拟机：

```
# virsh migrate --offline --persistent
```

15.5.1. 使用 virsh 迁移的附加提示

在单独的命令 shell 中运行每个迁移，可以执行多个并发实时迁移。但是，这应该非常谨慎，并应该涉及计算每个迁移实例每个端的 MAX_CLIENT（源和目标）。由于默认设置是 20，因此只需更改设置即可运行 10 个实例。如果您需要更改设置，请查看 过程 15.1, “配置 libvirtd.conf” 过程。

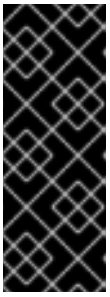
1. 打开 **libvirtd.conf** 文件，如 *过程 15.1, “配置 libvirtd.conf”* 所述。

2. 查找处理控制部分。

```
#################################################################
#
# Processing controls
#

# The maximum number of concurrent client connections to allow
# over all sockets combined.
#max_clients = 5000

# The maximum length of queue of connections waiting to be
# accepted by the daemon. Note, that some protocols supporting
# retransmission may obey this so that a later reattempt at
# connection succeeds.
#max_queued_clients = 1000

# The minimum limit sets the number of workers to start up
# initially. If the number of active clients exceeds this,
# then more threads are spawned, upto max_workers limit.
# Typically you'd want max_workers to equal maximum number
# of clients allowed
#min_workers = 5
#max_workers = 20


# The number of priority workers. If all workers from above
# pool will stuck, some calls marked as high priority
# (notably domainDestroy) can be executed in this pool.
#prio_workers = 5

# Total global limit on concurrent RPC calls. Should be
# at least as large as max_workers. Beyond this, RPC requests
# will be read into memory and queued. This directly impact
# memory usage, currently each request requires 256 KB of
# memory. So by default upto 5 MB of memory is used
#
# XXX this isn't actually enforced yet, only the per-client
# limit is used so far
#max_requests = 20

# Limit on concurrent requests from a single client
# connection. To avoid one client monopolizing the server
# this should be a small fraction of the global max_requests
# and max_workers parameter
#max_client_requests = 5

#################################################################
```

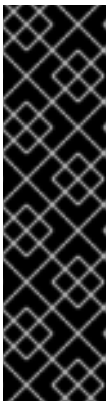
3.

更改 max_clients 和 max_workers 参数设置。建议两个参数中的数字都相同。max_clients 每次迁移过程中将使用 2 个客户端（每个迁移一个），max_workers 则在执行阶段，目标上的 1 个 worker 使用 1 个 worker，并在完成阶段在目标 1 个 worker 上使用 1 个 worker。



重要

max_clients 和 max_workers 参数设置会受到连接到 libvirtd 服务的所有客户机虚拟机连接的影响。这意味着，任何正在使用同一客户机虚拟机并同时执行迁移的用户也会遵循 max_clients 和 max_workers 参数设置的限制。这就是为什么在执行并发实时迁移之前，需要仔细考虑最大值的原因。



重要

max_clients 参数控制允许多少个客户端连接到 libvirt。当同时启动大量容器时，可轻松达到和超过这一限制。max_clients 参数的值可以增加以避免这种情况，但这样做会使系统更易受到攻击，从而防止系统遭到拒绝服务(DoS)攻击。要解决这个问题，Red Hat Enterprise Linux 7.0 中引入了一个新的 max_anonymous_clients 设置，用于指定接受但尚未通过身份验证的连接的限制。您可以实施 max_clients 和 max_anonymous_clients 的组合来适合您的工作负载。

4. 保存文件并重启该服务。

注意

有些情况下，迁移连接会下降，因为已启动但尚未通过身份验证的 ssh 会话太多。默认情况下，sshd 只允许 10 会话随时处于 "pre-authenticated state"。此设置由 sshd 配置文件中的 MaxStartups 参数（位于：/etc/ssh/sshd_config）来控制，可能需要进行一些调整。需要小心调整此参数，因为限制可以防止 DoS 攻击（一般情况下使用资源）。将此值设置为过高，将指示其用途。要更改这个参数，请编辑文件 /etc/ssh/sshd_config，从 MaxStartups 行的开头删除 #，并将 10 （默认值）改为更高数字。记住保存文件并重新启动 sshd 服务。如需更多信息，请参阅 sshd_config man page。

15.5.2. virsh migrate 命令的附加选项

除了 --live 之外，virsh 迁移还接受以下选项：

- --direct - 用于直接迁移

- --p2p - 用于 peer-to-peer 迁移
- --tunneled - 用于隧道迁移
- --offline - 在目标上不启动域定义而不在源主机上停止域。离线迁移可以与不活动域一起使用，且必须与 --persistent 选项一起使用。
- --persistent - 在目标主机物理机器上保留域
- --undefinesource - 取消定义源主机物理机器上的域
- --suspend - 使域暂停在目标主机物理机器上
- --change-protection - 强制执行不兼容的配置更改，在迁移过程中不会对域进行任何更改；在虚拟机监控程序支持时，该标志会隐式启用，但如果虚拟机监控程序缺少更改保护支持，则可以明确用于拒绝迁移。
- --unsafe - 强制迁移进行，忽略所有安全程序。
- --verbose - 在发生迁移时显示迁移的进度
- --compressed - 激活在实时迁移过程中必须重复传输的内存页面压缩。
- --abort-on-error - 如果在迁移过程中发生软错误（如 I/O 错误）取消迁移。
- --domain [name] - 设置域名、id 或 uuid。
- --desturi [URI] - 从客户端（常规迁移）或源（p2p 迁移）中看到的目标主机的连接 URI。

- --migrateuri [URI] - 迁移 URI，通常可以省略。

- --graphicsuri [URI] - 用于无缝图形迁移的图形 URI。

- --listen-address [address] - 设置目标端管理程序应绑定到 的监听地址以进行传入迁移。

- --timeout [seconds] - 当实时迁移计数器超过 N 秒时，强制客户机虚拟机挂起。它只能用于实时迁移。启动超时后，迁移将在暂停的客户机虚拟机上继续。

- --dname [newname] - 用于在迁移期间重命名域，这通常也可省略

- --XML [filename] - 指明的文件名可用于提供目标上使用的其他 XML 文件，以便为域 XML 的任何特定于主机的部分提供较大更改，如核算源和目的地之间的命名差异。通常省略这个选项。

- --migrate-disks [disk_identifiers] - 这个选项可用于选择在迁移过程中复制哪些磁盘。这样，在复制某些磁盘时可以更有效地进行实时迁移，比如在目标上已存在或者它们不再有用时，或者它们不再有用。[disk_identifiers] 应该由逗号分隔的磁盘列表替换，如在域 XML 文件的 <target dev= /> 行中找到的参数时。

另外，以下命令可能还会帮助：

- virsh migrate-setmax downtime [domain] [downtime] - 将为要实时迁移到其他主机的域设置最多可容忍的停机时间。指定停机时间以毫秒为单位。指定的域必须与正在迁移的域相同。

- virsh migrate-compcache [domain] --size - 将设置或获得缓存的大小（以字节为单位），用于压缩实时迁移期间重复传输的内存页面。如果不使用 --size 时，命令会显示压缩缓存的当前大小。当使用 --size 时，且以字节为单位指定，管理程序会要求更改压缩以匹配指定的大小，如下为当前大小。在域被实时迁移为迁移进度时，应使用 --size 参数，并将压缩缓存数量从 domjobinfo 获取。

- virsh migrate-setspeed [domain] [bandwidth] - 设置要迁移到其他主机的指定域的迁移带宽（以 Mib/sec 为单位）。

- *virsh migrate-getspeed [domain] - 获取可用于指定域的 Mib/sec 中的最大迁移带宽。*

详情请查看 迁移限制 或者 virsh man page。

15.6. 使用 VIRT-MANAGER 迁移

这部分论述了使用 virt-manager 将 KVM 客户机虚拟机从一台主机物理机器迁移到另一台主机。

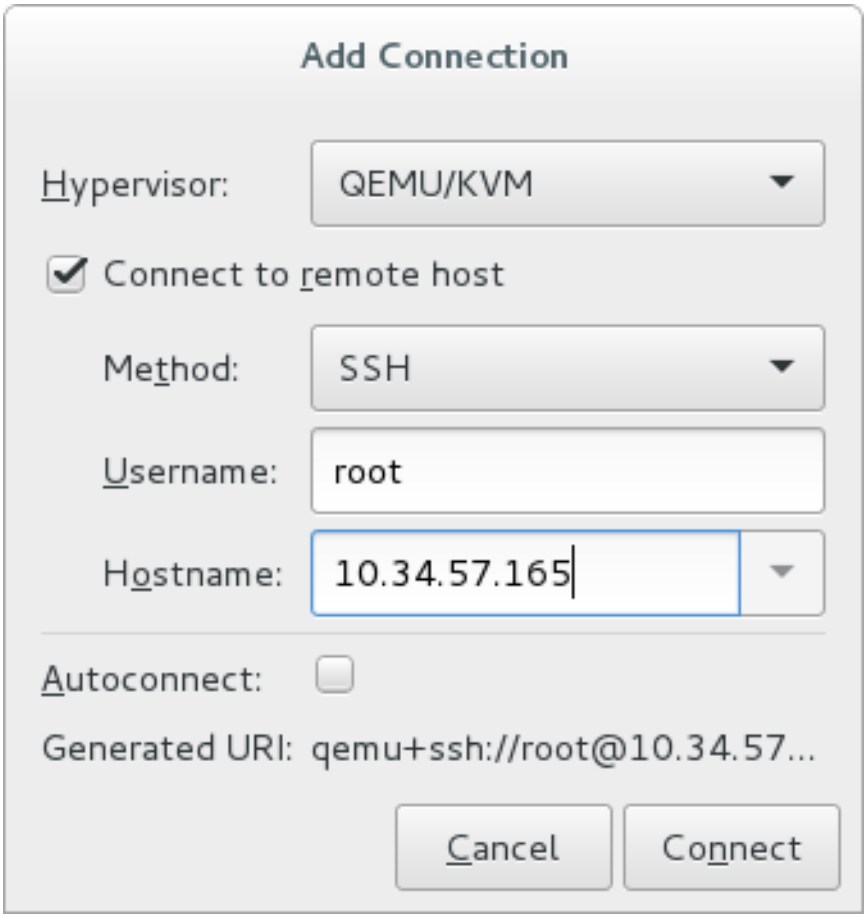
1. *连接到目标主机物理机器*

   在 virt-manager 接口 中，通过选择" 文件 "菜单连接到目标主机物理机器，然后单击" 添加连接 "。

2. *添加连接*

   此时会出现 Add Connection 窗口。

图 15.1. 向目标主机物理机器添加连接



输入以下详情：

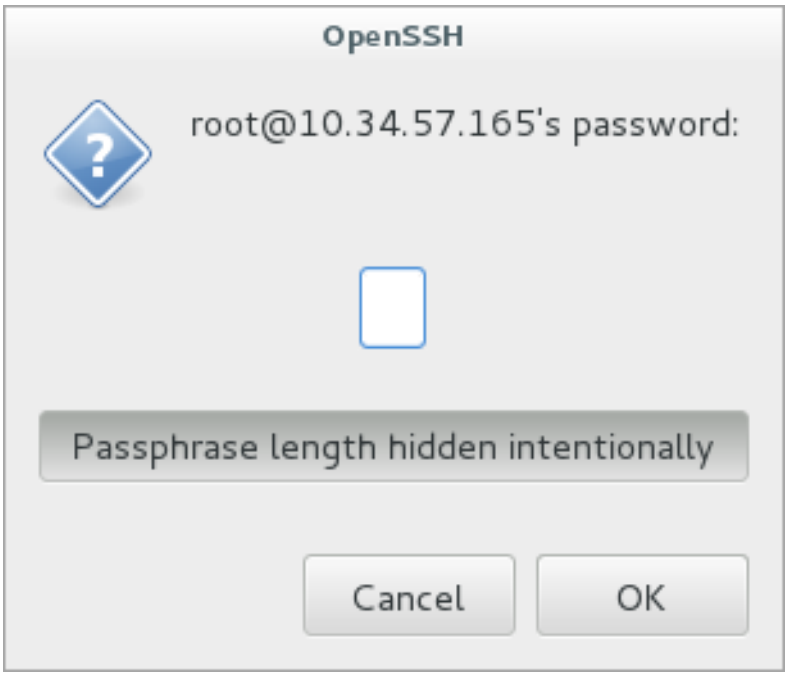
- *管理程序 ：选择 QEMU/KVM。*
- *方法 ：选择连接方法。*
- *用户名 ：输入远程主机物理机器的用户名。*
- *主机名 ：输入远程主机物理机器的主机名。*

注意

有关连接选项的详情请参考 第 19.5 节 “添加远程连接”。

点 连接。本例中使用了 SSH 连接，因此必须在下一步中输入指定的用户的密码。

图 15.2. 输入密码



3. *配置共享存储*

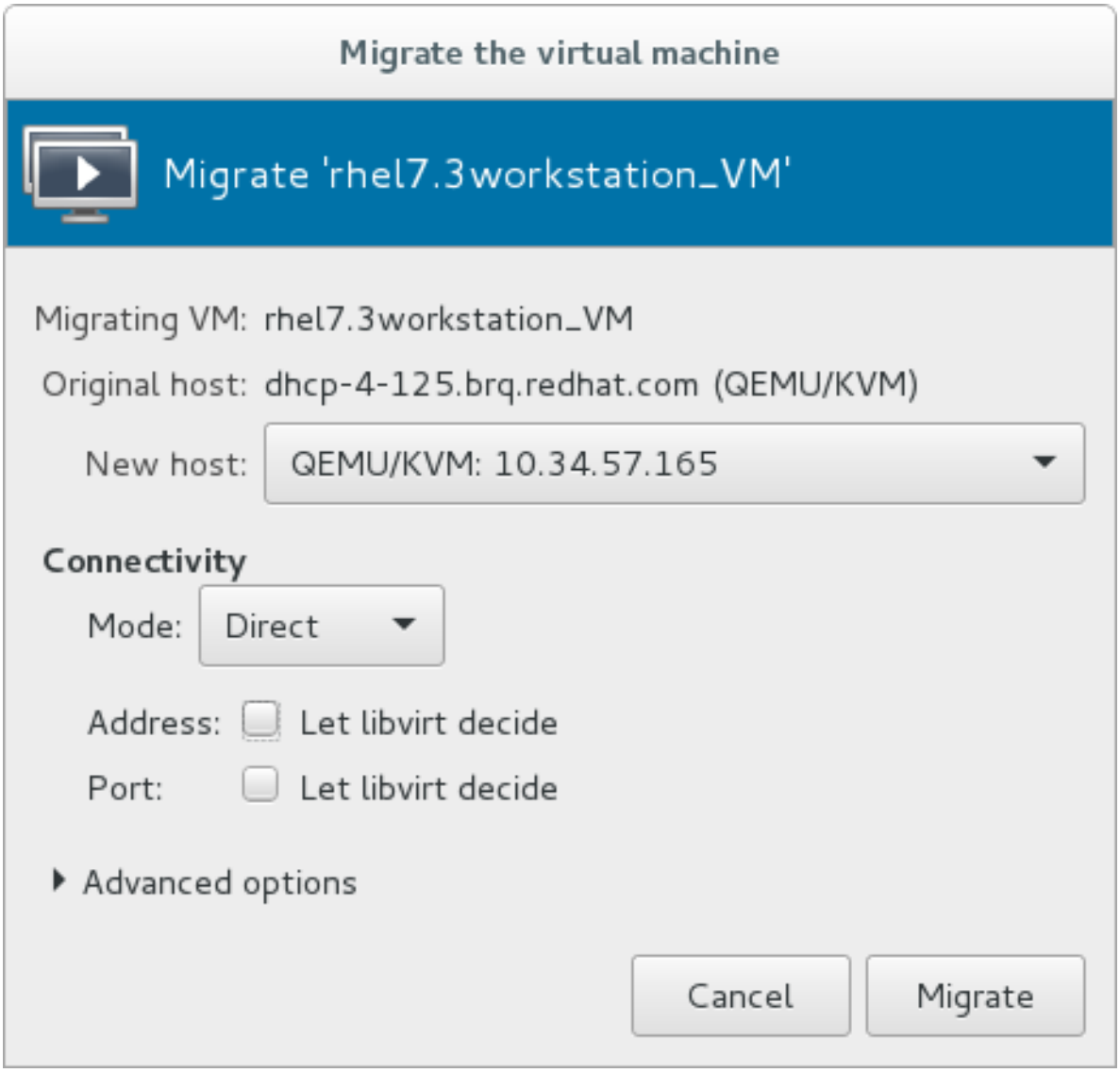
   确保源和目标主机都是共享存储，例如使用 NFS。

4. *迁移客户端虚拟机*

在要迁移的 guest 中右键单击，然后单击 迁移。

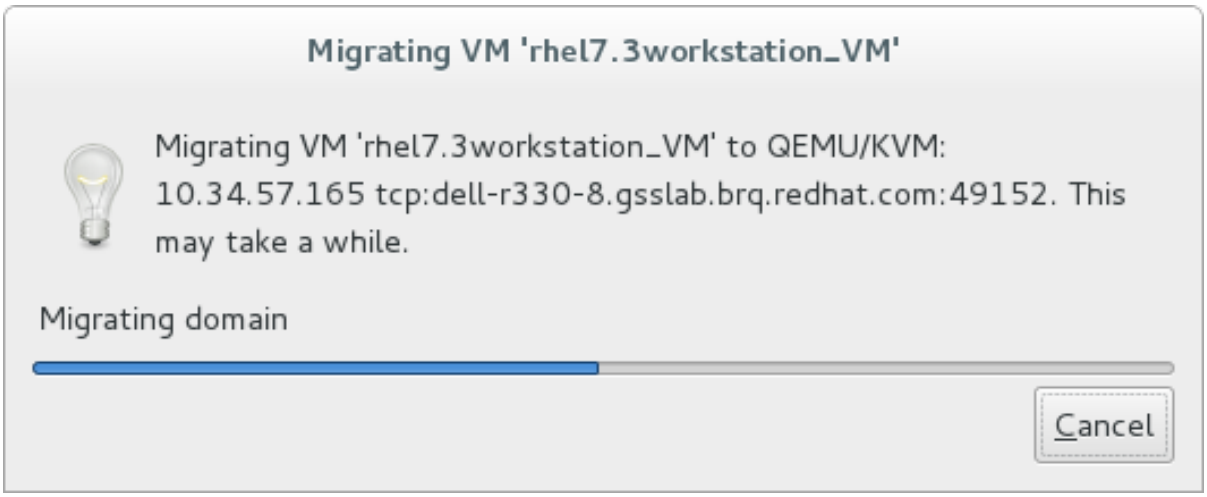
在 New Host 字段中，使用下拉列表选择要将客户机虚拟机迁移到的主机物理计算机，然后单击 Migrate。

图 15.3. 选择目的地主机物理计算机并启动迁移过程



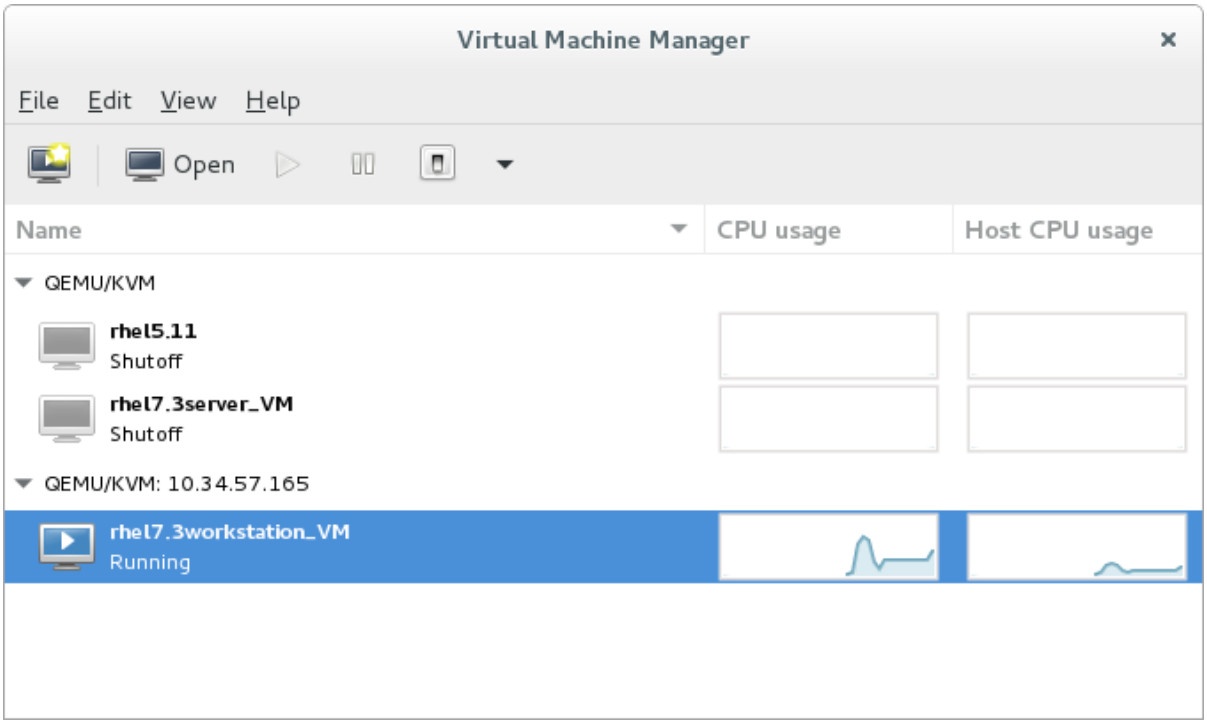
此时会出现进度窗口。

图 15.4. 进度窗口



如果迁移在没有问题的情况下完成，virt-manager 会显示在目标主机上运行的虚拟机。

图 15.5. 在目标主机物理机器中运行的迁移的客户端虚拟机



第 16 章 虚拟机设备配置

Red Hat Enterprise Linux 7 为客户机虚拟机支持三类设备：

- 模拟设备 纯是模拟实际硬件的虚拟设备，允许未经修改的客户机操作系统使用其标准 in-box 驱动程序来与其配合使用。

- VirtIO 设备 （也称为 半虚拟化）是纯粹的虚拟设备，设计为在虚拟机中最佳工作。VirtIO 设备与模拟设备类似，但默认情况下非 Linux 虚拟机不包含它们所需的驱动程序。虚拟化管理软件，如虚拟机管理器(virt-manager)和 Red Hat Virtualization Hypervisor，为支持的非 Linux 虚拟机操作系统自动安装这些驱动程序。Red Hat Enterprise Linux 7 支持 216 virtio 设备。如需更多信息，请参阅 第 5 章 KVM 半虚拟化(virtio)驱动程序。

- 分配的设备 是公开给虚拟机的物理设备。此方法也称为 passthrough。设备分配允许虚拟机独占访问 PCI 设备以实现一系列任务，并允许 PCI 设备看起来和行为，就像实际附加到客户端操作系统一样。Red Hat Enterprise Linux 7 每台虚拟机支持多达 32 个设备。

  PCIe 设备支持设备分配，包括 选择图形设备。并行 PCI 设备可能作为分配的设备支持，但由于安全性和系统配置冲突，它们存在严重的限制。

Red Hat Enterprise Linux 7 支持将 PCI 热插拔作为虚拟机单一功能插槽公开的。可以配置单一功能的主机设备以及多功能多主机设备的各个功能来启用此功能。仅建议非热插拔应用程序将设备作为多功能 PCI 插槽公开。

有关具体设备和相关限制的详情请参考 第 23.17 节 “Devices”。

注意

对于中断重新映射的平台支持，需要将客户机与主机中分配的设备进行完全隔离。如果没有这样的支持，主机可能会被受到攻击，以中断恶意客户机的注入攻击。在客户受信任的环境中，管理员可能选择使用 **allow_unsafe_interrupts** 选项为 **vfio_iommu_type1** 模块来允许 **PCI** 设备分配。这可以通过在 **/etc/modprobe.d** 中添加 **.conf** 文件（如 **local.conf**）来永久完成：

```
options vfio_iommu_type1 allow_unsafe_interrupts=1
```

或者使用 **sysfs** 条目动态执行相同的操作：

```
# echo 1 > /sys/module/vfio_iommu_type1/parameters/allow_unsafe_interrupts
```

16.1. PCI 设备

PCI 设备分配只在支持 **Intel VT-d** 或 **AMD IOMMU** 的硬件平台中使用。在主机 **BIOS** 中必须启用这些 **Intel VT-d** 或 **AMD IOMMU** 规格，才能使 **PCI** 设备分配正常工作。

过程 16.1. 为 PCI 设备分配准备 Intel 系统

1. 启用 **Intel VT-d** 规格

   Intel VT-d 规范提供对直接向虚拟机分配物理设备的硬件支持。在 **Red Hat Enterprise Linux** 中使用 **PCI** 设备分配需要这些规格。

   在 **BIOS** 中必须启用 **Intel VT-d** 规格。有些系统制造商默认禁用这些规格。查看这些规格的术语在制造商之间可能会不同；请咨询您的系统制造商文档，了解相关条款。

2. 在内核中激活 **Intel VT-d**

   在 **/etc/sysconfig/grub** 文件内添加 **intel_iommu=on** 和 **iommu=pt** 参数，在内核中激活 **Intel VT-d**。

   以下示例是启用了 **Intel VT-d** 的已修改 **GRUB** 文件。

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=vg_VolGroup00/LogVol01
vconsole.font=latarcyrheb-sun16 rd.lvm.lv=vg_VolGroup_1/root
vconsole.keymap=us $([ -x /usr/sbin/rhcrashkernel-param ] && /usr/sbin/
rhcrashkernel-param || :) rhgb quiet intel_iommu=on iommu=pt"
```

3. **重新生成配置文件**

   运行以下命令来重新生成 /etc/grub2.cfg：

```
grub2-mkconfig -o /etc/grub2.cfg
```

   请注意，如果您使用基于 UEFI 的主机，则目标文件应当是 /etc/grub2-efi.cfg。

4. **随时使用**

   重启系统以启用更改。您的系统现在可以分配 PCI 设备。

过程 16.2. 为 PCI 设备分配准备 AMD 系统

1. 启用 AMD IOMMU 规格

   在 Red Hat Enterprise Linux 中使用 PCI 设备分配需要 AMD IOMMU 规格。在 BIOS 中必须启用这些规格。有些系统制造商默认禁用这些规格。

2. 启用 IOMMU 内核支持

   在 /etc/sysconfig/grub 中的引号内，将 iommu=pt 附加到 GRUB_CMDLINX_LINUX 行的末尾，以便在启动时启用 AMD IOMMU 规格。

3. **重新生成配置文件**

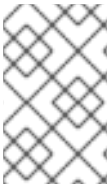
   运行以下命令来重新生成 /etc/grub2.cfg：

```
grub2-mkconfig -o /etc/grub2.cfg
```

   请注意，如果您使用基于 UEFI 的主机，则目标文件应当是 /etc/grub2-efi.cfg。

4. **随时使用**

   重启系统以启用更改。您的系统现在可以分配 PCI 设备。



注意

有关 IOMMU 的详情，请参考 附录 E, 使用 IOMMU 组。

16.1.1. 使用 virsh 分配 PCI 设备

这些步骤涵盖了将 PCI 设备分配给 KVM 管理程序上的虚拟机。

本例使用 PCIe 网络控制器和 PCI 标识符代码 pci_0000_01_00_0，以及一个名为 guest1-rhel7-64 的完全虚拟化客户端机器。

过程 16.3. 使用 virsh 为客户机虚拟机分配 PCI 设备

1. 确定设备

首先，识别为分配给虚拟机的设备分配的 PCI 设备。使用 lspci 命令列出可用的 PCI 设备。您可以使用 grep 重新定义 lspci 的输出。

本例使用在以下输出中突出显示的以太网控制器：

```
# lspci | grep Ethernet
00:19.0 Ethernet controller: Intel Corporation 82567LM-2 Gigabit Network Connection
01:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
01:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
```

此时会显示这个以太网控制器，其简短标识符 00:19.0。我们需要找到 virsh 使用的完整标识符，以将此 PCI 设备分配给虚拟机。

为此，请使用 virsh nodedev-list 命令列出附加到主机机器的特定类型(pci)的所有设备。然后查看映射到您要使用的设备的简短标识符的字符串的输出。

本例显示了使用简短标识符 00:19.0 映射到以太网控制器的字符串。请注意，：和 . 字符在完整的标识符中被替换为下划线。

```
# virsh nodedev-list --cap pci
pci_0000_00_00_0
pci_0000_00_01_0
pci_0000_00_03_0
pci_0000_00_07_0
pci_0000_00_10_0
pci_0000_00_10_1
pci_0000_00_14_0
pci_0000_00_14_1
pci_0000_00_14_2
pci_0000_00_14_3
pci_0000_00_19_0
pci_0000_00_1a_0
pci_0000_00_1a_1
```

```
pci_0000_00_1a_2
pci_0000_00_1a_7
pci_0000_00_1b_0
pci_0000_00_1c_0
pci_0000_00_1c_1
pci_0000_00_1c_4
pci_0000_00_1d_0
pci_0000_00_1d_1
pci_0000_00_1d_2
pci_0000_00_1d_7
pci_0000_00_1e_0
pci_0000_00_1f_0
pci_0000_00_1f_2
pci_0000_00_1f_3
pci_0000_01_00_0
pci_0000_01_00_1
pci_0000_02_00_0
pci_0000_02_00_1
pci_0000_06_00_0
pci_0000_07_02_0
pci_0000_07_03_0
```

记录映射到您要使用的设备的 PCI 设备号；其他步骤中这是必需的。

2. 查看设备信息

有关域、总线和功能的信息可从 virsh nodedev-dumpxml 命令的输出中获得：

图 16.1. 转储内容

```
# virsh nodedev-dumpxml pci_0000_00_19_0
<device>
  <name>pci_0000_00_19_0</name>
  <parent>computer</parent>
  <driver>
    <name>e1000e</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>0</bus>
    <slot>25</slot>
    <function>0</function>
    <product id='0x1502'>82579LM Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <iommuGroup number='7'>
      <address domain='0x0000' bus='0x00' slot='0x19' function='0x0'/>
    </iommuGroup>
  </capability>
</device>
```

注意

IOMMU 组根据 IOMMU 的视角的可见性和隔离来确定。每个 IOMMU 组可以包含一个或多个设备。当存在多个设备时，必须声明 IOMMU 组内的所有端点，以便将该组中的所有设备分配给客户机。这也可以通过为客户机分配额外的端点，也可以使用 virsh nodedev-detach 从主机驱动程序中分离它们。单个组中包含的设备可能不在多个客户机间分割，或者在主机和客户机间分割。PCIe 根端口、交换机端口和网桥等非端点设备不应与主机驱动程序分离，而不会干扰分配端点。

IOMMU 组中的设备可以使用 virsh nodedev-dumpxml 输出的 iommuGroup 部分确定。组的每个成员都在单独的 "address" 字段中提供。这些信息也可以在 sysfs 中使用以下内容：

```
$ ls /sys/bus/pci/devices/0000:01:00.0/iommu_group/devices/
```

输出示例如下：

```
0000:01:00.0  0000:01:00.1
```

要只为客户机分配 0000.01.00.0，在启动客户机前应该从主机分离未使用的端点：

```
$ virsh nodedev-detach pci_0000_01_00_1
```

3. **确定所需的配置详情**

如需配置文件所需的值，请参见 **virsh nodedev-dumpxml pci_0000_00_19_0** 命令的输出。

示例设备具有以下值： **bus = 0, 插槽 = 25, function = 0**。十进制配置使用这三个值：

```
bus='0'
slot='25'
function='0'
```

4. **添加配置详情**

运行 **virsh edit**，指定虚拟机名称，并在 **<devices>** 部分添加一个设备条目，将 PCI 设备分配给客户机虚拟机。例如：

```
# virsh edit guest1-rhel7-64
```

图 16.2. 添加 PCI 设备

```
<devices>
[...]
 <hostdev mode='subsystem' type='pci' managed='yes'>
   <source>
     <address domain='0' bus='0' slot='25' function='0'/>
   </source>
 </hostdev>
[...]
</devices>
```

或者，运行 virsh attach-device，指定虚拟机名称和客户机 XML 文件：

```
virsh attach-device guest1-rhel7-64 file.xml
```

注意

PCI 设备可以包括 可选的只读内存(ROM)模块 （也称为 ROM 或 扩展 ROM ）， 用于提供设备固件或预启动驱动程序（如 PXE）。通常，在使用 PCI 设备分配时，这些选项 ROM 也可在虚拟环境中工作，将物理 PCI 设备附加到虚拟机。

但是，在某些情况下，选项 ROM 可能是不必要的。这可能会导致虚拟机引导速度变慢，或者该设备提供的预引导驱动程序与虚拟化不兼容，这可能会导致客户端操作系统引导失败。在这种情况下，红帽建议从虚拟机屏蔽选项 ROM。要做到这一点：

1. 在主机上，验证要分配的设备是否具有扩展 ROM 基本地址寄存器 (BAR)。要做到这一点，将 lspci -v 命令用于该设备，并检查包含以下内容的行：

   ```
   Expansion ROM at
   ```

2. 将 <rom bar='off'/> 元素作为客户机 XML 配置中的 <hostdev> 元素的子项添加：

   ```
   <hostdev mode='subsystem' type='pci' managed='yes'>
     <source>
       <address domain='0' bus='0' slot='25' function='0'/>
     </source>
     <rom bar='off'/>
   </hostdev>
   ```

5. 启动虚拟机

```
# virsh start guest1-rhel7-64
```

PCI 设备现在应该能够成功分配给虚拟机，并可供客户端操作系统访问。

16.1.2. 使用 virt-manager 分配 PCI 设备

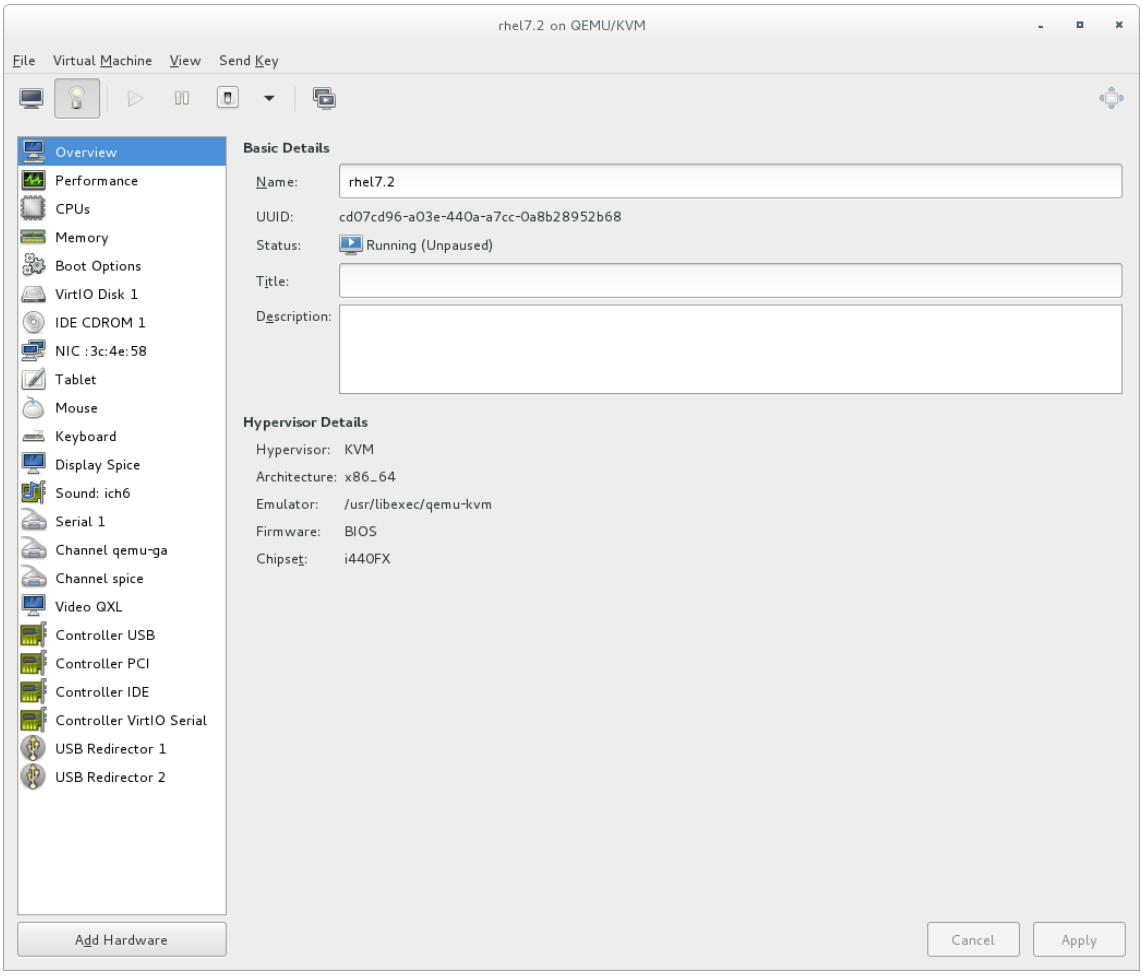
PCI 设备可以使用图形 virt-manager 工具添加到客户机虚拟机中。以下流程将千兆位以太网控制器添加到客户机虚拟机中。

过程 16.4. 使用 virt-manager 为客户机虚拟机分配 PCI 设备

1. 打开硬件设置

   打开 guest 虚拟机，然后单击添加硬件 按钮向虚拟机添加新设备。

图 16.3. 虚拟机硬件信息窗口

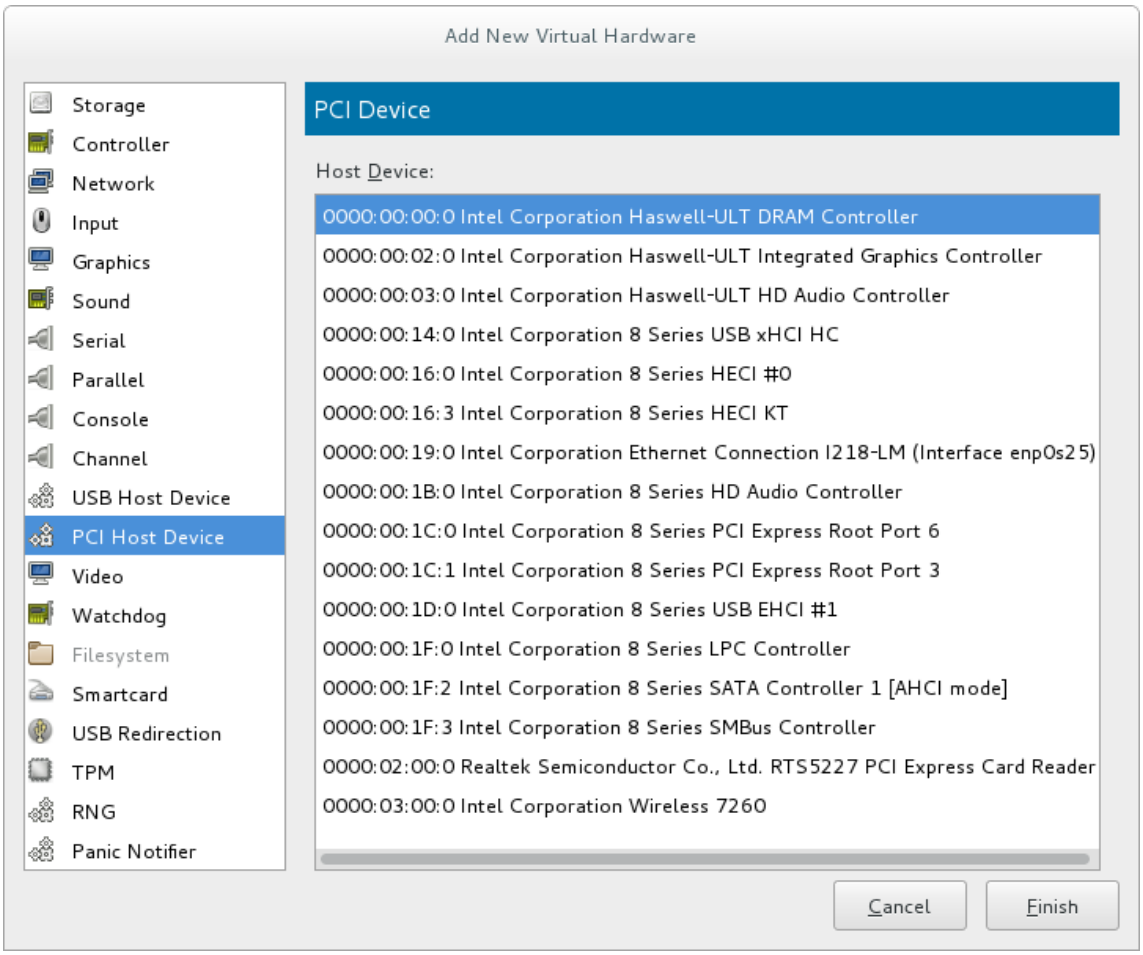


2. 选择 PCI 设备

从左侧 的硬件 列表中选择 PCI 主机设备。

选择未使用的 PCI 设备。请注意，选择存在于其他客户机中的 PCI 设备 会导致错误。本例中使用了备用的音频控制器。单击 Finish 以完成设置。

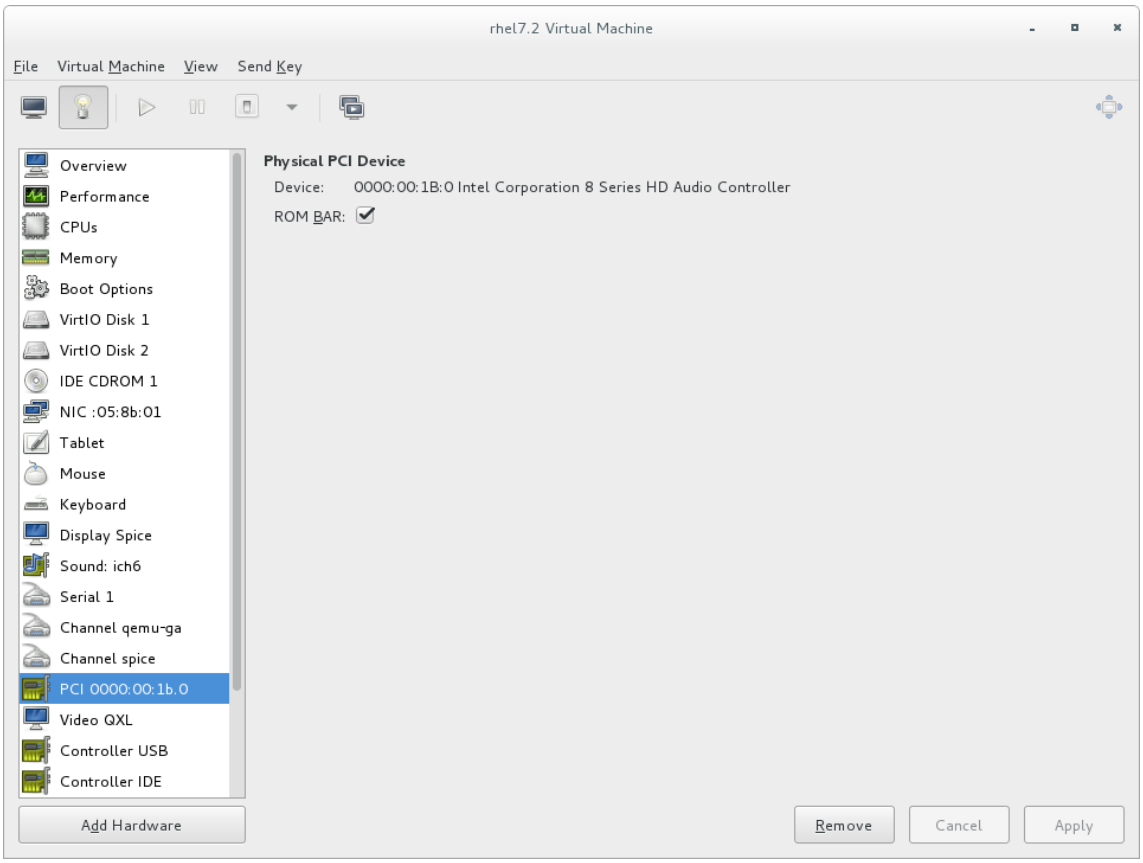
图 16.4. Add new virtual hardware 向导

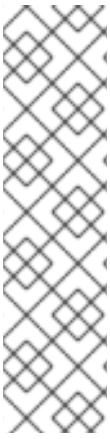


3. *添加新设备*

设置已完成，guest 虚拟机现在可以直接访问 PCI 设备。

图 16.5. 虚拟机硬件信息窗口





注意

如果设备分配失败，则同一 IOMMU 组中可能存在其他端点仍然附加到主机。无法使用 virt-manager 检索组信息，但 virsh 命令可用于分析 IOMMU 组的绑定，以及所需的 sequester 设备。

有关 IOMMU 组以及如何分离端点设备的更多信息，请参阅 注意 中的 第 16.1.1 节 “使用 virsh 分配 PCI 设备”。

16.1.3. 使用 virt-install 的 PCI 设备分配

在使用 virt-install 命令安装客户机时，可以分配 PCI 设备。为此，请使用 --host-device 参数。

过程 16.5. 使用 virt-install 为虚拟机分配 PCI 设备

1. *确定设备*

   识别为分配给客户机虚拟机的设备分配的 PCI 设备。

```
# lspci | grep Ethernet
00:19.0 Ethernet controller: Intel Corporation 82567LM-2 Gigabit Network Connection
```

```
01:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
01:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
```

virsh nodedev-list 命令列出与系统连接的所有设备，并使用字符串来识别每个 PCI 设备。要只把输出限制为 PCI 设备，请输入以下命令：

```
# virsh nodedev-list --cap pci
pci_0000_00_00_0
pci_0000_00_01_0
pci_0000_00_03_0
pci_0000_00_07_0
pci_0000_00_10_0
pci_0000_00_10_1
pci_0000_00_14_0
pci_0000_00_14_1
pci_0000_00_14_2
pci_0000_00_14_3
pci_0000_00_19_0
pci_0000_00_1a_0
pci_0000_00_1a_1
pci_0000_00_1a_2
pci_0000_00_1a_7
pci_0000_00_1b_0
pci_0000_00_1c_0
pci_0000_00_1c_1
pci_0000_00_1c_4
pci_0000_00_1d_0
pci_0000_00_1d_1
pci_0000_00_1d_2
pci_0000_00_1d_7
pci_0000_00_1e_0
pci_0000_00_1f_0
pci_0000_00_1f_2
pci_0000_00_1f_3
pci_0000_01_00_0
pci_0000_01_00_1
pci_0000_02_00_0
pci_0000_02_00_1
pci_0000_06_00_0
pci_0000_07_02_0
pci_0000_07_03_0
```

记录 **PCI** 设备编号；其他步骤中需要数字。

有关域、总线和功能的信息可从 virsh nodedev-dumpxml 命令的输出中获得：

```
# virsh nodedev-dumpxml pci_0000_01_00_0
```

图 16.6. PCI 设备文件内容

```
<device>
  <name>pci_0000_01_00_0</name>
  <parent>pci_0000_00_01_0</parent>
  <driver>
    <name>igb</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>1</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x10c9'>82576 Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <iommuGroup number='7'>
      <address domain='0x0000' bus='0x00' slot='0x19' function='0x0'/>
    </iommuGroup>
  </capability>
</device>
```

注意

如果在 IOMMU 组中有多个端点，且并非所有端点都分配给客户端，则需要在启动客户端前运行以下命令从主机中手动分离其他端点：

```
$ virsh nodedev-detach pci_0000_00_19_1
```

有关 IOMMU 组的更多信息，请参阅 注意 中的 第 16.1.1 节 “使用 virsh 分配 PCI 设备”。

2. 添加设备

使用 virsh nodedev 命令的 PCI 标识符输出作为 --host-device 参数的值。

```
virt-install \
--name=guest1-rhel7-64 \
--disk path=/var/lib/libvirt/images/guest1-rhel7-64.img,size=8 \
--vcpus=2 --ram=2048 \
--location=http://example1.com/installation_tree/RHEL7.0-Server-x86_64/os \
--nonetworks \
--os-type=linux \
--os-variant=rhel7
--host-device=pci_0000_01_00_0
```

3. 完成安装

   完成客户机安装。PCI 设备应连接到客户机。

16.1.4. 分离分配的 PCI 设备

当主机 PCI 设备分配给客户机机器时，主机将不再使用该设备。如果 PCI 设备处于 受管 模式（使用 域 XML 文件中的 managed='yes 参数进行配置），它会连接到客户端机器并从客户端机器分离，并根据需要重新连接到主机机器。如果 PCI 设备没有处于 受管 模式，您可以将 PCI 设备从虚拟客户机中分离，并使用 virsh 或 virt-manager 重新连接。

过程 16.6. 使用 virsh 从客户机中分离 PCI 设备

1. 分离设备

   使用以下命令，在客户机的 XML 文件中删除 PCI 设备来从客户机中分离 PCI 设备：

```
# virsh detach-device name_of_guest file.xml
```

2. 重新将设备附加到主机（可选）

   如果设备处于 受管 模式，则跳过此步骤。该设备将自动返回到主机。

   如果设备没有使用 受管 模式，使用以下命令将 PCI 设备重新关联到主机机器：

```
# virsh nodedev-reattach device
```

   例如，要将 pci_0000_01_00_0 设备重新关联到主机：

```
# virsh nodedev-reattach pci_0000_01_00_0
```

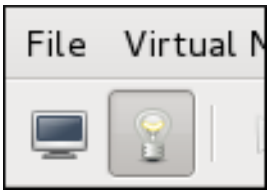
   该设备现在可供主机使用。

过程 16.7. 使用 virt-manager 从客户机中分离 PCI 设备

1. 打开虚拟硬件详情屏幕

   在 virt-manager 中，双击包含该设备的虚拟机。选择 Show virtual hardware details 按钮，以显示虚拟硬件列表。

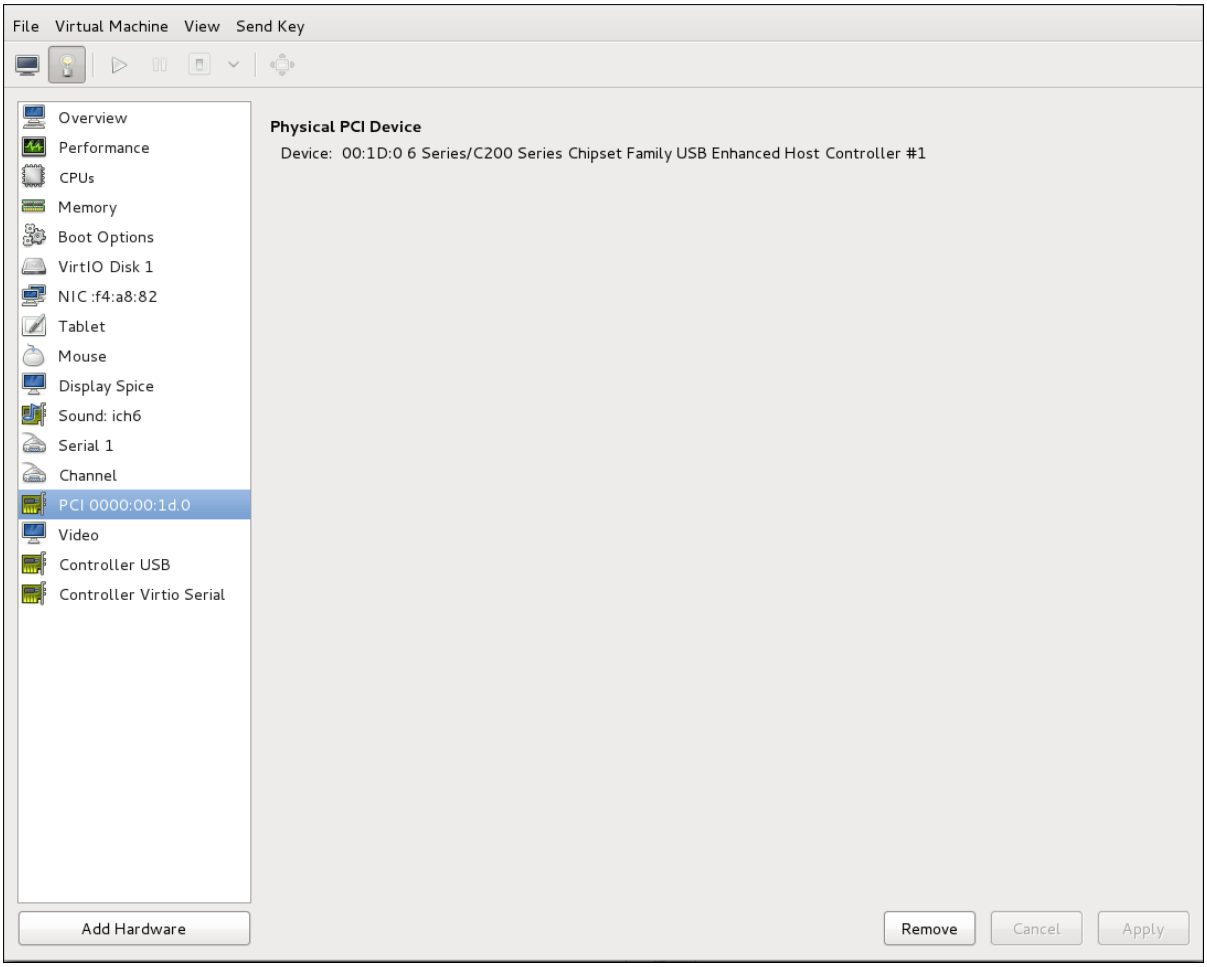
图 16.7. 虚拟硬件详情按钮



2. 选择并删除该设备

从左侧面板中的虚拟设备列表选择要分离的 PCI 设备。

图 16.8. 选择要分离的 PCI 设备



单击删除按钮确认。该设备现在可供主机使用。

16.1.5. PCI Bridges

外围设备组件互联(PCI)网桥用于附加到网卡、模式和声卡等设备。就像其物理对应设备一样，虚拟设备也可以附加到 PCI 网桥。过去，任何客户机虚拟机只能添加 31 个 PCI 设备。现在，当添加 31st PCI 设备时，PCI 网桥会自动放置在 31st 插槽中，将额外 PCI 设备移到 PCI 网桥。每个 PCI 网桥都有 31 个插槽用于 31 个设备，它们都可以是网桥的。这样，客户机虚拟机可以使用超过 900 个设备。

有关 PCI 网桥的 XML 配置示例，请参阅 PCI Bridge 的 Domain XML 示例。请注意，此配置是自动设置的，我们不推荐手动调整。

16.1.6. PCI 设备分配限制

PCI 设备分配（将 PCI 设备附加到虚拟机）需要主机系统具有 AMD IOMMU 或 Intel VT-d 支持，以启用 PCIe 设备的设备分配。

Red Hat Enterprise Linux 7 通过客户机设备驱动程序访问有限的 PCI 配置空间。这个限制可能会导致扩展 PCI 配置空间中取决于设备功能或功能的驱动程序失败。

每个 Red Hat Enterprise Linux 7 虚拟机有总计 32 个设备。这转换为 32 个 PCI 功能，无论虚拟机中存在 PCI 网桥的数量，或者如何合并这些功能来创建多功能插槽。

对于中断重新映射的平台支持，需要将客户机与主机中分配的设备进行完全隔离。如果没有这样的支持，主机可能会被受到攻击，以中断恶意客户机的注入攻击。在客户机受信任的环境中，管理员可能选择是否允许 PCI 设备分配使用 vfio_iommu_type1 模块的 allow_unsafe_interrupts 选项。这可以通过在 /etc/modprobe.d 中添加 .conf 文件（如 local.conf）来永久完成：

```
options vfio_iommu_type1 allow_unsafe_interrupts=1
```

或者使用 sysfs 条目动态执行相同的操作：

```
# echo 1 > /sys/module/vfio_iommu_type1/parameters/allow_unsafe_interrupts
```

16.2. 使用 SR-IOV 设备的 PCI 设备分配

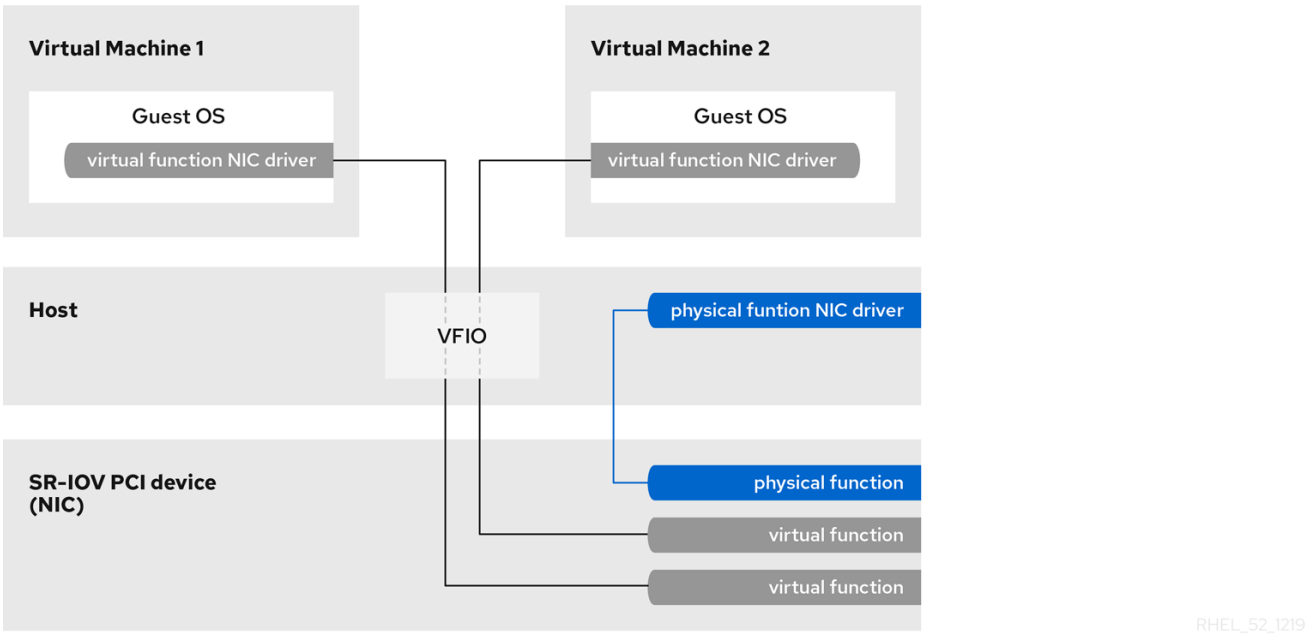
PCI 网络设备（在域 XML 中由 <源> 元素指定）可以使用直接设备分配（有时称为 passthrough）直接连接到客户机。 由于标准单一以太网卡驱动程序设计的限制，只能为客户机分配 单根 I/O 虚拟化 (SR-IOV) 虚拟功能(VF)设备；若要为客户机分配标准单端口 PCI 或 PCIe 以太网卡，请使用传统的 <hostdev> 设备定义。

图 16.9. PCI 设备分配的 XML 示例

```
<devices>
<interface type='hostdev'>
 <driver name='vfio'/>
 <source>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
 </source>
 <mac address='52:54:00:6d:90:02'>
 <virtualport type='802.1Qbh'>
  <parameters profileid='finance'/>
 </virtualport>
</interface>
</devices>
```

由 PCI-SIG(PCI Special Interest Group)开发，单一根 I/O 虚拟化(SR-IOV)规格是一种标准的 PCI 设备分配，可以将单个设备共享给多个虚拟机。SR-IOV 提高虚拟机的设备性能。

图 16.10. SR-IOV 的工作原理



SR-IOV 支持单根功能（例如，单个以太网端口）来显示为多个、独立的物理设备。可以将具有 SR-IOV 功能的物理设备配置为作为多种功能出现在 PCI 配置空间中。每个设备都有其自身的配置空间完成 Base Address Registers(BAR)。

SR-IOV 使用两个 PCI 功能：

- *物理功能(PF)是包含 SR-IOV 功能的完整 PCIe 设备。发现、托管和配置为普通 PCI 设备的物理功能。物理功能通过分配虚拟功能配置和管理 SR-IOV 功能。*

- 虚拟功能(VF)是简单的 PCIe 功能，仅处理 I/O。每个虚拟功能都源自物理功能。设备可能拥有的虚拟功能数量受设备硬件的限制。单个以太网设备（物理设备）可映射到可共享虚拟机的许多虚拟功能。

管理程序可向虚拟机分配一个或多个虚拟功能。虚拟功能的配置空间将分配给提供给 guest 的配置空间。

每个虚拟功能每次只能分配给一个客户机，因为虚拟功能需要实际硬件资源。虚拟机可以有多个虚拟功能。虚拟功能显示为网卡的方式，其方式与常规网卡出现在操作系统中一样。

SR-IOV 驱动程序在内核中实现。核心实施包含在 PCI 子系统中，但还必须对物理功能(PF)和虚拟功能(VF)设备提供驱动程序支持。具有 SR-IOV 功能的设备可以从 PF 分配 VF。VF 显示为 PCI 设备，它们根据队列和注册集等资源在物理 PCI 设备上支持。

16.2.1. SR-IOV 的优点

SR-IOV 设备可以与多个虚拟机共享单个物理端口。

当为虚拟机分配了 SR-IOV VF 时，可以将其配置为（与虚拟机完全转换为虚拟机）会将所有网络流量都离开于特定的 VLAN。虚拟机无法检测到其流量是否已标记为 VLAN，且无法更改或消除此标记。

虚拟功能具有接近原生的性能，比半虚拟化驱动程序和仿真访问提供更好的性能。虚拟功能提供与数据在同一物理服务器上虚拟机之间的数据保护，由硬件控制。

这些功能允许在数据中心中的主机上增加虚拟机密度。

SR-IOV 更好地利用多个客户机的设备带宽。

16.2.2. 使用 SR-IOV

本节介绍了使用 PCI 透传将支持 SR-IOV 的虚拟功能（multiport 网卡）为虚拟机为虚拟机分配为网络设备。

通过添加带有 virsh edit 或 virsh attach-device 命令的 <hostdev> 中的设备条目，可将 SR-IOV 虚拟功能(VF)分配给虚拟机。但是，这个问题可能会有问题，因为与常规网络设备不同，SR-IOV VF 网络设

备没有永久唯一的 MAC 地址，每次主机重启时都会分配一个新的 MAC 地址。因此，即使客户端重启后分配了相同的 VF，当主机重启时，客户机确定其网络适配器具有新的 MAC 地址。因此，客户机在每次都连接了新硬件，通常需要对客户机的网络设置进行重新配置。

libvirt 包含 <interface type='hostdev'> 接口设备。通过使用此接口设备，libvirt 将首先执行指示的任何网络特定硬件/交换机初始化（如设置 MAC 地址、VLAN 标签或 802.1Qbh 虚拟端口参数），然后对客户机执行 PCI 设备分配。

使用 <interface type='hostdev'> 接口设备需要：

- 支持 SR-IOV 的网卡，
- 支持 Intel VT-d 或 AMD IOMMU 扩展的主机硬件
- 要分配的 VF 的 PCI 地址。

重要

将 SR-IOV 设备分配给虚拟机需要主机硬件支持 Intel VT-d 或 AMD IOMMU 规格。

要在 Intel 或 AMD 系统中附加 SR-IOV 网络设备，请按照以下步骤执行：

过程 16.8. 在 Intel 或 AMD 系统中附加 SR-IOV 网络设备

1. 在 BIOS 和内核中启用 Intel VT-d 或者 AMD IOMMU 规格

   在 Intel 系统中，在 BIOS 中启用 Intel VT-d（如果尚未启用）。请参阅 过程 16.1, “为 PCI 设备分配准备 Intel 系统” 在 BIOS 和内核中启用 Intel VT-d 中的程序帮助。

   如果 Intel VT-d 已经启用并正常工作，请跳过这一步。

   在 AMD 系统中，在 BIOS 中启用 AMD IOMMU 规格（如果尚未启用）。请参阅 过程 16.2, “为 PCI 设备分配准备 AMD 系统” 有关在 BIOS 中启用 IOMMU 的过程帮助。

2. *验证支持*

验证是否检测到具有 SR-IOV 功能的 PCI 设备。这个示例列出了支持 SR-IOV 的 Intel 82576 网络接口卡。使用 lspci 命令验证该设备是否已检测到。

```
# lspci
03:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
03:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
```

请注意，输出已被修改来删除所有其它设备。

3. *激活虚拟功能*

运行以下命令：

```
# echo ${num_vfs} > /sys/class/net/enp14s0f0/device/sriov_numvfs
```

4. *使虚拟功能持久*

要使虚拟功能在重新引导时持久保留，请使用您选择的编辑器创建类似如下的 udev 规则，其中指定 VF 的预期数量（本例中为 2），最多使用网络接口卡支持的限制。在以下示例中，将 enp14s0f0 替换为 PF 网络设备名称，并调整 ENV{ID_NET_DRIVER} 值以匹配正在使用的驱动程序：

```
# vim /etc/udev/rules.d/enp14s0f0.rules
```

```
ACTION=="add", SUBSYSTEM=="net", ENV{ID_NET_DRIVER}=="ixgbe",
ATTR{device/sriov_numvfs}="2"
```

这将确保在引导时启用该功能。

5. *检查新的虚拟功能*

使用 lspci 命令，列出附加到 Intel 82576 网络设备的新增虚拟功能。（此外，使用 grep 搜索 虚拟功能，以搜索支持虚拟功能的设备。）

```
# lspci | grep 82576
0b:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
0b:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network Connection (rev 01)
0b:10.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:10.1 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:10.2 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:10.3 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:10.4 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
```

```
0b:10.5 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:10.6 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:10.7 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:11.0 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:11.1 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:11.2 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:11.3 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:11.4 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
0b:11.5 Ethernet controller: Intel Corporation 82576 Virtual Function (rev 01)
```

PCI 设备标识符通过 lspci 命令的 -n 参数找到。物理功能与 0b:00.0 和 0b:00.1 对应。所有虚拟功能在描述中包含 虚拟功能。

6. 使用 virsh 验证设备是否存在

libvirt 服务必须在向虚拟机中添加设备前识别该设备。libvirt 使用与 lspci 输出类似的标记。lspci 输出的所有标点字符 : 和 . 均更改为下划线(_)。

使用 virsh nodedev-list 命令和 grep 命令从可用主机设备列表中过滤 Intel 82576 网络设备。0b 是本例中 Intel 82576 网络设备的过滤器。这可能会因您的系统而异，并可能导致额外的设备。

```
# virsh nodedev-list | grep 0b
pci_0000_0b_00_0
pci_0000_0b_00_1
pci_0000_0b_10_0
pci_0000_0b_10_1
pci_0000_0b_10_2
pci_0000_0b_10_3
pci_0000_0b_10_4
pci_0000_0b_10_5
pci_0000_0b_10_6
pci_0000_0b_11_7
pci_0000_0b_11_1
pci_0000_0b_11_2
pci_0000_0b_11_3
pci_0000_0b_11_4
pci_0000_0b_11_5
```

虚拟功能和物理功能的 PCI 地址应该位于列表中。

7. 使用 virsh 获取设备详情

pci_0000_0b_00_0 是物理功能之一，pci_0000_0b_10_0 是物理功能第一个对应的虚拟功能。使用 virsh nodedev-dumpxml 命令获取这两个设备的详情。

```
# virsh nodedev-dumpxml pci_0000_03_00_0
```

```
<device>
  <name>pci_0000_03_00_0</name>
  <path>/sys/devices/pci0000:00/0000:00:01.0/0000:03:00.0</path>
  <parent>pci_0000_00_01_0</parent>
  <driver>
    <name>igb</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>3</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x10c9'>82576 Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <capability type='virt_functions'>
      <address domain='0x0000' bus='0x03' slot='0x10' function='0x0'/>
      <address domain='0x0000' bus='0x03' slot='0x10' function='0x2'/>
      <address domain='0x0000' bus='0x03' slot='0x10' function='0x4'/>
      <address domain='0x0000' bus='0x03' slot='0x10' function='0x6'/>
      <address domain='0x0000' bus='0x03' slot='0x11' function='0x0'/>
      <address domain='0x0000' bus='0x03' slot='0x11' function='0x2'/>
      <address domain='0x0000' bus='0x03' slot='0x11' function='0x4'/>
    </capability>
    <iommuGroup number='14'>
      <address domain='0x0000' bus='0x03' slot='0x00' function='0x0'/>
      <address domain='0x0000' bus='0x03' slot='0x00' function='0x1'/>
    </iommuGroup>
  </capability>
</device>
```

```
# virsh nodedev-dumpxml pci_0000_03_11_5
<device>
  <name>pci_0000_03_11_5</name>
  <path>/sys/devices/pci0000:00/0000:00:01.0/0000:03:11.5</path>
  <parent>pci_0000_00_01_0</parent>
  <driver>
    <name>igbvf</name>
  </driver>
  <capability type='pci'>
    <domain>0</domain>
    <bus>3</bus>
    <slot>17</slot>
    <function>5</function>
    <product id='0x10ca'>82576 Virtual Function</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <capability type='phys_function'>
      <address domain='0x0000' bus='0x03' slot='0x00' function='0x1'/>
    </capability>
    <iommuGroup number='35'>
      <address domain='0x0000' bus='0x03' slot='0x11' function='0x5'/>
    </iommuGroup>
  </capability>
</device>
```

本例将虚拟功能 pci_0000_03_10_2 添加到第 8 步的虚拟机。请注意虚拟功能的 bus、插槽

和功能参数：这些是添加该设备所必需的。

将这些参数复制到临时 XML 文件中，例如 /tmp/new-interface.xml。

```
<interface type='hostdev' managed='yes'>
  <source>
    <address type='pci' domain='0x0000' bus='0x03' slot='0x10' function='0x2'/>
  </source>
</interface>
```

注意

当虚拟机启动时，它应该会看到物理适配器提供的类型的网络设备，并配置了 MAC 地址。此 MAC 地址将在主机间保持不变，并且 guest 重新启动。

以下 <interface> 示例显示可选 <mac address>、<virtualport> 和 <vlan> 元素的语法。在实践中，使用 <vlan> 或 <virtualport> 元素，不能同时使用这两个元素，如示例所示：

```
...
 <devices>
   ...
   <interface type='hostdev' managed='yes'>
     <source>
       <address type='pci' domain='0' bus='11' slot='16' function='0'/>
     </source>
     <mac address='52:54:00:6d:90:02'>
     <vlan>
        <tag id='42'/>
     </vlan>
     <virtualport type='802.1Qbh'>
       <parameters profileid='finance'/>
     </virtualport>
   </interface>
   ...
 </devices>
```

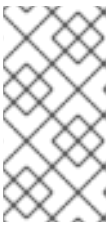
如果您没有指定 MAC 地址，则会自动生成一个地址。<virtualport> 元素仅在连接到 802.11Qbh 硬件交换机时使用。<vlan> 元素将透明地将客户机的设备放在标记 42 的 VLAN 上。

8. 将虚拟功能添加到虚拟机

通过以下命令，将虚拟功能添加到虚拟机，以及上一步中创建的临时文件。这会立即附加新设备，并保存它以便在以后的客户机重启。

```
virsh attach-device MyGuest /tmp/new-interface.xml --live --config
```

使用 virsh attach-device 指定 --live 选项，将新设备附加到运行的客户机中。使用 --config 选项可确保新设备在以后的客户机重启后可用。



注意

只有在客户机运行时接受 --live 选项。如果在非运行的客户机中使用 --live 选项，则 virsh 将返回错误。

虚拟机检测到一个新的网卡。这个新卡是 SR-IOV 设备的虚拟功能。

16.2.3. 使用 SR-IOV 设备配置 PCI 分配

SR-IOV 网卡提供多个 VF，各自可使用 PCI 设备分配的客户机虚拟机单独分配给客户机虚拟机。分配后，每个操作都作为完整的物理网络设备运行。这允许很多客户机虚拟机获得直接 PCI 设备分配的性能优势，而仅在主机物理计算机上使用一个插槽。

这些 VF 可使用 <hostdev> 元素以传统方式分配给客户机虚拟机。但是，SR-IOV VF 网络设备没有永久唯一的 MAC 地址，这会导致在每次主机物理机器重启时客户机虚拟机的网络设置需要重新配置的问题。要解决这个问题，您需要在在每次引导客户机虚拟机后将 VF 分配给主机物理机器前设置 MAC 地址。要分配这个 MAC 地址以及其它选项，请参阅以下步骤：

过程 16.9. 配置 MAC 地址、vLAN 和虚拟端口，以在 SR-IOV 上分配 PCI 设备

<hostdev> 元素不能用于特定于函数的项目，如 MAC 地址分配、vLAN 标签 ID 分配或虚拟端口分配，因为 <mac>、<vlan> 和 <virtualport> 元素都不是 <hostdev> 的有效子项。相反，这些元素可以和 hostdev 接口类型一起使用：<interface type='hostdev'>。这个设备类型作为 <接口和> <hostdev> 的混合方式运行。因此，在将 PCI 设备分配给客户机虚拟机之前，libvirt 将初始化指定网络特定硬件/交换机（如设置 MAC 地址、设置 vLAN 标签）或与客户机虚拟机 XML 配置文件中的 802.1Qbh 开关关联。有关设置 vLAN 标签的详情，请参考 第 17.16 节 “设置 vLAN 标签”。

1. 收集信息

   要使用 <接口类型='hostdev'>，您必须具有支持 SR-IOV 的网卡，托管支持 Intel VT-d 或 AMD IOMMU 扩展的主机物理硬件，且必须知道您要分配的 VF 的 PCI 地址。

2. 关闭客户端虚拟机

使用 virsh shutdown 命令，关闭 guest 虚拟机 （这里名为 guestVM）。

```
# virsh shutdown guestVM
```

3. 打开 XML 文件进行编辑

```
# virsh edit guestVM.xml
```

可选 ：对于由 virsh save 命令创建的 XML 配置文件，请运行：

```
# virsh save-image-edit guestVM.xml --running
```

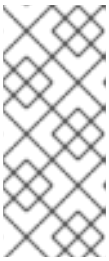
此示例中的 guestVM.xml 配置文件在默认编辑器中打开。如需更多信息，请参阅 第 20.7.5 节 “编辑客户机虚拟机配置”。

4. 编辑 XML 文件

更新配置文件(guestVM.xml)，使其具有类似如下的 <设备> 条目：

图 16.11. hostdev 接口类型的域 XML 示例

```
<devices>
  ...
  <interface type='hostdev' managed='yes'>
    <source>
      <address type='pci' domain='0x0' bus='0x00' slot='0x07' function='0x0'/> <!--these
values can be decimal as well-->
    </source>
    <mac address='52:54:00:6d:90:02'/>                    <!--sets the mac address--
>
    <virtualport type='802.1Qbh'>                    <!--sets the virtual port for the
802.1Qbh switch-->
      <parameters profileid='finance'/>
    </virtualport>
    <vlan>                                          <!--sets the vlan tag-->
     <tag id='42'/>
    </vlan>
  </interface>
  ...
</devices>
```



注意

如果没有提供 MAC 地址，系统将自动生成，就像其他类型的接口设备一样。另外，只有当您连接到 802.11Qgh 硬件交换机时，才会使用 <virtualport> 元素。不支持 802.11Qbg（也称为 "VEPA"）交换机。

5. 重启客户机虚拟机

运行 virsh start 命令，以重新启动您在第 2 步中关闭的 guest 虚拟机。如需更多信息，请参阅 第 20.6 节 “启动、恢复和恢复虚拟机”。

```
# virsh start guestVM
```

当客户机虚拟机启动时，它会看到由物理主机机器适配器提供的网络设备，且配置了 MAC 地址。此 MAC 地址在客户机虚拟机之间保持不变，主机物理机重新引导。

16.2.4. 从 SR-IOV 虚拟功能池设置 PCI 设备分配

将特定虚拟功能(VF)的 PCI 地址硬编码到客户机配置中有两个严重的限制：

- 任何虚拟机启动后，指定的 VF 都必须可用。因此，管理员必须将每个 VF 永久分配给一个客户机虚拟机（或者修改每个客户机虚拟机的配置文件，以便在每次启动客户机虚拟机时都指定目前未使用的 VF 的 PCI 地址）。
- 如果 guest 虚拟机移动到另一台主机物理计算机，则该主机物理计算机必须在 PCI 总线上的同一位置拥有完全相同的硬件（或者启动之前必须修改客户机虚拟机配置）。

通过在包含 SR-IOV 设备的所有 VF 的设备池创建一个 libvirt 网络，从而避免这两个问题。完成后，配置 guest 虚拟机以引用此网络。每次启动客户机时，都会从池中分配单个 VF，并分配给客户机虚拟机。当客户机虚拟机停止后，VF 将返回到池，供其他客户机虚拟机使用。

过程 16.10. 创建设备池

1. 关闭客户端虚拟机

使用 virsh shutdown 命令，关闭名为 guestVM 的客户机虚拟机。

```
# virsh shutdown guestVM
```

2. 创建配置文件

使用首选的编辑器，在 /tmp 目录中创建一个 XML 文件（名为 passthrough.xml ）。确保将 pf dev='eth3' 替换为您自己的 SR-IOV 设备物理功能(PF)的 netdev 名称。

以下是一个示例网络定义，该定义将在主机物理机器上使用 PF（在主机物理机器上使用 PF）提供 SR-IOV 适配器的所有 VF 池：

图 16.12. 网络定义域 XML 示例

```
<network>
  <name>passthrough</name> <!-- This is the name of the file you created -->
  <forward mode='hostdev' managed='yes'>
    <pf dev='myNetDevName'/>  <!-- Use the netdev name of your SR-IOV devices PF
here -->
  </forward>
</network>
```

3. 加载新 XML 文件

输入以下命令，将 /tmp/passthrough.xml 替换为您在上一步中创建的 XML 文件的名称和位置：

```
# virsh net-define /tmp/passthrough.xml
```

4. 重启客户端

运行以下命令，将 passthrough.xml 替换为您在上一步中创建的 XML 文件的名称：

```
# virsh net-autostart passthrough # virsh net-start passthrough
```

5. 重新启动 guest 虚拟机

运行 virsh start 命令以在第一步中重启您关闭的客户机虚拟机（例如，使用 guestVM 作为客户机虚拟机的域名）。如需更多信息，请参阅 第 20.6 节 “启动、恢复和恢复虚拟机”。

```
# virsh start guestVM
```

6. 启动设备的透传

虽然只会显示单一设备，但在客户机虚拟机首次启动时，libvirt 将自动获得与该 PF 关联的所有 VF 列表，如下所示：

图 16.13. 接口网络定义的域 XML 示例

```
<interface type='network'>
  <source network='passthrough'>
</interface>
```

7. *验证*

在启动使用网络的第一个客户机后，您可以运行 virsh net-dumpxml passthrough 命令进行验证；您可能会得到类似如下的输出：

图 16.14. XML 转储文件 透传 内容

```
<network connections='1'>
  <name>passthrough</name>
  <uuid>a6b49429-d353-d7ad-3185-4451cc786437</uuid>
  <forward mode='hostdev' managed='yes'>
    <pf dev='eth3'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10' function='0x1'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10' function='0x3'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10' function='0x5'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x10' function='0x7'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x11' function='0x1'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x11' function='0x3'/>
    <address type='pci' domain='0x0000' bus='0x02' slot='0x11' function='0x5'/>
  </forward>
</network>
```

16.2.5. SR-IOV 限制

SR-IOV 只使用以下设备进行了彻底测试：

- *Intel® 82576NS Gigabit Ethernet Controller（igb 驱动程序）*
- *Intel® 82576EB Gigabit Ethernet Controller（igb 驱动程序）*
- *Intel® 82599ES 10 Gigabit Ethernet Controller（ixgbe 驱动程序）*

- Intel® 82599EB 10 Gigabit Ethernet Controller（ixgbe 驱动程序）

其他 SR-IOV 设备可能可以正常工作，但没有在发布时测试

16.3. USB 设备

本节提供了处理 USB 设备所需的命令。

16.3.1. 为客户机虚拟机分配 USB 设备

Web 相机、卡读取器、磁盘驱动器、键盘等大多数设备都使用 USB 端口和电缆连接到计算机。有两种方法可将这些设备传递给客户端虚拟机：

- 使用 USB 透传 - 这需要将设备物理连接到托管客户机虚拟机的主机物理计算机。在这种情况下不需要 SPICE。主机上的 USB 设备可以在命令行或 virt-manager 中传递到 guest。有关 virt Manager 方向，请参阅 第 19.3.2 节 “将 USB 设备附加到虚拟机”。请注意，virt-manager 指令不适合热插拔或热拔设备。如果要热插拔/或热拔 USB 设备，请参阅 过程 20.4, “热插拔 USB 设备供客户机虚拟机使用”。

- 使用 USB 重定向 - USB 在数据中心中运行的主机物理计算机时最好使用 USB 重定向。用户从本地机器或瘦客户端连接到其/及客户机虚拟机。此本地机器上有 SPICE 客户端。用户可以将任何 USB 设备附加到瘦客户端，SPICE 客户端会将设备重定向到数据中心中的主机物理机器，以便供瘦客户端上运行的客户机虚拟机使用。有关通过 virt-manager 的说明，请参阅 第 19.3.3 节 “USB 重定向”。

16.3.2. 在 USB 设备重定向中设置限制

要从重定向过滤出某些设备，请将 filter 属性传递给 -device usb-redir。filter 属性使用一个由过滤规则组成的字符串，规则的格式是：

```
<class>:<vendor>:<product>:<version>:<allow>
```

使用 -1 值指定它接受特定字段的任何值。您可以使用 | 作为分隔符，在同一命令行中使用多个规则。请注意，如果设备不匹配通过规则，则不允许重定向该设备！

例 16.1. 使用客户机虚拟机限制重定向的示例

1. 准备客户机虚拟机。

2. 在客户机虚拟机的域 XML 文件中添加以下代码摘录：

```
<redirdev bus='usb' type='spicevmc'>
  <alias name='redir0'/>
  <address type='usb' bus='0' port='3'/>
</redirdev>
<redirfilter>
  <usbdev class='0x08' vendor='0x1234' product='0xBEEF' version='2.0' allow='yes'/>
  <usbdev class='-1' vendor='-1' product='-1' version='-1' allow='no'/>
</redirfilter>
```

3. 启动客户机虚拟机并通过运行以下命令确认设置更改：

```
#ps -ef | grep $guest_name
```

```
-device usb-redir,chardev=charredir0,id=redir0,/
filter=0x08:0x1234:0xBEEF:0x0200:1|-1:-1:-1:-1:0,bus=usb.0,port=3
```

4. 将 USB 设备插入到主机物理计算机中，并使用 virt-manager 连接到客户机虚拟机。

5. 在菜单中点击 USB 设备选择，这将生成以下信息："主机策略会阻止某些 USB 设备"。单击确定以确认 并继续。

   过滤器生效。

6. 为确保过滤器捕获正确检查 USB 设备供应商和产品，然后在主机物理计算机的域 XML 中进行以下更改以允许 USB 重定向。

```
<redirfilter>
  <usbdev class='0x08' vendor='0x0951' product='0x1625' version='2.0' allow='yes'/>
  <usbdev allow='no'/>
</redirfilter>
```

7. 重新启动 guest 虚拟机，然后使用 virt-viewer 连接到客户机虚拟机。USB 设备现在会将流量重定向到客户机虚拟机。

16.4. 配置设备控制器

根据客户机虚拟机架构，一些设备总线可能多次出现，有一组绑定到虚拟控制器的虚拟设备。通常，libvirt 可以自动推断这样的控制器，无需显式 XML 标记，但在某些情况下最好显式设置虚拟控制器元素。

图 16.15. 虚拟控制器的域 XML 示例

```
...
<devices>
  <controller type='ide' index='0'/>
  <controller type='virtio-serial' index='0' ports='16' vectors='4'/>
  <controller type='virtio-serial' index='1'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x0a' function='0x0'/>
  </controller>
  ...
</devices>
...
```

每个控制器都有一个强制属性 <控制器类型>，它必须是以下之一：

- IDE
- fdc
- scsi
- SATA
- usb
- ccid
- virtio-serial

- *pci*

<controller> 元素有一个强制属性 <控制器索引>，它是一个十进制整数，用于描述总线控制器被遇到的顺序（用于 <地址> 元素的控制器属性）。当控制器 <type ='virtio-serial'> 时，有两个额外可选属性（指定端口 和 向量），它们控制可以通过控制器连接的设备数量。

<当控制器 type ='scsi'> 时，有一个可选属性 模型，它可采用以下值：

- *auto*
- *buslogic*
- *ibmvscsi*
- *lsilogic*
- *lsisas1068*
- *lsisas1078*
- *virtio-scsi*
- *vmpvscsi*

当 <控制器类型 ='usb'> 时，有一个可选属性 模型，它可采用以下值：

- *piix3-uhci*
- *piix4-uhci*

- ehci
- ich9-ehci1
- ich9-uhci1
- ich9-uhci2
- ich9-uhci3
- vt82c686b-uhci
- pci-ohci
- nec-xhci

请注意，如果需要为客户机虚拟机明确禁用 USB 总线，可以使用 <model='none'>。

对于作为 PCI 或 USB 总线中的设备本身的控制器，可选的子元素 <地址> 可以指定控制器到其主总线的确切关系，使用 第 16.5 节 “为设备设置地址” 所示的语义。

可选的 sub-element <驱动程序> 可以指定特定于驱动程序的选项。目前，它只支持属性队列，它指定控制器的队列数。为了获得最佳性能，建议指定一个与 vCPU 数量匹配的值。

USB 配套控制器有一个可选的子元素 <master>，以指定与其主控制器相配的确切关系。配套控制器位于与其 master 的同一总线上，因此相应的 索引 值应该相等。

可以使用的 XML 示例：

图 16.16. USB 控制器的域 XML 示例

```
  ...
<devices>
  <controller type='usb' index='0' model='ich9-ehci1'>
    <address type='pci' domain='0' bus='0' slot='4' function='7'/>
  </controller>
  <controller type='usb' index='0' model='ich9-uhci1'>
    <master startport='0'/>
    <address type='pci' domain='0' bus='0' slot='4' function='0' multifunction='on'/>
  </controller>
  ...
</devices>
...
```

PCI 控制器具有具有以下可能值的可选 模型 属性：

- **pci-root**
- **pcie-root**
- **pci-bridge**
- **dmi-to-pci-bridge**

对于提供隐式 PCI 总线的机器类型，pci-root 控制器使用 index='0' 是自动添加的，且需要使用 PCI 设备。pci-root 没有地址。如果对由 model='pci-root' 或指定大于零的 PCI 总线，则存在太多的设备适合一个总线上，则会自动添加 PCI 网桥。也可以手动指定 PCI 网桥，但其地址应该只看到已经指定的 PCI 控制器提供的 PCI 总线。在 PCI 控制器索引中造成空白可能会导致无效的配置。以下 XML 示例可添加到 <devices> 部分：

图 16.17. PCI 网桥的域 XML 示例

```
...
<devices>
  <controller type='pci' index='0' model='pci-root'/>
  <controller type='pci' index='1' model='pci-bridge'>
    <address type='pci' domain='0' bus='0' slot='5' function='0' multifunction='off'/>
  </controller>
</devices>
...
```

对于提供隐式 PCI Express 总线（例如，基于 Q35 芯片组的机器类型），带有 index='0' 的 pcie-root 控制器已自动添加到域的配置。pcie-root 也没有地址，但提供 31 插槽（数字为 1-31），且只能附加到 PCIe 设备。要在具有 pcie-root 控制器的系统中连接标准 PCI 设备，会自动添加带有 model='dmi-to-pci-bridge 的 pci 控制器。dmi-to-pci-bridge 控制器插件到一个 PCIe 插槽（由 pcie-root 提供），并且自己提供 31 标准 PCI 插槽（这不是热插拔的）。为了在客户机系统中具有热插拔 PCI 插槽，则也会自动创建 pci-bridge 控制器，并连接到自动创建的 dmi-to-pci-bridge 控制器中的一个插槽；所有具有 PCI 地址的客户机设备均会被放置到此 pci-bridge 设备上。

图 16.18. PCIe 的域 XML 示例(PCI express)

```
  ...
<devices>
  <controller type='pci' index='0' model='pcie-root'/>
  <controller type='pci' index='1' model='dmi-to-pci-bridge'>
    <address type='pci' domain='0' bus='0' slot='0xe' function='0'/>
  </controller>
  <controller type='pci' index='2' model='pci-bridge'>
    <address type='pci' domain='0' bus='1' slot='1' function='0'/>
  </controller>
</devices>
...
```

以下 XML 配置用于 USB 3.0 / xHCI 模拟：

图 16.19. USB3/xHCI 设备的域 XML 示例

```
  ...
<devices>
  <controller type='usb' index='3' model='nec-xhci'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x0f' function='0x0'/>
  </controller>
</devices>
  ...
```

16.5. 为设备设置地址

许多设备具有可选 <地址> 子元素，用于描述将设备放置在虚拟客户机的虚拟总线中的位置。如果在输入中省略了地址（或地址中的任何可选属性），libvirt 将生成适当的地址；但是，如果需要更多地控制布局，则需要明确的地址。有关包含 <地址> 元素的域 XML 设备示例，请参考 图 16.9 “PCI 设备分配的 XML 示例”。

每个地址都有一个强制属性 类型，用于描述该设备所在的总线。选择为给定设备使用的地址将在设备和客户机虚拟机的架构中限制。例如，<磁盘设备> 使用 type='drive'，而 <控制台> 设备在 i686 或 x86_64 客户机虚拟机架构上使用 type='pci'。每个地址类型都有进一步可选的属性，这些属性控制该设备在总线中的位置，如表中所述：

表 16.1. 支持的设备类型

地址类型	描述
type='pci'	PCI 地址有以下额外属性： • 域（2 字节十六进制整数，目前不供 qemu 使用） • 总线（包含 0 到 0xff 的十六进制值） • 插槽（包含 0x0 到 0x1f 的十六进制值） • 功能（包含 0 到 7 之间的值） • 在 PCI 控制中，对于 PCI 控制寄存器的特定插槽/功能，多功能控制打开了多功能，默认情况下，它被设置为 'off'，但应设置为 'on'，用于使用多个功能的插槽 0。
type='drive'	驱动器地址有以下额外属性： • 控制器（2 位控制器数） • 总线（一个 2 位总线号 • 目标（2 位总线号） • 单元（总线中的 2 位单元号）
type='virtio-serial'	每个 virtio-serial 地址有以下额外属性： • 控制器（2 位控制器数） • 总线（2 位总线号） • 插槽（总线中的 2 位插槽）

地址类型	描述
type='ccid'	用于 smart-cards 的 CCID 地址有以下额外属性： • 总线（2 位总线号） • 插槽属性（总线中的 2 位插槽）
type='usb'	USB 地址有以下额外属性： • 总线（包含 0 到 0xfff 的十六进制值） • 端口（最多四个八位字节表示法，如 1.2 或 2.1.3.1）
type='isa'	ISA 地址有以下额外属性： • iobase • IRQ

16.6. 随机数字生成器设备

随机数生成器对于操作系统安全性非常重要。为了保护虚拟操作系统，红帽企业 Linux 7 包含 virtio-rng，这是一个虚拟硬件随机数生成器设备，可根据需要为 guest 提供全新的熵。

在主机物理计算机上，硬件 RNG 接口在 /dev/hwrng 处创建一个 chardev，它可以打开，然后从主机物理机器读取提取熵。在与 rngd 守护进程的合作中，主机物理计算机中的熵可以路由到客户机虚拟机的 /dev/random，这是随机性的主要来源。

当键盘、鼠标和其他输入等设备在客户机虚拟机上生成足够的熵时，使用随机数生成器特别有用。虚拟随机数生成器设备允许主机物理计算机将熵传递到客户端虚拟机操作系统。此过程可以通过命令行或 virt-manager 接口执行。具体步骤请查看以下操作。有关 virtio-rng 的详情请参考 Red Hat Enterprise Linux 虚拟机：访问随机数字 Made Easy。

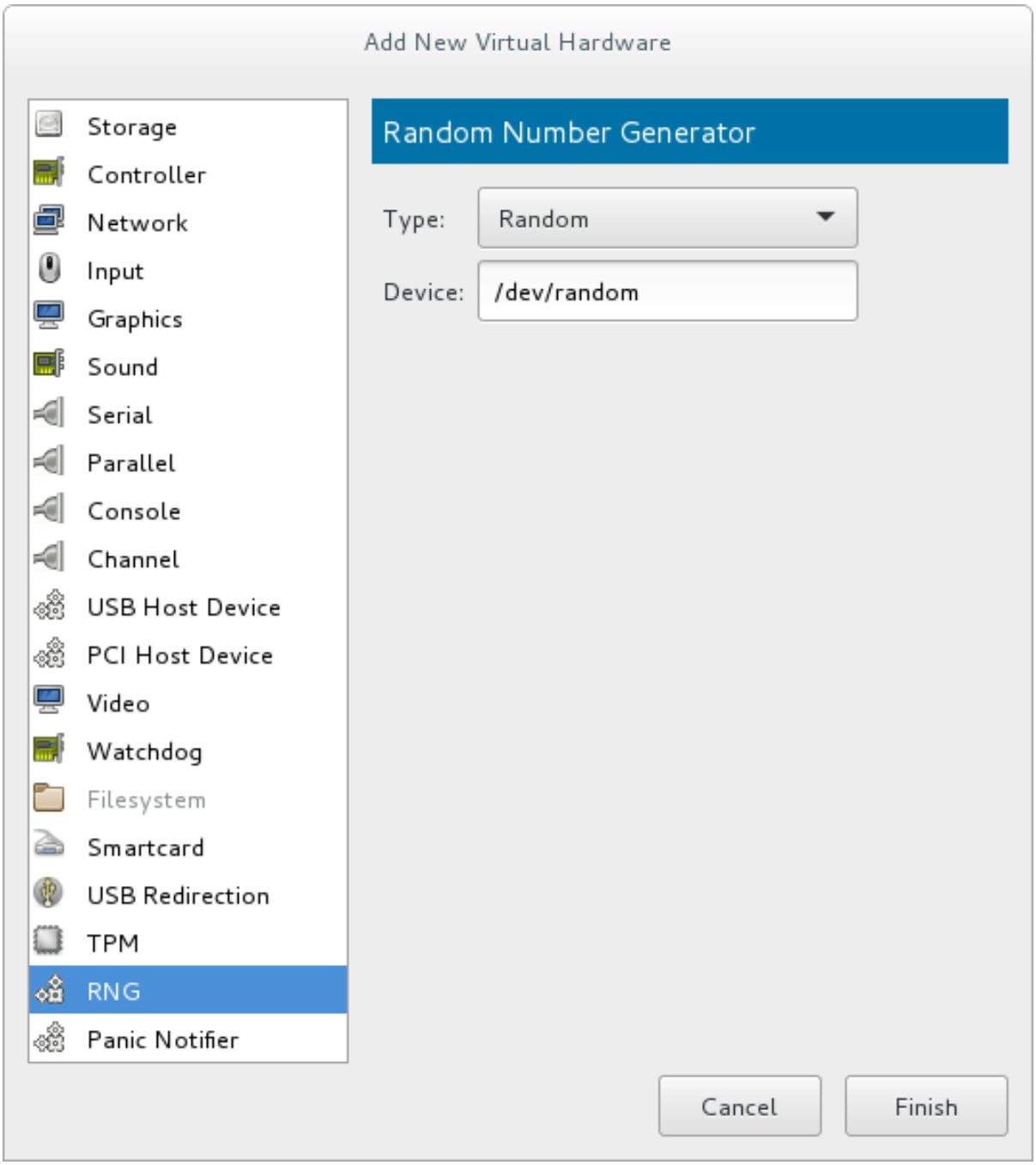
过程 16.11. 使用虚拟机管理器实施 virtio-rng

1. *关闭客户机虚拟机。*
2. *选择 guest 虚拟机并从 编辑 菜单中选择 Virtual Machine Details，以打开指定 guest 虚拟机的 Details 窗口。*

3. 点 添加硬件 按钮。

4. 在 Add New Virtual Hardware 窗口中，选择 RNG 打开 Random Number Generator 窗口。

图 16.20. 随机数字生成器窗口



输入预期参数，然后在完成时点击 Finish。参数在 virtio-rng 元素 中进行说明。

过程 16.12. 使用命令行工具实施 virtio-rng

1. 关闭客户机虚拟机。

2. 使用 virsh edit domain-name 命令，为所需的 guest 虚拟机打开 XML 文件。

3. 编辑 <devices> 元素以包括以下内容：

图 16.21. 随机数生成器设备

```
...
<devices>
  <rng model='virtio'>
    <rate period='2000' bytes='1234'/>
    <backend model='random'>/dev/random</backend>
    <!-- OR -->
    <backend model='egd' type='udp'>
      <source mode='bind' service='1234'/>
      <source mode='connect' host='1.2.3.4' service='1234'/>
    </backend>
  </rng>
</devices>
...
```

随机数字生成器设备允许以下 XML 属性和元素：

virtio-rng 元素

- <model> - 必需的 model 属性指定提供 RNG 设备的类型。

- <后端模型> - <后端> 元素指定用于客户机的熵源。源模型使用 model 属性进行配置。支持的源模型包括 'random' 和 'egd' 。

  - < <backend> model='random'> - 此后端类型需要一个非阻塞字符设备作为输入。此类设备的示例为 /dev/random 和 /dev/urandom。文件名被指定为 <backend> 元素的内容。如果没有指定文件名，则使用虚拟机监控程序默认。

  - <backend model='egd'> - 这个后端使用 EGD 协议连接到源。源指定为字符设备。如需更多信息，请参阅字符设备主机物理机器接口。

16.7. 分配 GPU 设备

要将 GPU 分配给客户端，请使用以下方法之 一：

- GPU PCI 设备分配 - 使用此方法，可以从主机中删除 GPU 设备并将其分配给单个客户端。
- NVIDIA vGPU 分配 - 这个方法可以从物理 GPU 创建多个介质设备，并将这些设备分配为多个客户端。这只在所选 NVIDIA GPU 上被支持，且只能将一个介质设备分配给单个客户端。

16.7.1. GPU PCI 设备分配

Red Hat Enterprise Linux 7 支持将以下基于 PCIe 的 GPU 设备分配为非VGA 图形设备：

- NVIDIA Quadro K-Series、M-Series、P-Series 以及更新的构架（型号为 2000 系列或更高版本）
- NVIDIA Tesla K-Series、M-Series 和更高的架构

注意

可附加到虚拟机的 GPU 数量受分配的最大 PCI 设备数的限制，在 RHEL 7 中，当前为 32。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 不可用。

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。

要将 GPU 分配给客户机虚拟机，您必须在主机中启用 I/O 内存管理单元(IOMMU)，使用 lspci 命令确定 GPU 设备，将设备从主机分离，将其附加到客户端，并在客户端 - 如以下步骤中所述：

过程 16.13. 在主机机器内核中启用 IOMMU 支持

1. 编辑内核命令行

对于 Intel VT-d 系统，IOMMU 通过在内核命令行中添加 intel_iommu=on 和 iommu=pt 参数来激活。对于 AMD-Vi 系统，所需的选项仅为 iommu=pt。要启用这个选项，请编辑或将 GRUB_CMDLINX_LINUX 行添加到 /etc/sysconfig/grub 配置文件，如下所示：

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=vg_VolGroup00/LogVol01
vconsole.font=latarcyrheb-sun16 rd.lvm.lv=vg_VolGroup_1/root
vconsole.keymap=us $([ -x /usr/sbin/rhcrashkernel-param ]  &&
/usr/sbin/rhcrashkernel-param || :) rhgb quiet intel_iommu=on iommu=pt"
```

注意

有关 IOMMU 的详情，请参考 附录 E, 使用 IOMMU 组。

2. 重新生成引导装载程序配置

要应用内核命令行的更改，请使用 grub2-mkconfig 命令重新生成引导装载程序配置：

```
# grub2-mkconfig -o /etc/grub2.cfg
```

请注意，如果您使用基于 UEFI 的主机，则目标文件应当是 /etc/grub2-efi.cfg。

3. 重启主机

要使更改生效，请重启主机机器：

```
# reboot
```

过程 16.14. 将 GPU 设备绑定到主机物理机器驱动程序中排除

对于 GPU 分配，建议将设备从绑定到主机驱动程序中排除，因为这些驱动程序通常不支持动态未绑定设备。

1. 识别 PCI 总线地址

要识别设备的 PCI 总线地址和 ID，请运行以下 lspci 命令。在本例中，使用的是 VGA 控制器，如 NVIDIA Quadro 或 GRID 卡：

```
# lspci -Dnn | grep VGA
0000:02:00.0 VGA compatible controller [0300]: NVIDIA Corporation GK106GL [Quadro
K4000] [10de:11fa] (rev a1)
```

生成的搜索显示，这个设备的 PCI 总线地址为 0000:02:00.0，该设备的 PCI ID 为 10de:11fa。

2. 防止原生主机机器驱动程序使用 GPU 设备

要防止原生主机机器驱动程序使用 GPU 设备，您可以使用带有 pci-stub 驱动程序的 PCI ID。要做到这一点，将 pci-stub.ids 选项及其值附加到位于 /etc/sysconfig/grub 配置文件中的 GRUB_CMDLINX_LINUX 行，例如：

```
GRUB_CMDLINE_LINUX="rd.lvm.lv=vg_VolGroup00/LogVol01
vconsole.font=latarcyrheb-sun16 rd.lvm.lv=vg_VolGroup_1/root
vconsole.keymap=us $([ -x /usr/sbin/rhcrashkernel-param ]  &&
/usr/sbin/rhcrashkernel-param || :) rhgb quiet intel_iommu=on iommu=pt pci-
stub.ids=10de:11fa"
```

要为 pci-stub 添加额外的 PCI ID，使用逗号分隔它们。

3. 重新生成引导装载程序配置

使用 grub2-mkconfig 来重新生成引导装载程序配置，使其包含这个选项：

```
# grub2-mkconfig -o /etc/grub2.cfg
```

请注意，如果您使用基于 UEFI 的主机，则目标文件应当是 /etc/grub2-efi.cfg。

4. 重启主机机器

要使更改生效，重启主机机器：

```
# reboot
```

过程 16.15. 可选：编辑 GPU IOMMU 配置

在附加 GPU 设备前，可能需要编辑其 IOMMU 配置以便 GPU 在客户机上正常工作。

1. 显示 GPU 的 XML 信息

要以 XML 格式显示 GPU 的设置，您首先需要将其 PCI 总线地址转换为兼容 libvirt 的格式，方法是附加 pci_ 和将分隔符转换为下划线。在本例中，通过 0000:02:00.0 总线地址标识的 GPU PCI 设备（如上一步中获取）将变为 pci_0000_02_00_0。使用 virsh nodedev-dumpxml 的设备 libvirt 地址显示其 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>
    <!-- pay attention to the following lines -->
  <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>
```

注意 XML 的 <iommuGroup> 元素。iommuGroup 表示一组由于 IOMMU 功能和 PCI 总线拓扑考虑了与其他设备隔离的设备。iommuGroup 内的所有端点设备（认为不是 PCIe 根端口、网桥或交换机端口的设备）都需要从原生主机驱动程序中绑定，才能分配到客户机。在上面的示例中，组由 GPU 设备(0000:02:00.0)和 companion 音频设备(0000:02:00.1)组成。如需更多信息，请参阅 附录 E, 使用 IOMMU 组。

2. *调整 IOMMU 设置*

在本例中，不支持分配 NVIDIA 音频功能，因为存在旧中断支持的硬件问题。另外，在没有 GPU 本身的情况下，G GPU 音频功能通常不有用。因此，为了将 GPU 分配给客户机，必须首先将音频功能与原生主机驱动程序分离。这可使用以下方法之一完成：

- *检测设备的 PCI ID，并将其附加到 /etc/sysconfig/grub 文件中的 pci-stub.ids 选项中，具体如 过程 16.14, “将 GPU 设备绑定到主机物理机器驱动程序中排除”*

- *使用 virsh nodedev-detach 命令，如下所示：*

```
# virsh nodedev-detach pci_0000_02_00_1
Device pci_0000_02_00_1 detached
```

过程 16.16. 附加 GPU

GPU 可使用以下任一方法附加到客户端：

1. 使用 虚拟机管理器 接口.详情请查看 第 16.1.2 节 “使用 virt-manager 分配 PCI 设备”。

2. 为 GPU 创建 XML 配置片段并使用 virsh attach-device 附加它：

    1. 为该设备创建 XML，如下所示：

```
<hostdev mode='subsystem' type='pci' managed='yes'>
 <driver name='vfio'/>
 <source>
  <address domain='0x0000' bus='0x02' slot='0x00' function='0x0'/>
 </source>
</hostdev>
```

    2. 将它保存到文件中，并运行 virsh attach-device [domain] [file] --persistent，以在客户机配置中包含 XML。请注意，除了客户机机器中现有仿真图形设备外，还添加了分配的 GPU。分配的 GPU 作为虚拟机中的辅助图形设备进行处理。不支持作为主图形设备分配，不应删除 guest XML 中的模拟图形设备。

3. 使用 virsh edit 命令编辑客户机 XML 配置并手动添加相应的 XML 段。

过程 16.17. containerruntimeodify 在客户机上的 Xorg 配置

客户机上的 GPU PCI 总线地址将与主机上的不同。要启用主机正确使用 GPU，将 guest 的 Xorg 显示服务器配置为使用分配的 GPU 地址：

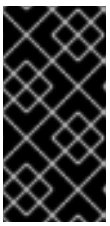
1. 在客户机中，使用 lspci 命令确定 GPU 的 PCI 总线适配器：

```
# lspci | grep VGA
00:02.0 VGA compatible controller: Device 1234:111
00:09.0 VGA compatible controller: NVIDIA Corporation GK106GL [Quadro K4000] (rev a1)
```

在本例中，总线地址为 00:09.0。

2. 在客户机上的 /etc/X11/xorg.conf 文件中，在检测到的地址添加一个 BusID 选项，如下所示：

```
Section "Device"
    Identifier     "Device0"
    Driver         "nvidia"
    VendorName     "NVIDIA Corporation"
    BusID          "PCI:0:9:0"
EndSection
```



重要

如果第 1 步中检测到的总线地址是十六进制的，您需要将分隔符之间的值转换为十进制系统。例如，00:0a.0 应转换为 PCI:0:10:0。

注意

当在客户机中使用分配的 NVIDIA GPU 时，只支持 NVIDIA 驱动程序。其他驱动程序可能无法正常工作，并可能会生成错误。对于 Red Hat Enterprise Linux 7 客户机，可以在安装期间在内核命令行中使用 modprobe.blacklist=nouveau 选项将 nouveau 驱动程序列入黑名单。有关其他客户虚拟机的详情，请查看操作系统的特定文档。

根据载入 NVIDIA 驱动程序的客户机操作系统，客户机可能会同时使用仿真图形和分配的图形支持，也可以禁用模拟图形。请注意，对分配的图形帧缓冲的访问不是由 virt-manager 等应用程序提供的。如果分配的 GPU 未连接到物理显示，则可能需要基于 guest 的删除解决方案来访问 GPU 桌面。与所有 PCI 设备分配一样，不支持迁移带有分配 GPU 的客户机，每个 GPU 都由单个虚拟机所有。根据客户端操作系统，可能会对 GPU 的热插拔支持可用。

16.7.2. NVIDIA vGPU 分配

NVIDIA vGPU 功能可以将物理 GPU 设备划分为多个虚拟设备（称为 介质设备 ）。然后可将这些 mediated devices 分配给多个客户机，作为虚拟 GPU。因此，这些客户机共享单个物理 GPU 的性能。

重要

这个功能仅适用于有限的 NVIDIA GPU。有关这些设备的最新列表，请参阅 NVIDIA GPU 软件文档。

16.7.2.1. NVIDIA vGPU 设置

要设置 vGPU 功能，您首先需要为 GPU 设备获取 NVIDIA vGPU 驱动程序，然后创建介质设备，并将其分配给预期的客户端机器：

1. 获取 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/ 目录中创建 .conf 文件（名称）。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，然后重启：

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

如果您需要使用带有介质设备支持的内核版本，请为所有安装的内核版本重新生成初始 ramdisk：

```
# dracut --regenerate-all --force
# reboot
```

4. 检查 nvidia_vgpu_vfio 模块是否已由内核加载，且 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)
 [...]
```

5. 将设备 UUID 写入 /sys/class/mdev_bus/pci_dev/mdev_supported_types/type-id/create，其中 pci_dev 是主机 GPU 的 PCI 地址，type-id 是主机 GPU 类型的 ID。

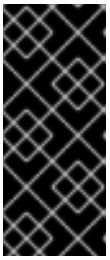
以下示例演示了如何在 NVIDIA Tesla P4 卡上创建 nvidia-63 vGPU 类型的介质设备：

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

有关特定设备的 type-id 值，请参阅 section 1.3.1。虚拟 GPU 类型 在 Virtual GPU 软件文档中。请注意，只有 Q series NVIDIA vGPUs（如 GRID P4-2Q ）作为 Linux 客户机上的介质设备 GPU 类型被支持。

6. 在您要共享 vGPU 资源的 XML 配置中的 <devices/> 部分中添加以下行。使用上一步中 uuidgen 命令生成的 UUID 值。每个 UUID 每次只能分配给一个 guest。

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci'>
  <source>
    <address uuid='30820a6f-b1a5-4503-91ca-0c10ba58692a'/>
  </source>
</hostdev>
```



重要

要让 vGPU 介质设备在分配的客户机中正常工作，需要为客户机设置 NVIDIA vGPU 客户机软件许可。如需更多信息和说明，请参阅 NVIDIA 虚拟 GPU 软件文档。

16.7.2.2. 通过 NVIDIA vGPU 设置并使用 VNC 控制台进行视频流

作为技术预览，虚拟网络计算(VNC)控制台可与基于 GPU 的介质设备一起使用，包括在 Red Hat Enterprise Linux 7.7 及之后的版本中。因此，您可以使用 VNC 显示 NVIDIA vGPU 设备提供的加速图形输出。

警告

这个功能目前仅作为技术预览提供，且不受红帽支持。因此，不建议在生产环境中使用以下过程。

要在虚拟机上的 VNC 控制台中配置 vGPU 输出渲染，请执行以下操作：

1. *在您的系统中安装 NVIDIA vGPU 驱动程序并配置 NVIDIA vGPU，如 第 16.7.2.1 节 "NVIDIA vGPU 设置" 所述。确保介质设备的 XML 配置包含 display='on' 参数。例如：*

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci' display='on'>
  <source>
    <address uuid='ba26a3e2-8e1e-4f39-9de7-b26bd210268a'/>
  </source>
</hostdev>
```

2. *（可选）将虚拟机的视频模型类型设置为 none。例如：*

```
<video>
  <model type='none'/>
</video>
```

如果没有指定，您会收到两个不同的显示输出 - 一个来自于模拟的 Cirrus 或 QXL 卡，并从 NVIDIA vGPU 中获得。另请注意，使用 模型 type='none' 目前无法看到引导图形输出，直到驱动程序初始化为止。因此，显示的第一个图形输出是登录屏幕。

3. *确定虚拟机图形类型的 XML 配置是 vnc。*

例如：

```
<graphics type='vnc' port='-1' autoport='yes'>
  <listen type='address'/>
</graphics>
```

4. 启动虚拟机。

5. 使用 VNC viewer 远程桌面客户端连接到虚拟机。

注意

如果虚拟机设置了模拟 VGA 作为主视频设备，且 vGPU 作为二级设备，请使用 ctrl+alt+2 键盘快捷键切换到 vGPU 显示。

16.7.2.3. 删除 NVIDIA vGPU 设备

要删除 mediated vGPU 设备，请在设备不活跃时运行以下命令，并将 uuid 替换为设备的 UUID，例如 30820a6f-b1a5-4503-91ca-0c10ba58692a。

```
# echo 1 > /sys/bus/mdev/devices/uuid/remove
```

请注意，尝试删除当前供客户端使用的 vGPU 设备会触发以下错误：

```
echo: write error: Device or resource busy
```

16.7.2.4. 查询 NVIDIA vGPU 功能

要获得有关系统中介质设备的其他信息，如可以创建给定类型中的介质设备数量，请使用 virsh nodedev-list --cap mdev_types 和 virsh nodedev-dumpxml 命令。例如，以下在 Tesla P4 卡中显示可用的 vGPU 类型：

```
$ virsh nodedev-list --cap mdev_types
pci_0000_01_00_0
$ virsh nodedev-dumpxml pci_0000_01_00_0
<...>
  <capability type='mdev_types'>
    <type id='nvidia-70'>
      <name>GRID P4-8A</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>1</availableInstances>
    </type>
    <type id='nvidia-69'>
      <name>GRID P4-4A</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>2</availableInstances>
    </type>
    <type id='nvidia-67'>
      <name>GRID P4-1A</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>8</availableInstances>
    </type>
    <type id='nvidia-65'>
      <name>GRID P4-4Q</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>2</availableInstances>
    </type>
    <type id='nvidia-63'>
      <name>GRID P4-1Q</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>8</availableInstances>
    </type>
    <type id='nvidia-71'>
      <name>GRID P4-1B</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>8</availableInstances>
    </type>
    <type id='nvidia-68'>
      <name>GRID P4-2A</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>4</availableInstances>
    </type>
    <type id='nvidia-66'>
      <name>GRID P4-8Q</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>1</availableInstances>
    </type>
    <type id='nvidia-64'>
      <name>GRID P4-2Q</name>
      <deviceAPI>vfio-pci</deviceAPI>
      <availableInstances>4</availableInstances>
    </type>
  </capability>
</...>
```

16.7.2.5. 用于 NVIDIA vGPU 的远程桌面流服务

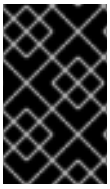
以下远程桌面流服务已被成功测试，以便与 Red Hat Enterprise Linux 7 上的 NVIDIA vGPU 功能一起使用：

- HP-RGS
- Mechdyne TGX - 目前无法将 Mechdyne TGX 与 Windows Server 2016 guest 搭配使用。
- NICE DCV - 当使用此流服务时，红帽建议使用固定解析设置，因为在一些情况下使用动态解析会导致一个黑屏。

16.7.2.6. 使用 NVIDIA vGPU 为视频流设置 VNC 控制台

简介

作为技术预览，虚拟网络计算(VNC)控制台可用于基于 GPU 的介质设备，包括 NVIDIA vGPU（包括在 Red Hat Enterprise Linux 8 中）。因此，您可以使用 VNC 显示 NVIDIA vGPU 设备提供的加速图形输出。



重要

由于是技术预览，红帽不支持此功能。因此，不建议在生产环境中使用以下过程。

Configuration

要在虚拟机上的 VNC 控制台中配置 vGPU 输出渲染，请执行以下操作：

1. 在您的主机上安装 NVIDIA vGPU 驱动程序并配置 NVIDIA vGPU，如 第 16.7.2 节 “NVIDIA vGPU 分配” 所述。确保介质设备的 XML 配置包含 display='on' 参数。例如：

```
<hostdev mode='subsystem' type='mdev' managed='no' model='vfio-pci' display='on'>
 <source>
   <address uuid='ba26a3e2-8e1e-4f39-9de7-b26bd210268a'/>
 </source>
</hostdev>
```

2. （可选）将虚拟机的视频模型类型设置为 none。例如：

```
<video>
 <model type='none'/>
</video>
```

3. 确定虚拟机图形类型的 XML 配置是 spice 或 vnc。

spice 的一个示例：

```
<graphics type='spice' autoport='yes'>
 <listen type='address'/>
 <image compression='off'/>
</graphics>
```

vnc 的示例：

```
<graphics type='vnc' port='-1' autoport='yes'>
 <listen type='address'/>
</graphics>
```

4. 启动虚拟机。

5. 使用适合您在前面步骤中配置的图形协议的客户端连接到虚拟机。

- 对于 VNC，使用 VNC viewer 远程桌面客户端。如果虚拟机设置了模拟 VGA 作为主视频设备，且 vGPU 作为辅助，请使用 ctrl+alt+2 键盘快捷键切换到 vGPU 显示。

- 对于 SPICE，请使用 virt-viewer 应用程序。

第 17 章 虚拟网络

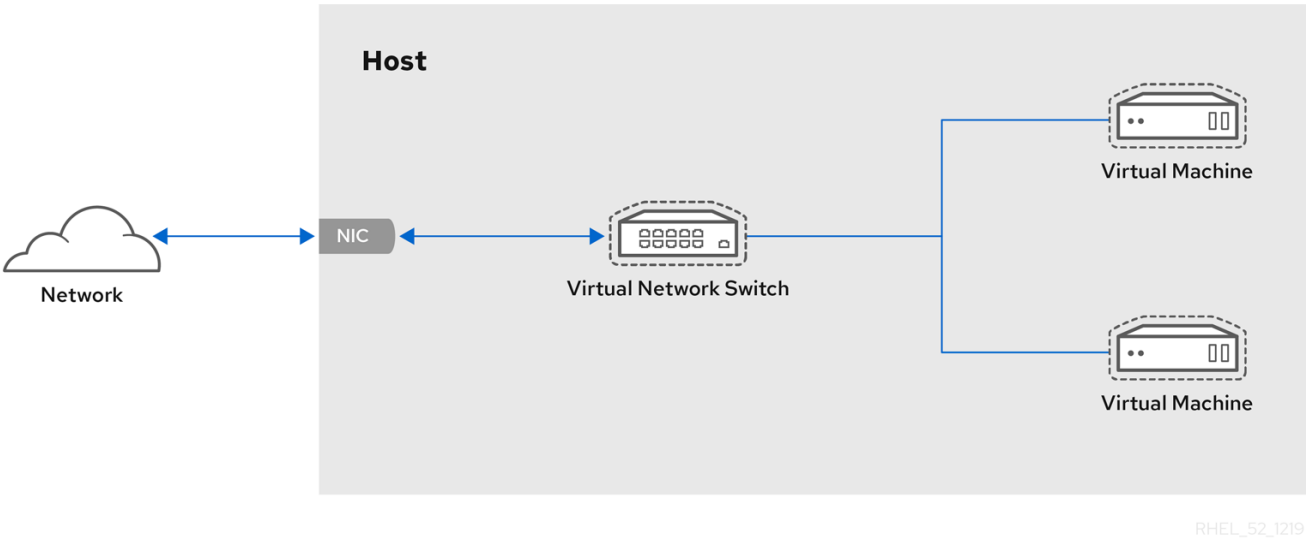
本章介绍了使用 libvirt 创建、启动、停止、删除和修改虚拟网络所需的概念。

更多信息，请参阅 libvirt 参考章节

17.1. 虚拟网络切换

libvirt 虚拟网络使用 虚拟网络交换机 的概念。虚拟网络交换机是在主机物理计算机服务器上运行的软件构造，虚拟机（虚拟机）连接。客户机的网络流量通过这个交换机定向：

图 17.1. 带两个客户机的虚拟网络交换机



Linux 主机物理计算机服务器表示一个虚拟网络交换机作为网络接口。当首先安装和启动libvirtd守护进程时，代表虚拟网络交换机的默认网络接口为 virbr0。

这种 virbr0 接口可以像任何其他接口一样通过 ip 命令来查看：

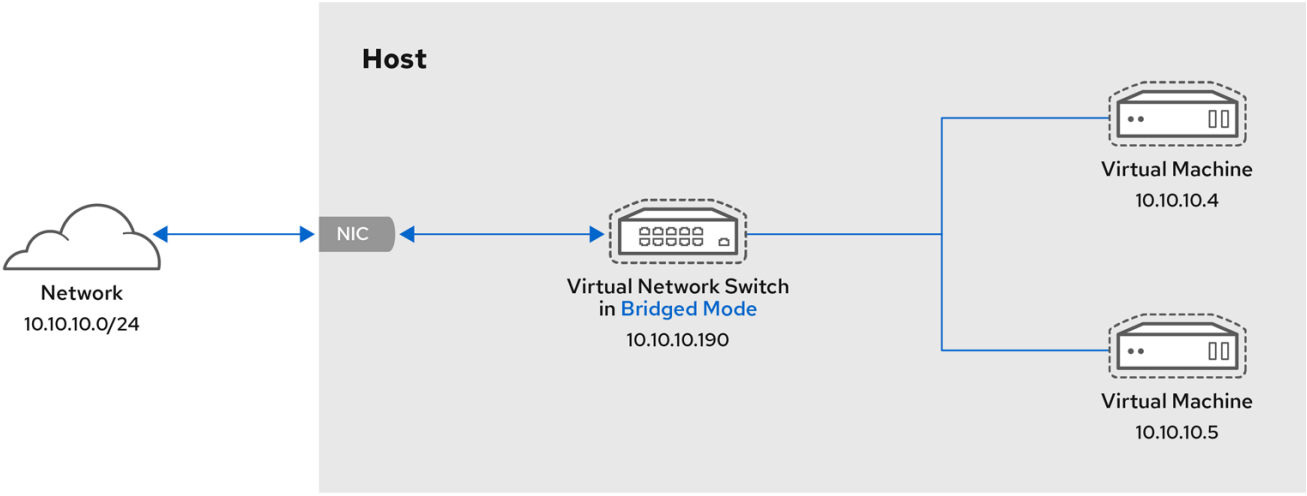
```
$ ip addr show virbr0
3: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    link/ether 1b:c4:94:cf:fd:17 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.1/24 brd 192.168.122.255 scope global virbr0
```

17.2. 网桥模式

在使用 桥接模式 时，所有客户机虚拟机都与主机物理机器位于同一个子网中。同一物理网络中的所有

其他物理机器都了解虚拟机，并可访问该虚拟机。桥接在 OSI 网络模型的第 2 层上运行。

图 17.2. 网桥模式的虚拟网络交换机



通过将它们与 绑定 一起加入，可以在虚拟机监控程序上使用多个物理接口。然后，这个绑定被添加到网桥中，客户端虚拟机也会添加到网桥中。但是，绑定驱动程序有多种操作模式，只有几个模式可以与使用虚拟客户机的桥接一起工作。

警告

在使用桥接模式时，与客户机虚拟机一起使用的唯一绑定模式为模式 1、模式 2 和模式 4。使用模式 0、3、4、5 或 6 可能会导致连接失败。另请注意，Media-Independent Interface(MII)监控应该用来监控绑定模式，因为地址解析协议(ARP)监控无法正常工作。

有关绑定模式的更多信息，请参阅 相关知识库文章 或 Red Hat Enterprise Linux 7 网络指南。

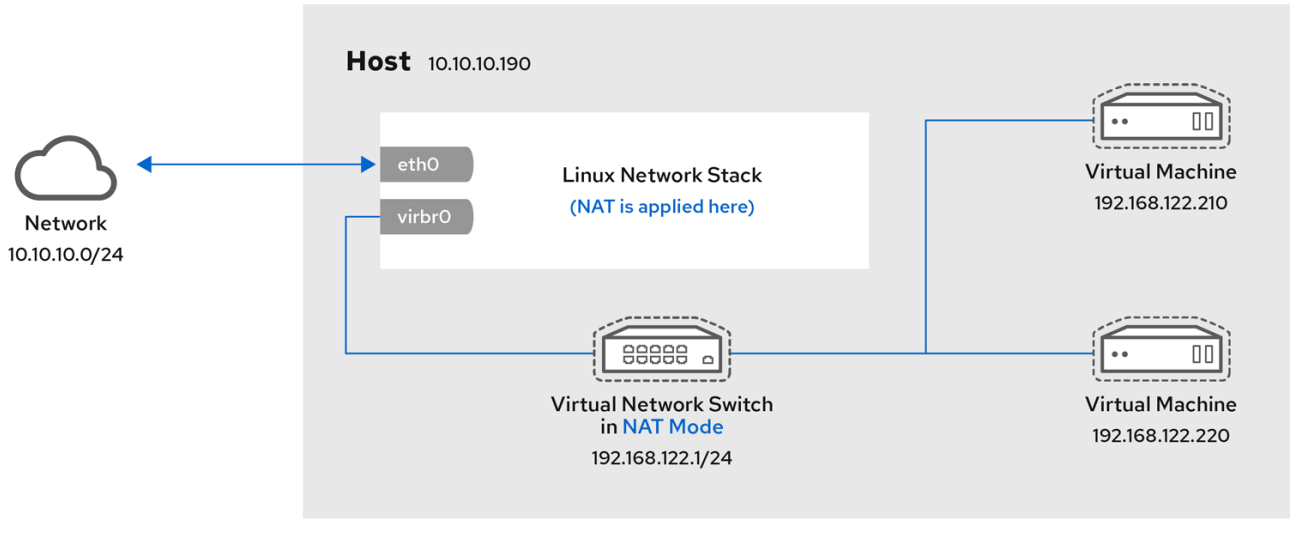
有关 bridge_opts 参数（用于配置桥接网络模式）的详情，请查看 Red Hat Virtualization 管理指南。

17.3. 网络地址转换

默认情况下，虚拟网络交换机在 NAT 模式下运行。它们使用 IP 伪装而不是 Source-NAT(SNAT)或

Destination-NAT(DNAT)。IP 伪装可让连接的客户机使用主机物理计算机 IP 地址与任何外部网络通信。默认情况下，当虚拟网络交换机在 NAT 模式运行时，外部放置到主机物理机器的计算机无法与客户机通信，如下图所示：

图 17.3. 使用两个客户机的 NAT 的虚拟网络交换机



警告

虚拟网络交换机使用 iptables 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无法通信。

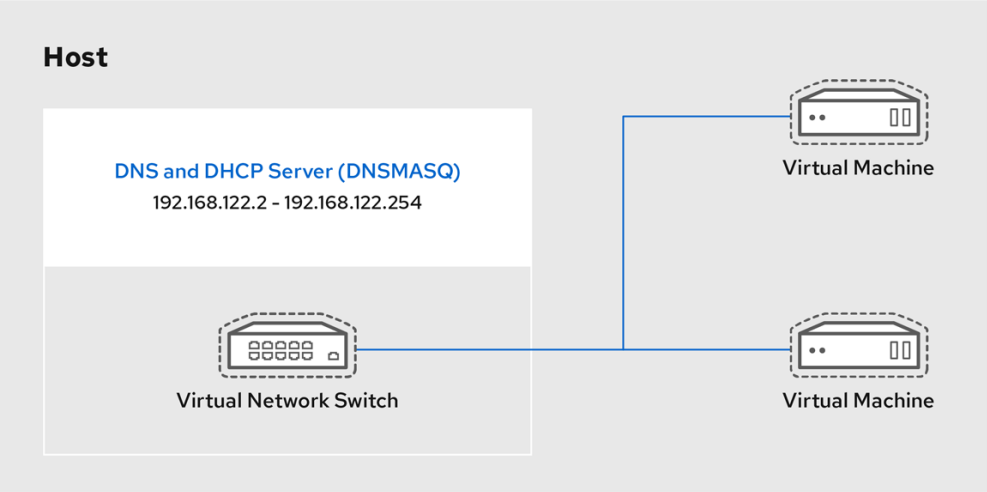
如果交换机没有运行，您可以为转发模式 NAT 设置公共 IP 范围，以便通过运行以下命令创建端口伪装范围：

```
# iptables -j SNAT --to-source [start]-[end]
```

17.4. DNS 和 DHCP

IP 信息可以通过 DHCP 分配给客户机。为此，可以将地址池分配到虚拟网络交换机。libvirt 使用 dnsmasq 程序。libvirt 会自动配置并启动需要它的每个虚拟网络交换机的 dnsmasq 实例。

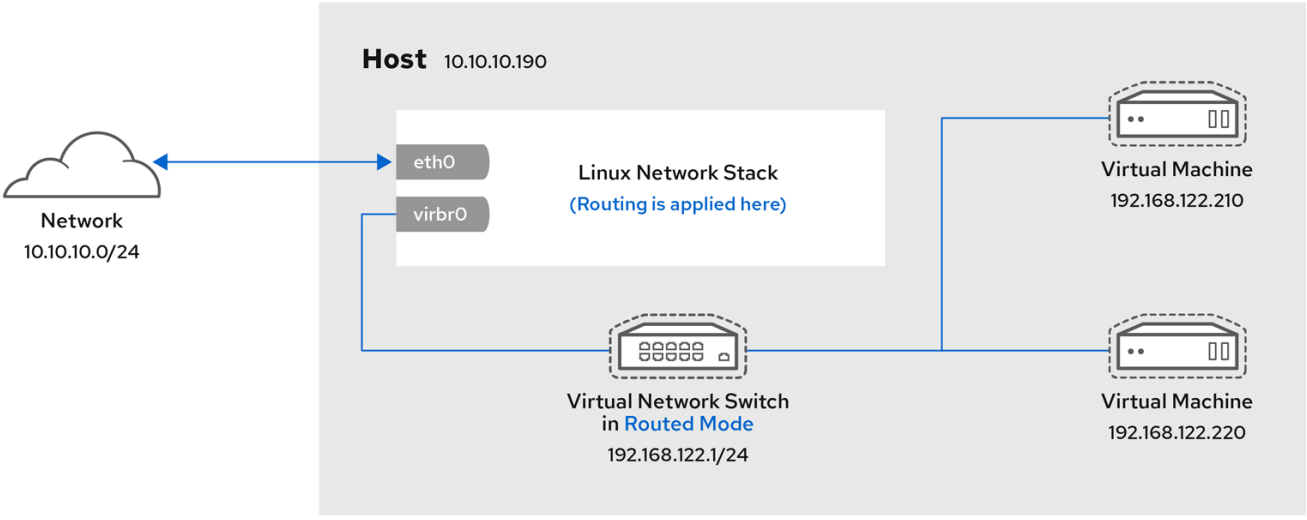
图 17.4. 运行 dnsmasq 的虚拟网络交换机



17.5. 路由模式

当使用 Routed 模式 时，虚拟交换机会连接到连接到主机物理机器的物理 LAN，并在不使用 NAT 的情况下传输流量。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，所有虚拟机都位于自己的子网中，通过虚拟交换机进行路由。这种情形并非始终理想选择，因为物理网络中的其他主机物理机器不知道虚拟机，无需手动物理路由器配置，也无法访问虚拟机。路由模式在 OSI 网络模型的第 3 层运行。

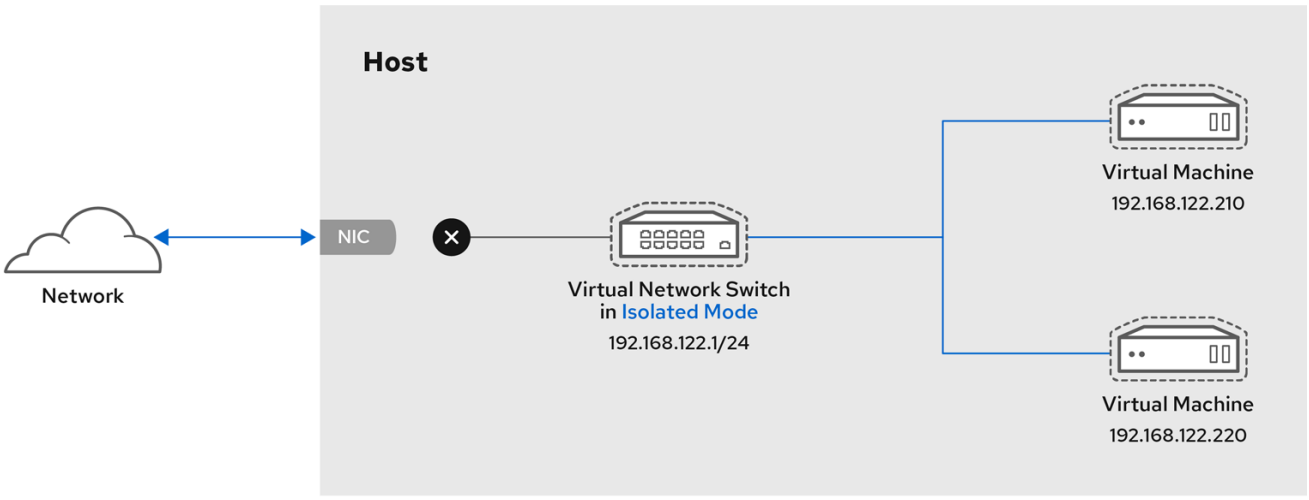
图 17.5. 使用路由模式的虚拟网络切换



17.6. 隔离模式

在使用 隔离模式 时，连接到虚拟交换机的客户机可以相互通信，并与主机物理机器通信，但其流量不会在主机物理机器外传递，而且它们无法从主机物理计算机外接收流量。在这个模式中使用 dnsmasq 是基本功能（如 DHCP）所必需的。但是，即使此网络与任何物理网络隔离，DNS 名称仍会解析。因此，当 DNS 名称解析但 ICMP 回显请求(ping)命令时可能会出现情况。

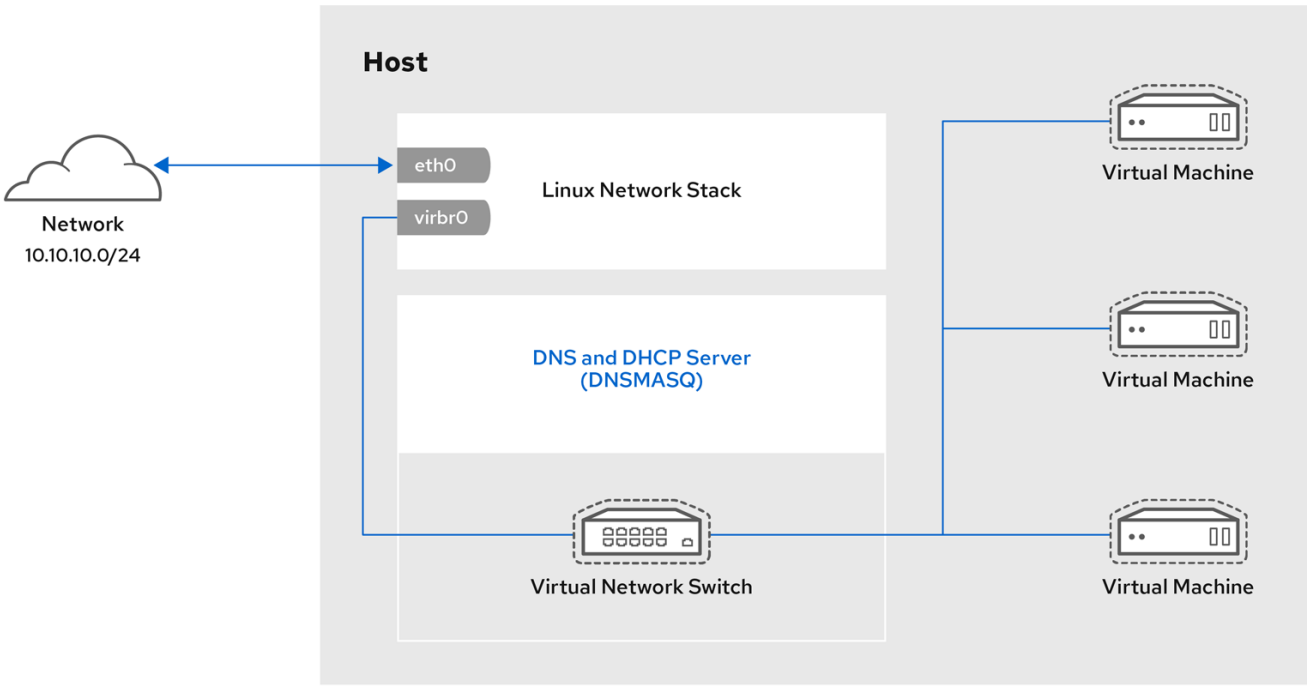
图 17.6. 处于隔离模式的虚拟网络交换机



17.7. 默认配置

当首先安装libvirtd守护进程时，它会在 NAT 模式中包含初始虚拟网络交换机配置。使用此配置以便已安装的客户机可以通过主机物理计算机与外部网络通信。下图展示了 libvirtd 的以下默认配置：

图 17.7. 默认 libvirt 网络配置



注意

虚拟网络可以限制为特定的物理接口。这对具有多个接口的物理系统很有用（如 eth 0、eth1 和 eth2）。这仅适用于路由和 NAT 模式，可以在 dev=<interface> 选项中定义，或者在创建新虚拟网络时在 virt-manager 中定义。

17.8. COMMON SCENARIOS 示例

本节演示了不同的虚拟网络模式，并提供一些示例方案。

17.8.1. 网桥模式

网桥模式在 OSI 模型的第 2 层上运行。在使用时，所有 guest 虚拟机都将与主机物理机器位于同一子网中。使用桥接模式的最常见用例包括：

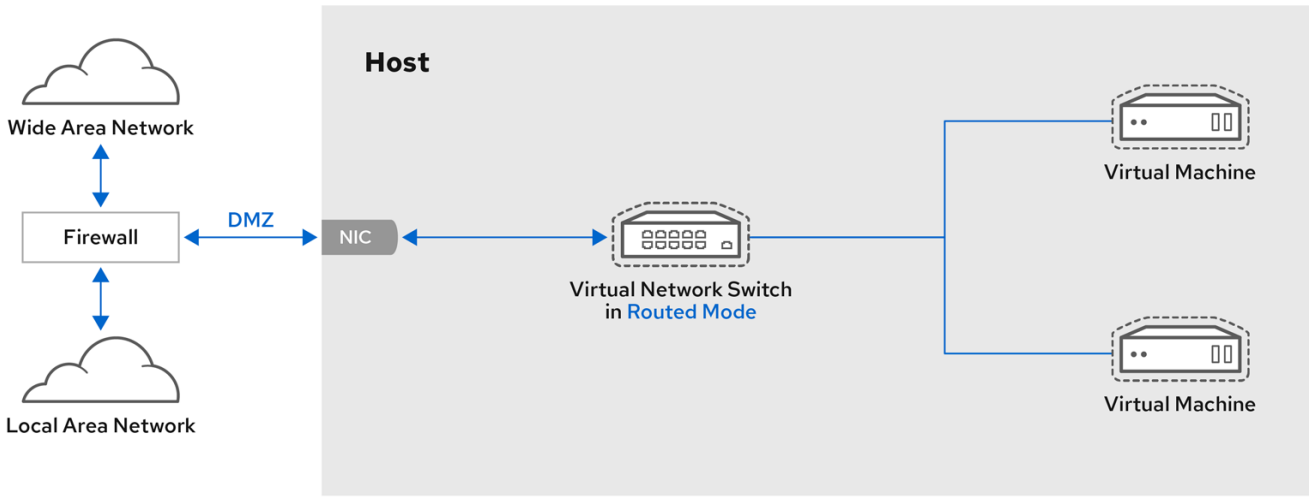
- 在现有网络中部署客户机虚拟机以及主机物理计算机对最终用户而言，虚拟机与物理机器之间的差别是透明的。
- 在不更改现有物理网络配置设置的情况下部署客户机虚拟机。
- 部署客户端虚拟机，这些虚拟机必须可以被现有物理网络轻松访问。将客户机虚拟机放置到物理网络中，以便它们必须在现有广播域中访问服务，如 DHCP。
- 将客户机虚拟机连接到使用 VLAN 的网络.

17.8.2. 路由模式

DMZ

考虑一个网络，因为安全原因，将一个或多个节点放置在受控的子网络中。特殊的子网络部署，如这一个常见做法，而子网络则称为 DMZ。有关这个布局的详情，请参见以下图：

图 17.8. DMZ 配置示例

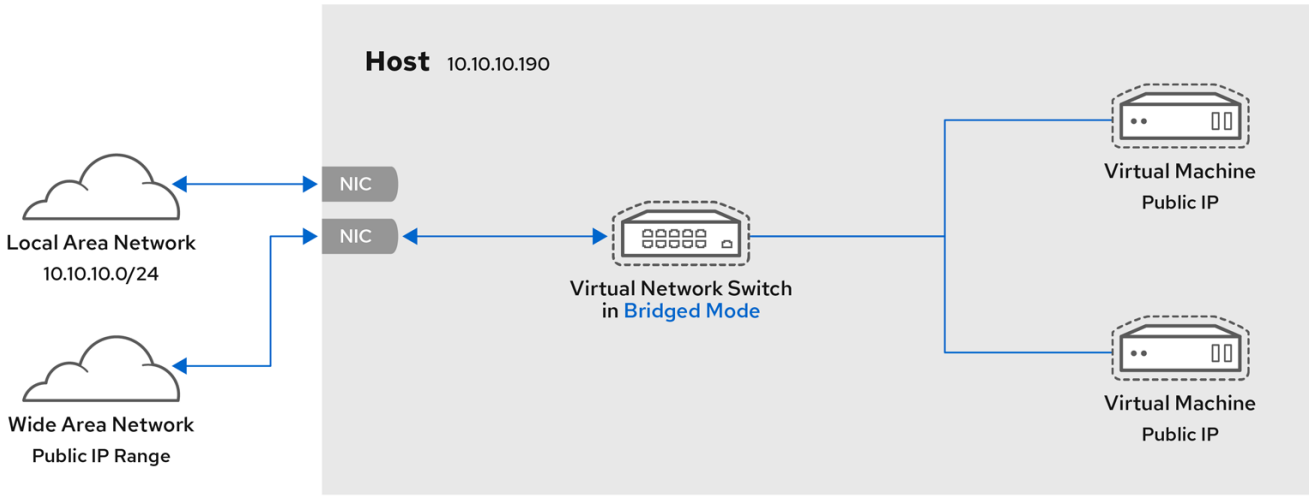


DMZ 中的主机物理机器通常向 WAN（外部）主机物理机器以及 LAN（内部）主机物理机器提供服务。因此，这需要它们能从多个位置访问，并考虑这些位置可根据其安全性和信任级别的不同方式进行控制和操作，路由模式是此环境的最佳配置。

虚拟服务器托管

考虑一台含有多台主机物理计算机的虚拟服务器托管公司，每个机器都有两个物理网络连接。一个接口用于管理和核算，另一个用于虚拟机进行连接。每个 guest 都有自己的公共 IP 地址，但主机物理计算机使用私有 IP 地址作为 guest 管理，只有内部管理员才能执行。参阅下图以了解这种情况：

图 17.9. 托管配置示例的虚拟服务器



17.8.3. NAT 模式

NAT（网络地址转换）模式是默认模式。当不需要直接查看网络时，可以使用它进行测试。

17.8.4. 隔离模式

隔离模式允许虚拟机相互通信。它们无法与物理网络交互。

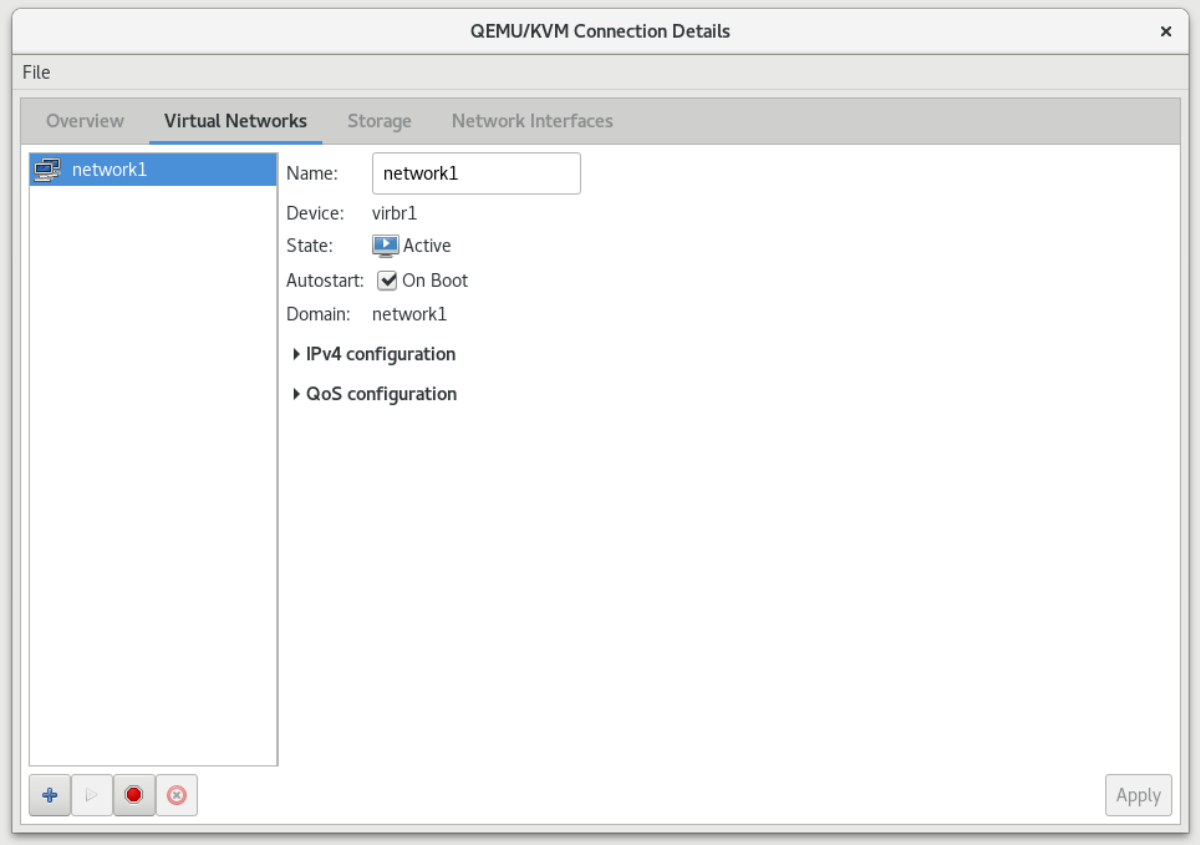
17.9. 管理虚拟网络

在您的系统中配置虚拟网络：

1. 在 Edit 菜单中，选择 Connection Details。

2. 这将打开 Connection Details 菜单。单击 Virtual Networks 选项卡。

图 17.10. 虚拟网络配置



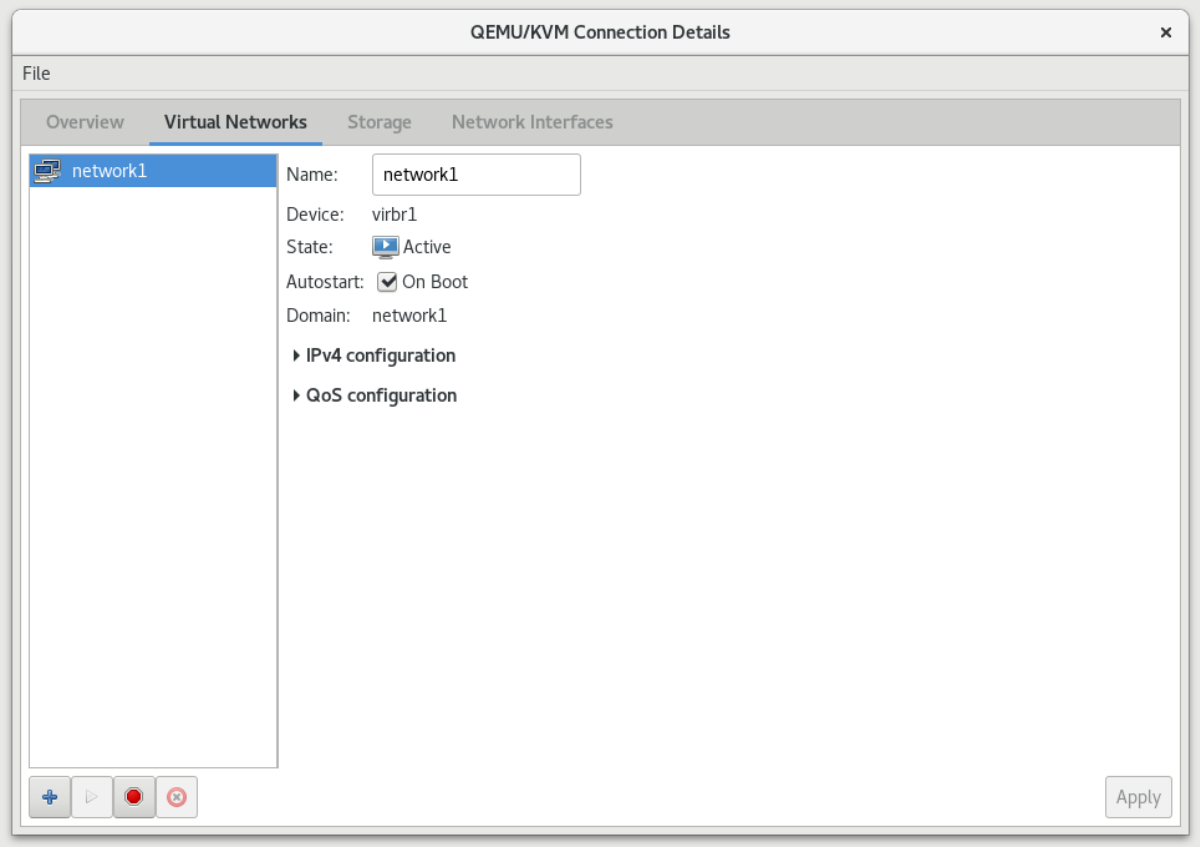
3. 菜单左侧列出了所有可用的虚拟网络。您可编辑虚拟网络的配置，方法是从此框中选择它，并在您看到合适的时进行编辑。

17.10. 创建虚拟网络

使用虚拟机管理器(virt-manager)在系统上创建虚拟网络：

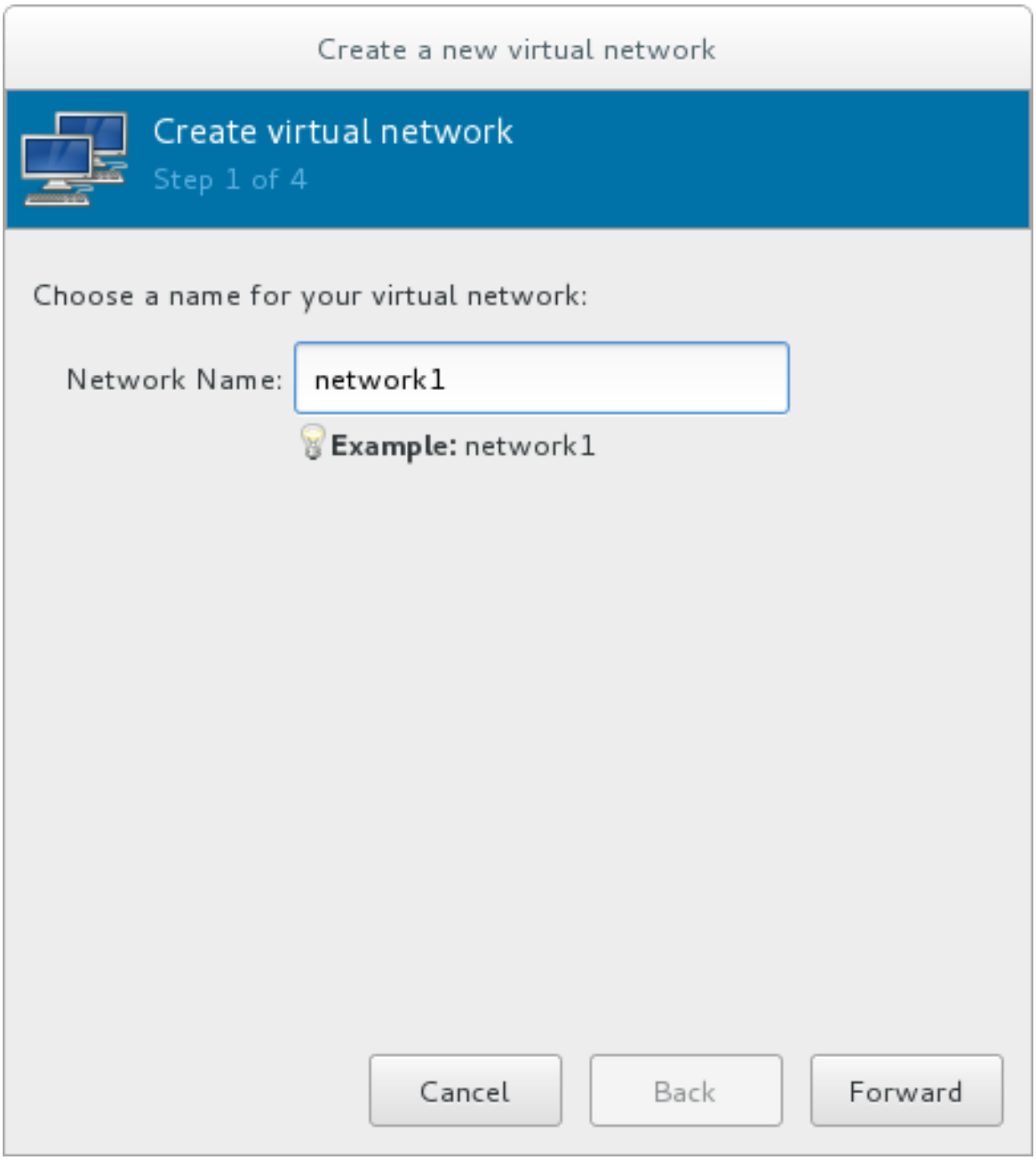
1. *从" 连接详细信息 "菜单内打开 虚拟网络 选项卡。单击添加网络 按钮，由加号(+)图标标识。如需更多信息，请参阅 第 17.9 节 “管理虚拟网络”。*

图 17.11. 虚拟网络配置



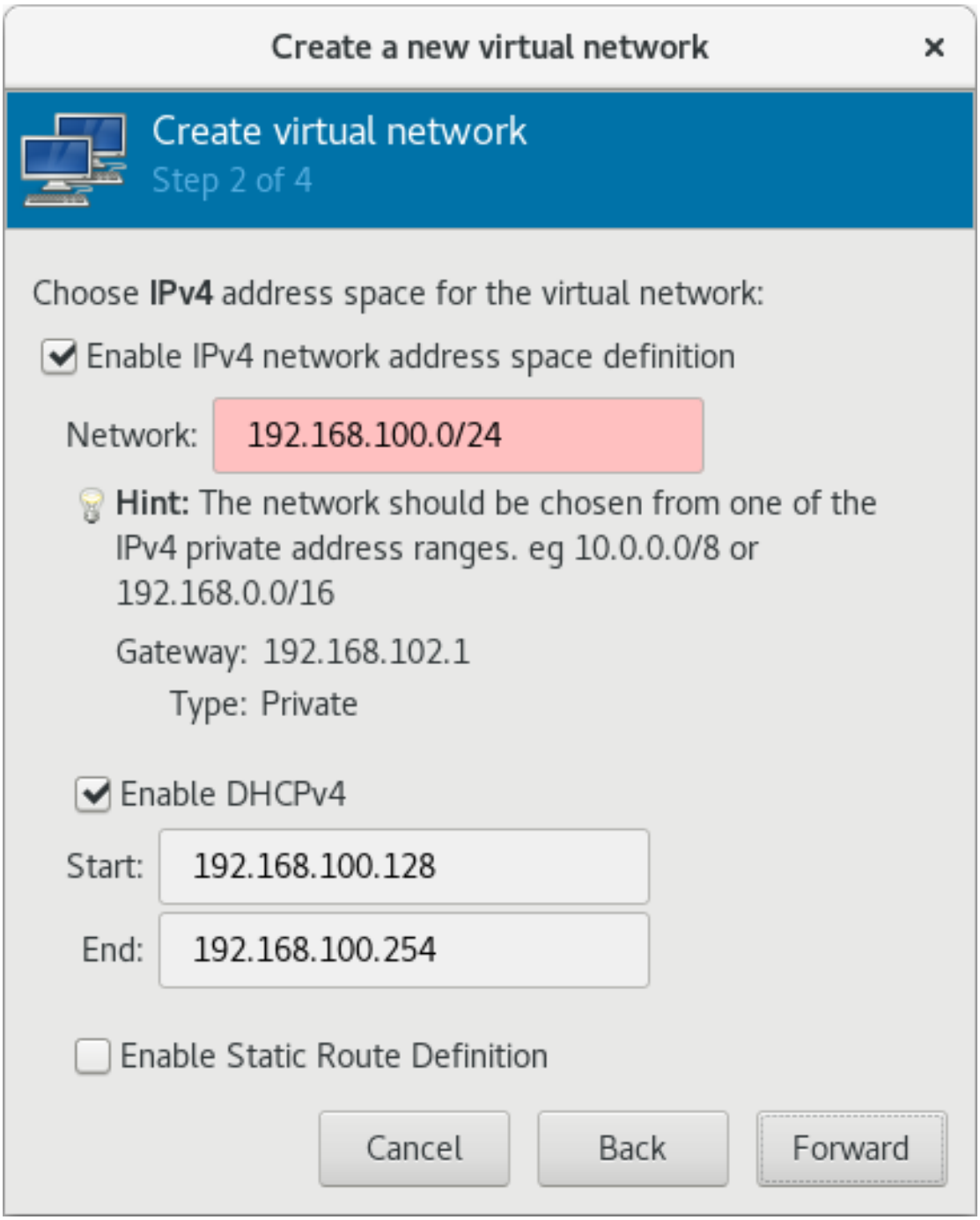
这将打开 Create a new virtual network 窗口。点 Forward 继续。

图 17.12. 命名您的新虚拟网络



2. *为您的虚拟网络输入适当的名称，然后单击" 下一步"。*

图 17.13. 选择 IPv4 地址空间



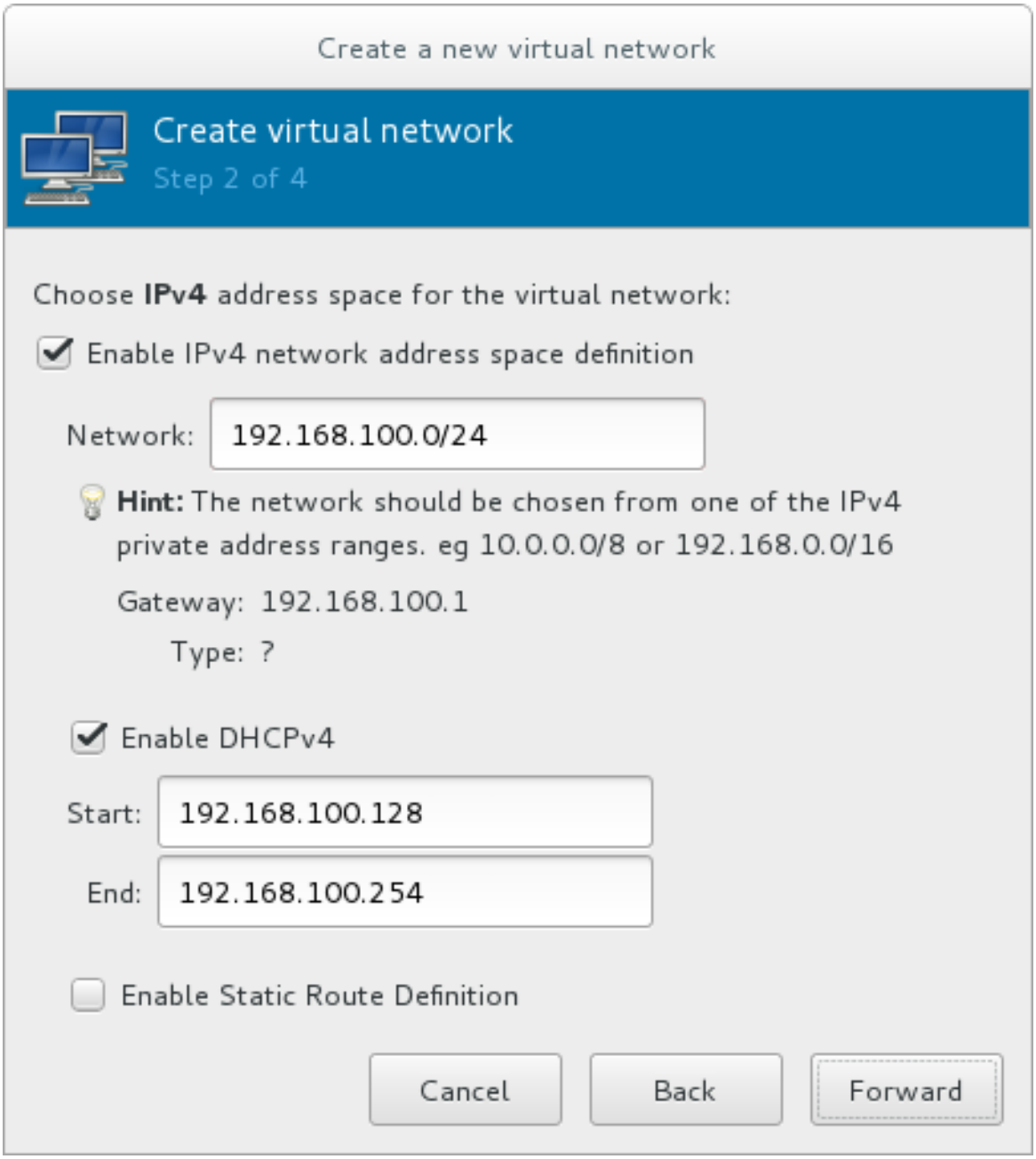
3. *选中启用 IPv4 网络地址空间定义 复选框。*

   在 Network 字段中输入您的虚拟网络的 IPv4 地址空间。

   选中 Enable DHCPv4 复选框。

   通过指定起始和 结束 IP 地址范围来定义您的虚拟网络的 DHCP 范围。

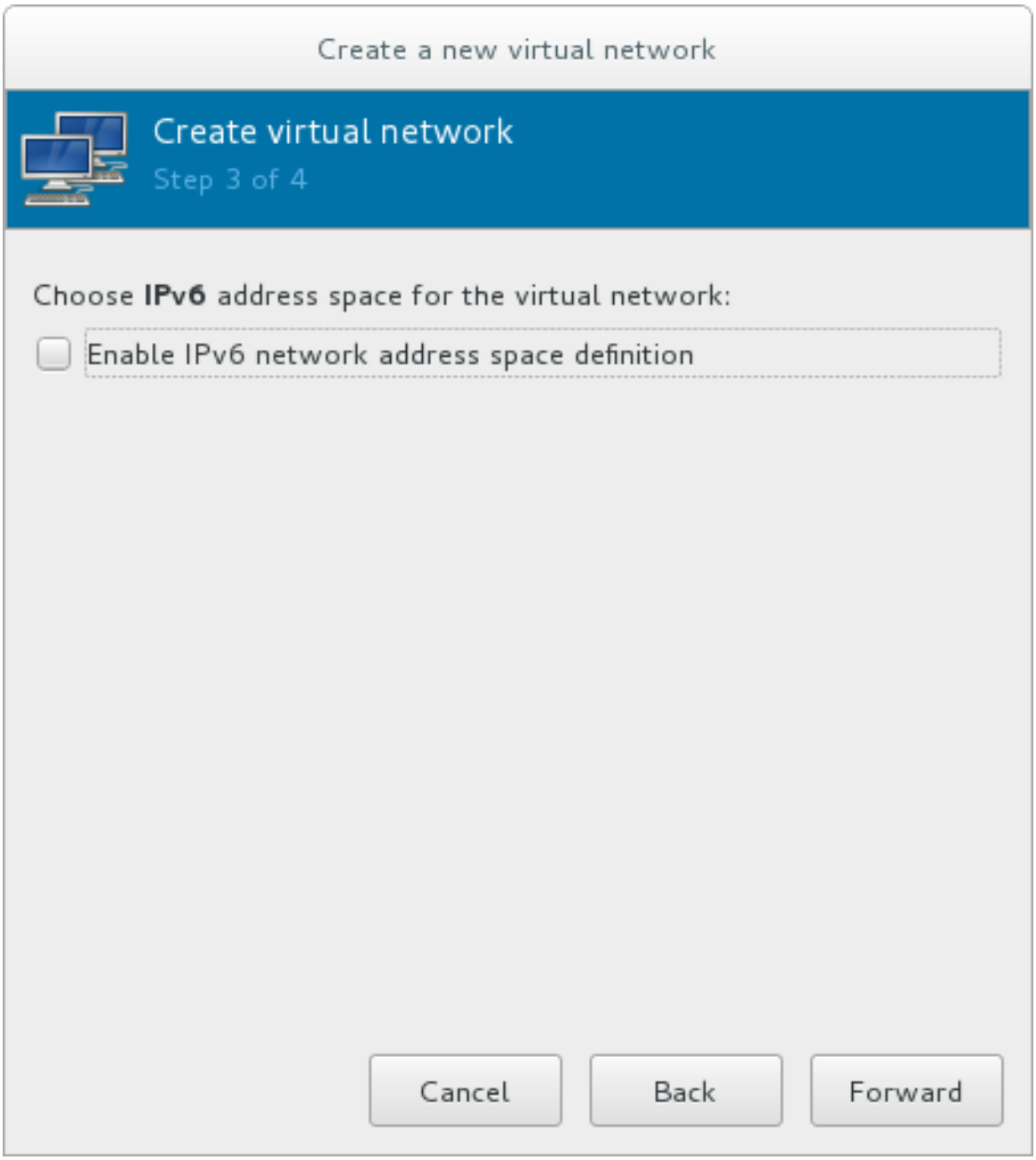
图 17.14. 选择 IPv4 地址空间



点 Forward 继续。

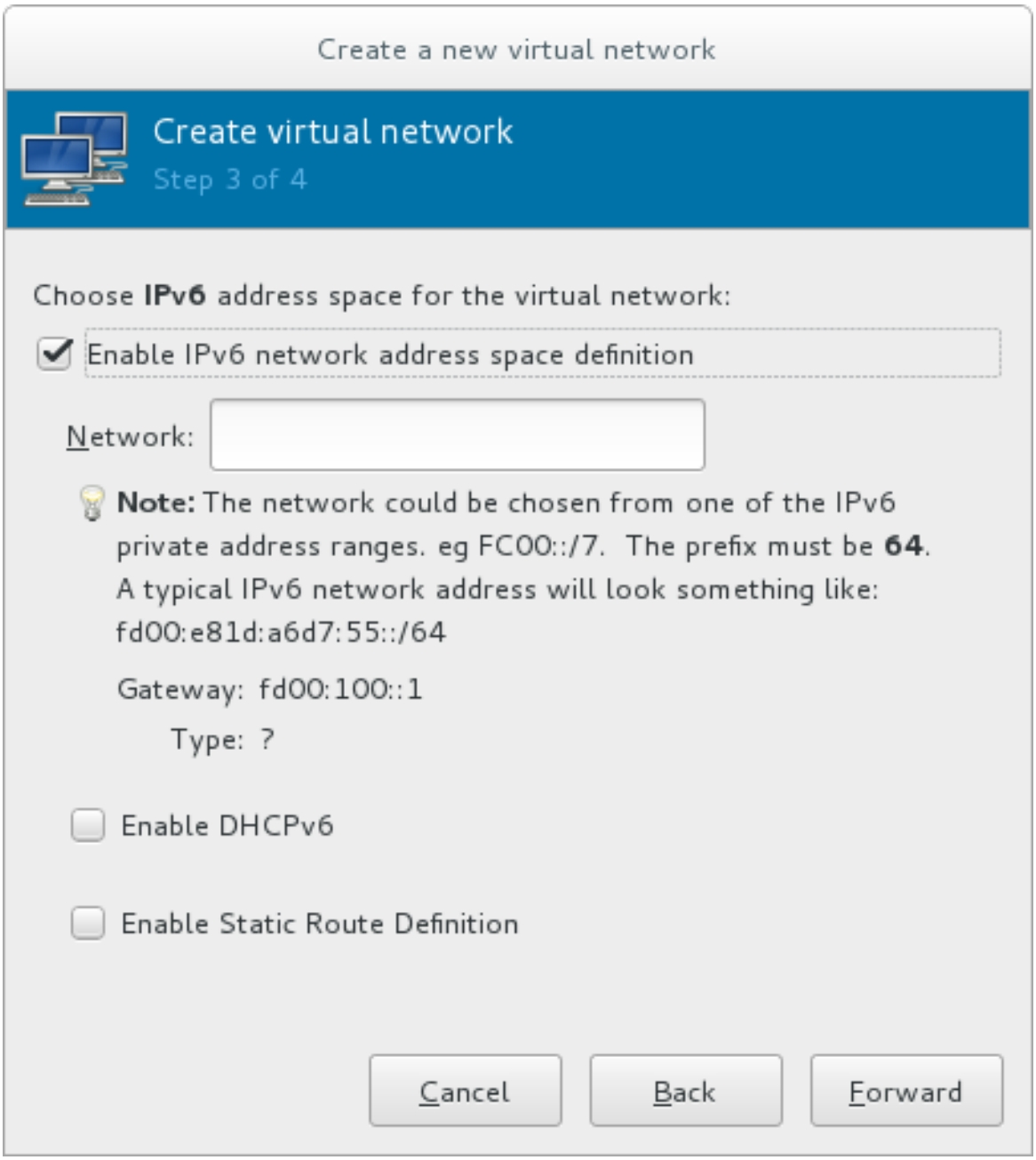
4. *如果要启用 IPv6，请检查 启用 IPv6 网络地址空间定义。*

图 17.15. 启用 IPv6



其他字段会出现在 Create a new virtual network 窗口中。

图 17.16. 配置 IPv6

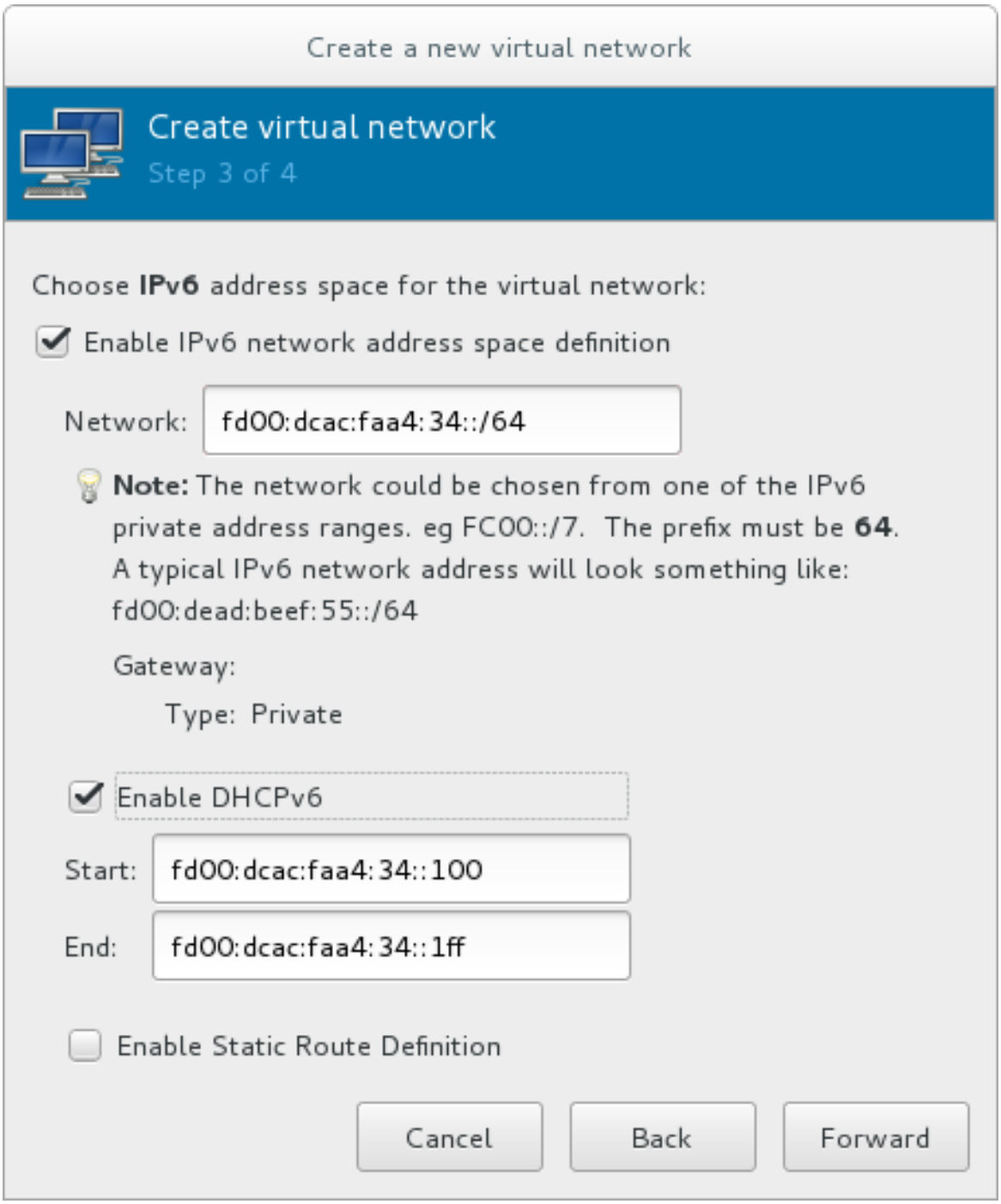


在 Network 字段中输入 IPv6 地址。

5. *如果要启用 DHCPv6，请选中 Enable DHCPv6 复选框。*

其他字段会出现在 Create a new virtual network 窗口中。

图 17.17. 配置 DHCPv6

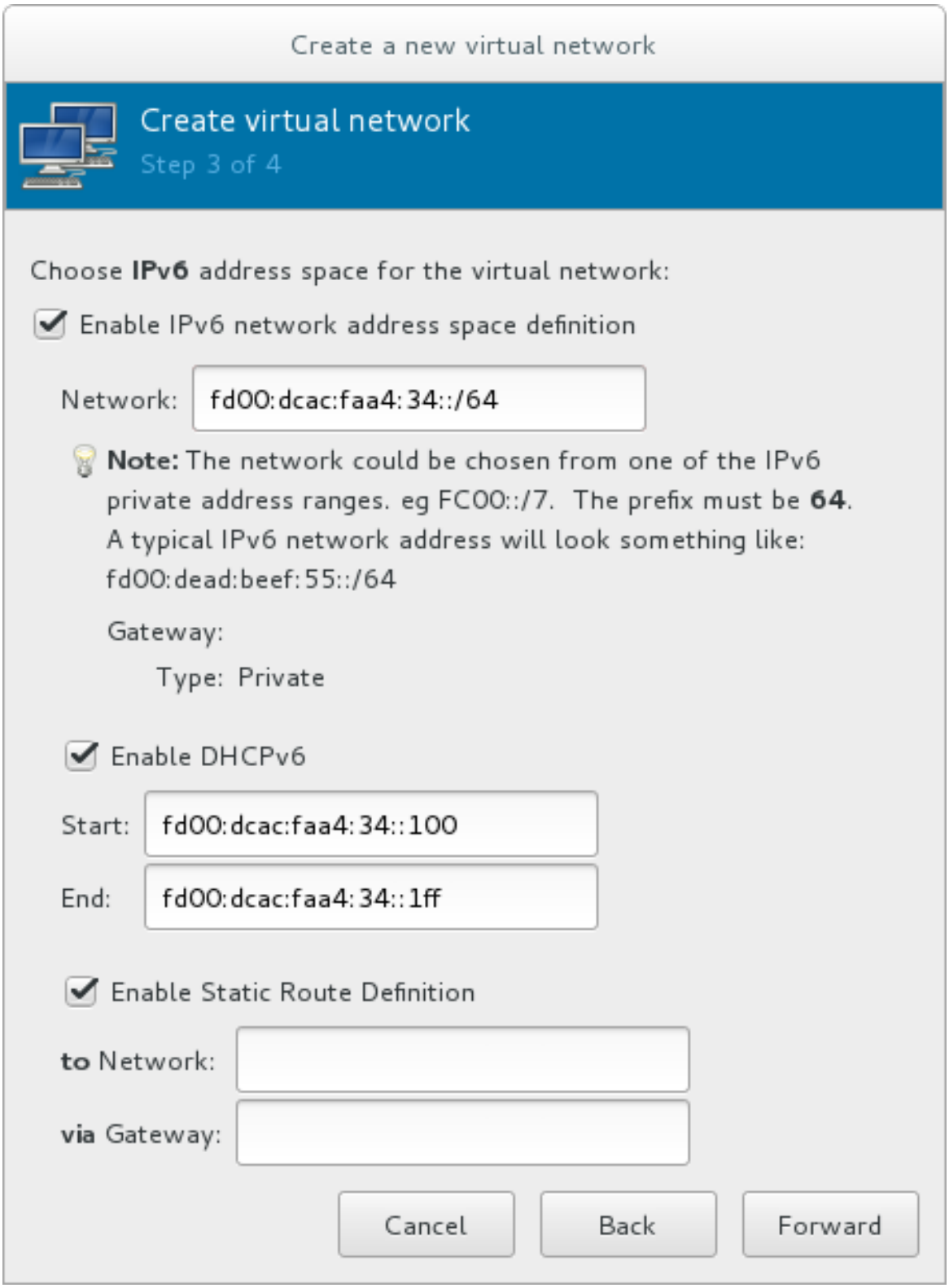


（可选）编辑 DHCPv6 范围的开头和结尾。

6. *如果要启用静态路由定义，请选中 Enable Static Route Definition 复选框。*

   其他字段会出现在 Create a new virtual network 窗口中。

图 17.18. 定义静态路由



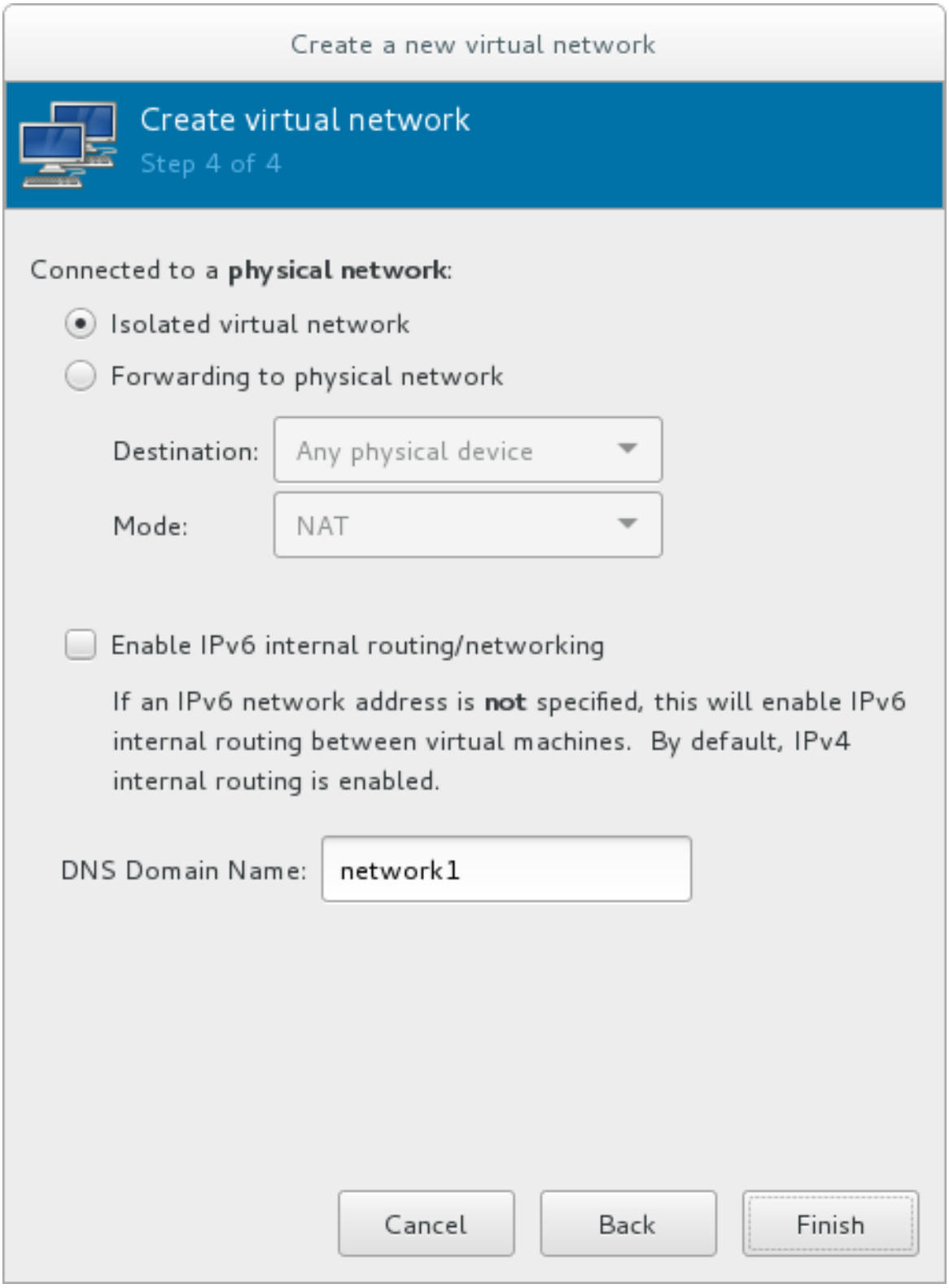
在相应字段中输入网络地址和网关，该网关将用于该网络的路由。

点 Forward。

7.

选择虚拟网络应如何连接到物理网络。

图 17.19. 连接到物理网络



如果您希望隔离虚拟网络，请确保选择了 Isolated 虚拟网络 单选按钮。

如果您希望虚拟网络连接到物理网络，请选择"转发到物理网络"， 并选择" 目标 "应为 任何

物理设备 还是特定物理设备。另外，选择 Mode 应为 NAT 还是 Routed。

如果要在虚拟网络中启用 IPv6 路由，请选中 Enable IPv6 内部路由/ 网络复选框。

输入虚拟网络的 DNS 域名。

单击 Finish 以创建该虚拟网络。

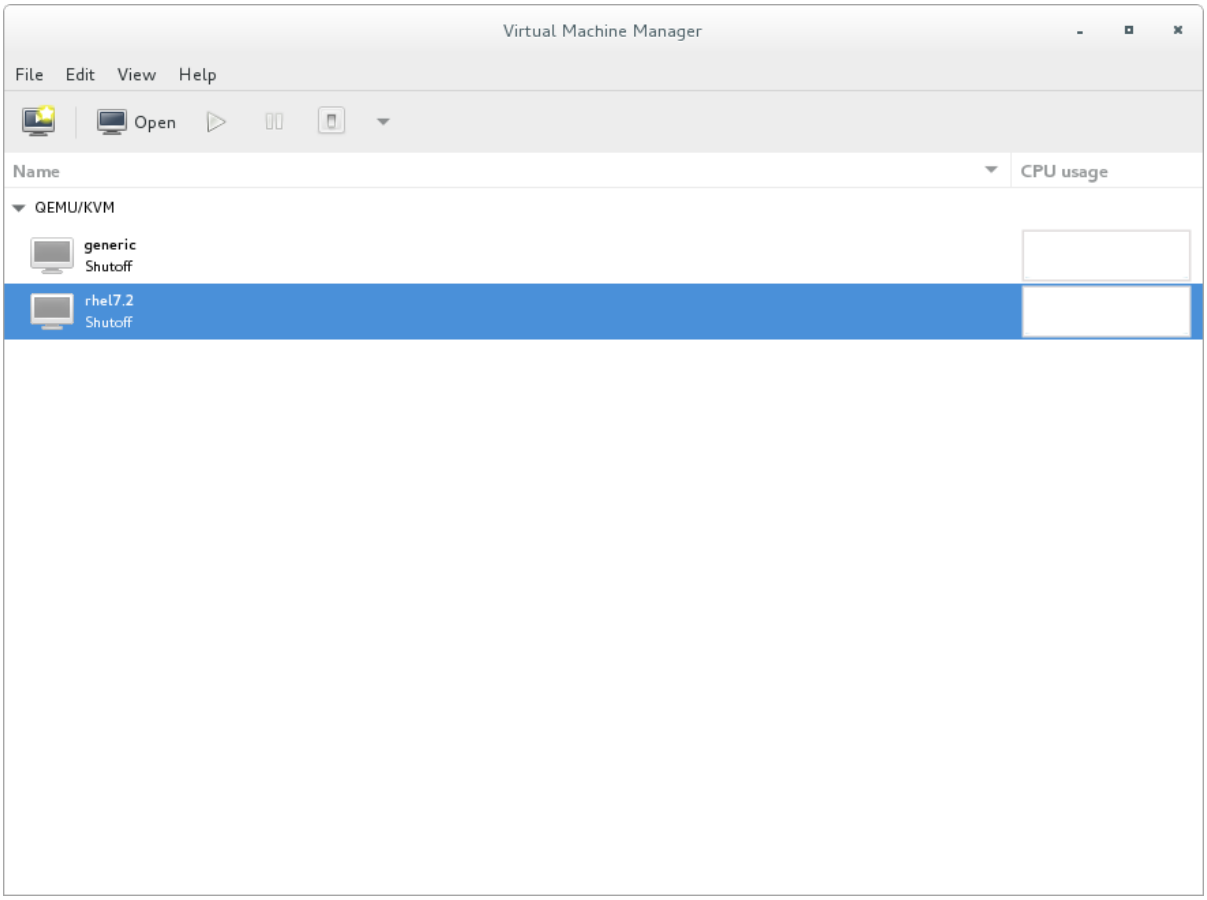
8. 新的虚拟网络现在包括在 Connection Details 窗口的 Virtual Networks 选项卡中。

17.11. 将虚拟网络附加到客户机

将虚拟网络附加到客户端：

1. 在 Virtual Machine Manager 窗口中，突出显示分配了该网络的客户机。

图 17.20. 选择要显示的虚拟机

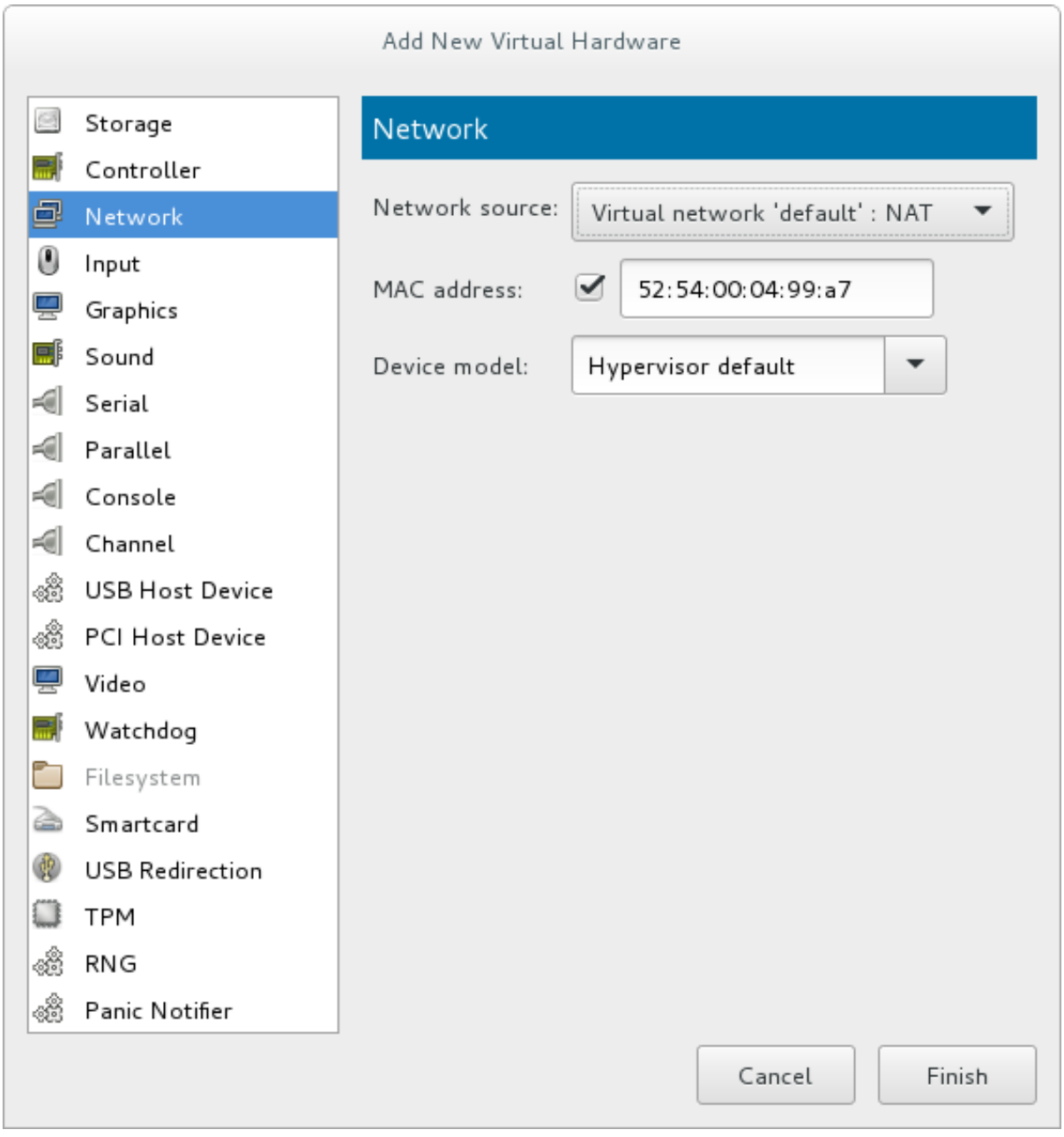


2. 在 Virtual Machine Manager Edit 菜单中，选择 Virtual Machine Details。

3. 点击 Virtual Machine Details 窗口中的 Add Hardware 按钮。

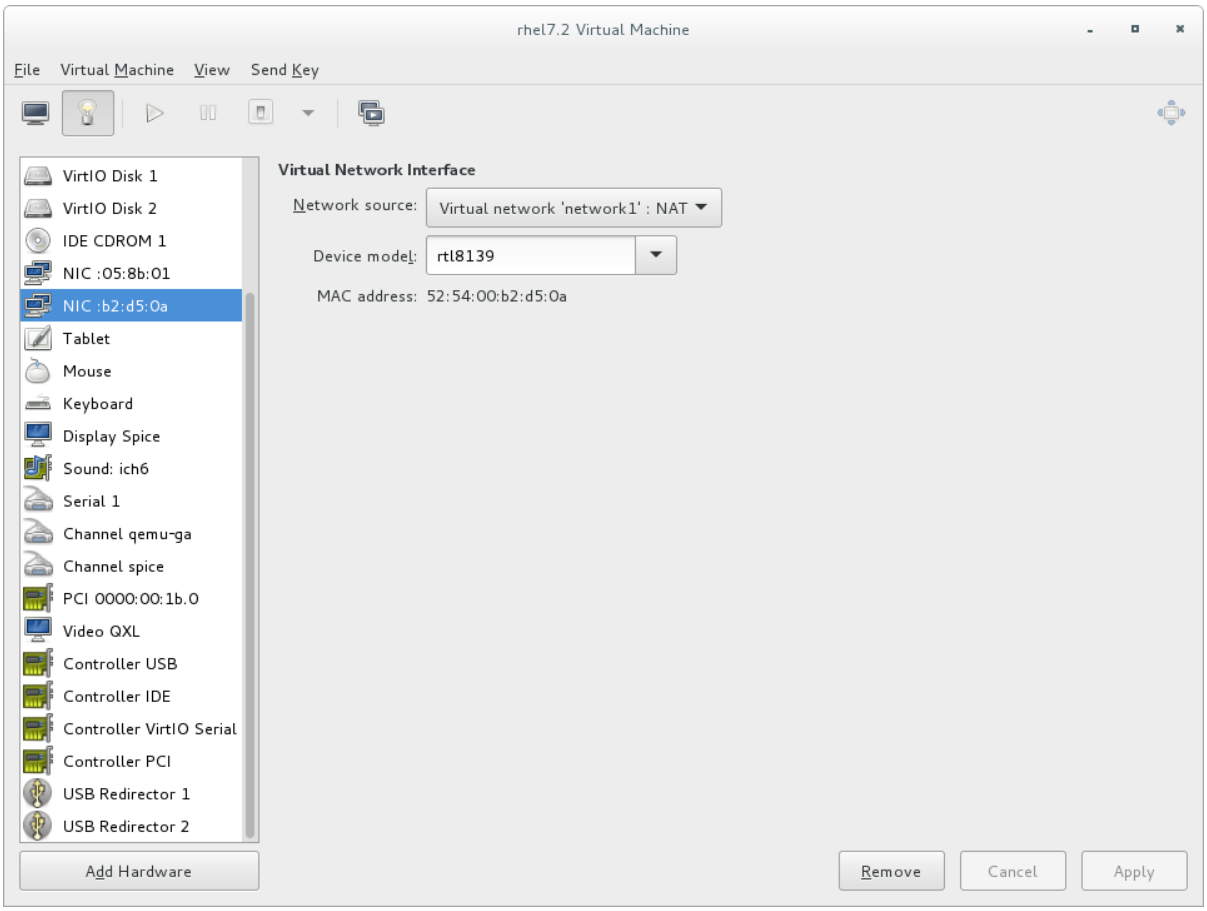
4. 在 Add new virtual hardware 窗口中，从左侧窗格中选择 Network，然后从网络 源 菜单中选择您的网络名称（本例中为network1 ）。修改 MAC 地址（如有必要），然后选择 设备模型。点 Finish。

图 17.21. 从 Add new virtual hardware 窗口选择您的网络



5. 新网络现在显示为一个虚拟网络接口，它将在启动时提供给 guest。

图 17.22. 在客户端硬件列表中显示的新网络



17.12. 直接将虚拟 NIC 附加到物理接口

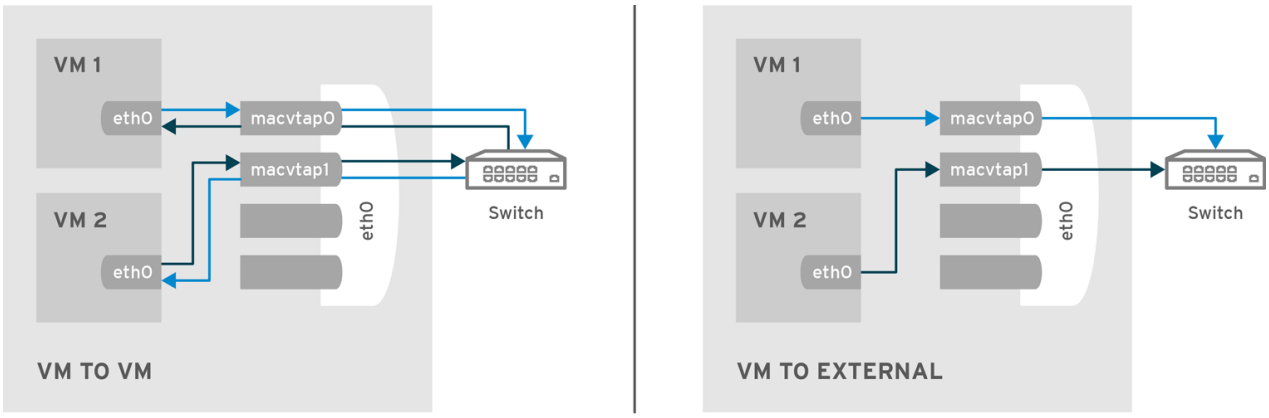
作为默认 NAT 连接的替代选择，您可以使用 macvtap 驱动程序将 guest 的 NIC 直接附加到主机的指定物理接口。这不会与 设备分配 混淆（也称为 passthrough）。macvtap 连接有以下模式，每个模式具有不同的优点和使用案例：

物理接口交付模式

VEPA

在虚拟以太网端口聚合器(VEPA)模式中，所有来自客户机的数据包都发送到外部交换机。这可让用户通过交换机强制客户机流量。要使 VEPA 模式正常工作，外部交换机还必须支持 hairpin 模式，这样可确保其目的地为同一主机计算机上的虚拟客户机的数据包由外部交换机发回到主机。

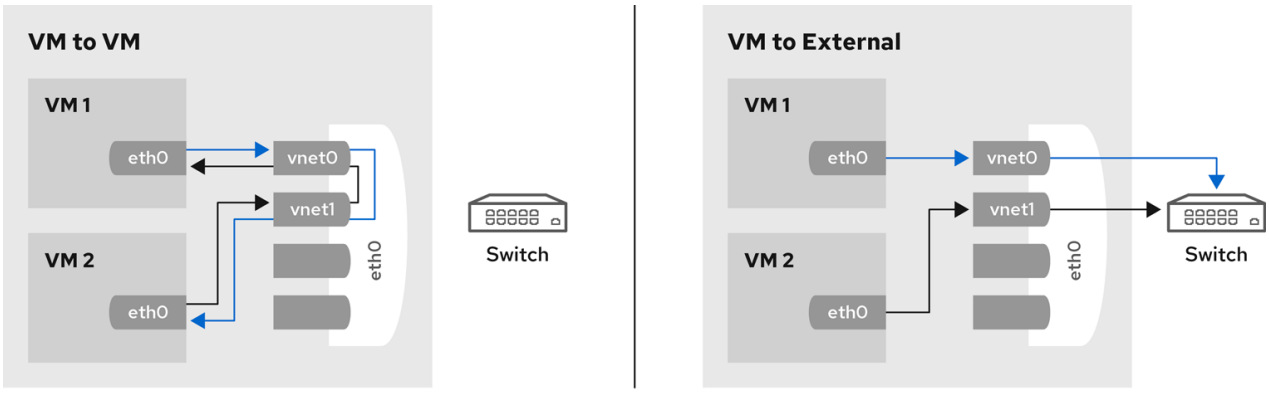
图 17.23. VEPA 模式



bridge

其目的地位于与其源客户机相同的主机中，直接传送到目标 macvtap 设备。源设备和目标设备都需要处于网桥模式，才能成功发送。如果其中任何一个设备处于 VEPA 模式，则需要支持 hairpin 的外部交换机。

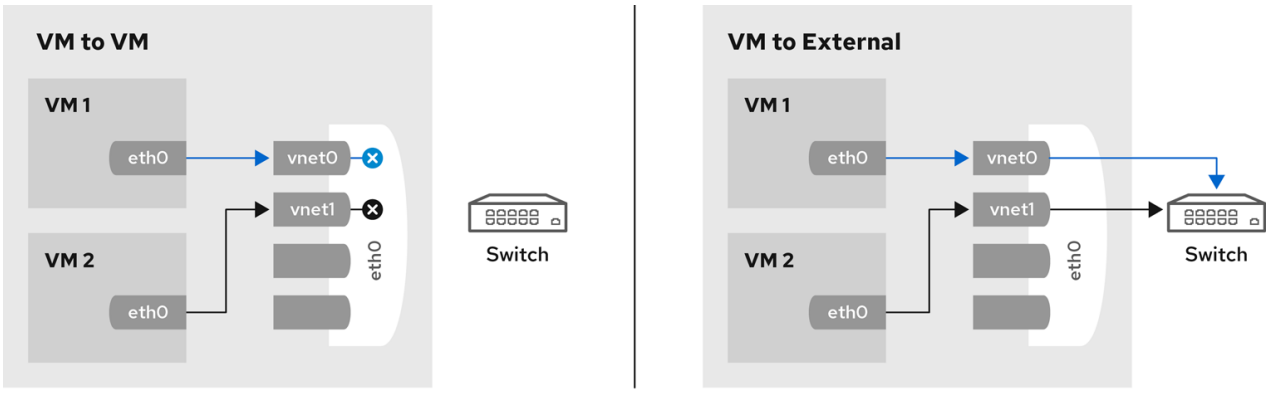
图 17.24. 网桥模式



私有

所有数据包都发送到外部交换机，并且仅发送到同一主机计算机上的目标 guest（如果通过外部路由器或网关发送它们），并将这些数据包发回到主机。私有模式可用于防止单一主机上的各个客户机相互通信。如果源或者目标设备处于私有模式，则会显示这个过程。

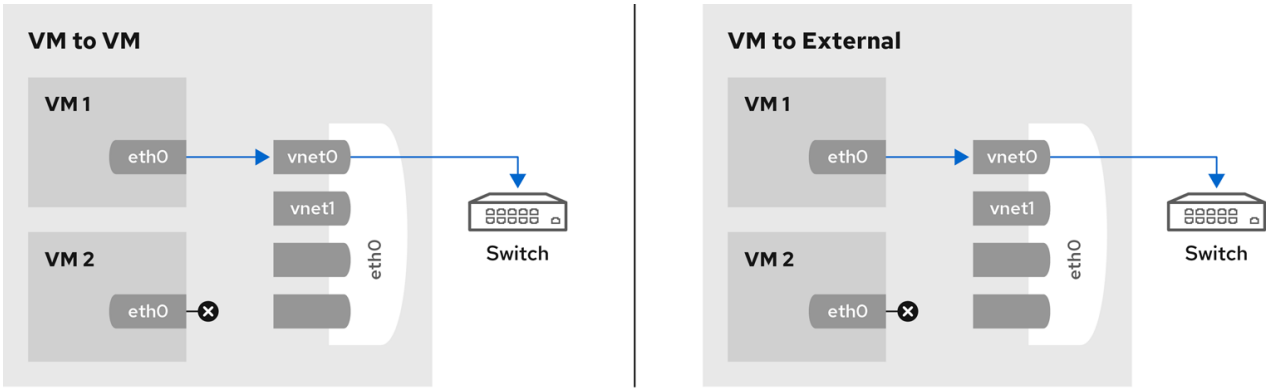
图 17.25. 私有模式



passthrough

此功能直接向客户机附加物理接口设备或 SR-IOV 虚拟功能(VF)，而不会丢失迁移功能。所有数据包都直接发送到指定的网络设备。请注意，单个网络设备只能传递给单个虚拟机，因为无法在以 passthrough 模式的客户机间共享网络设备。

图 17.26. Passthrough 模式



可以通过更改域 XML 文件或使用 virt-manager 接口来配置 macvtap。

17.12.1. 使用域 XML 配置 macvtap

打开客户机的域 XML 文件，并按如下所示修改 <devices> 元素：

```
<devices>
 ...
 <interface type='direct'>
  <source dev='eth0' mode='vepa'/>
 </interface>
</devices>
```

直接连接的客户机虚拟机的网络访问可由硬件交换机管理，其中的硬件交换机可以连接到主机物理机器的物理接口。

如果交换机符合 IEEE 802.1Qbg 标准，接口可以具有如下所示的附加参数。virtualport 元素的参数在 IEEE 802.1Qbg 标准中详细记录。值是特定于网络，应当由网络管理员提供。在 802.1Qbg 术语中，虚拟工作站接口(VSI)代表虚拟机的虚拟接口。另请注意，IEEE 802.1Qbg 需要 VLAN ID 的非零值。

虚拟工作站接口类型

managerid

VSI Manager ID 标识包含 VSI 类型和实例定义的数据库。这是一个整数值，值 0 为保留。

typeid

VSI 类型 ID 标识了 VSI 类型字符化网络访问。VSI 类型通常由网络管理员管理。这是一个整数值。

typeidversion

VSI Type Version 允许多个 VSI 类型版本。这是一个整数值。

InstanceID

当创建 VSI 实例（虚拟机的虚拟接口）时，将生成 VSI 实例 ID。这是一个全局唯一标识符。

profileid

配置集 ID 包含要应用到此接口的端口配置文件的名称。此名称由 port profile 数据库解析为网络参数，这些网络参数将应用到这个接口。

通过更改域 XML 文件来配置四种类型中的每个类型。打开该文件后，更改模式设置，如下所示：

```
<devices>
 ...
 <interface type='direct'>
  <source dev='eth0.2' mode='vepa'/>
   <virtualport type="802.1Qbg">
    <parameters managerid="11" typeid="1193047" typeidversion="2" instanceid="09b11c53-8b5c-
4eeb-8f00-d84eaa0aaa4f"/>
```

```
   </virtualport>
  </interface>
</devices>
```

配置集 ID 如下所示：

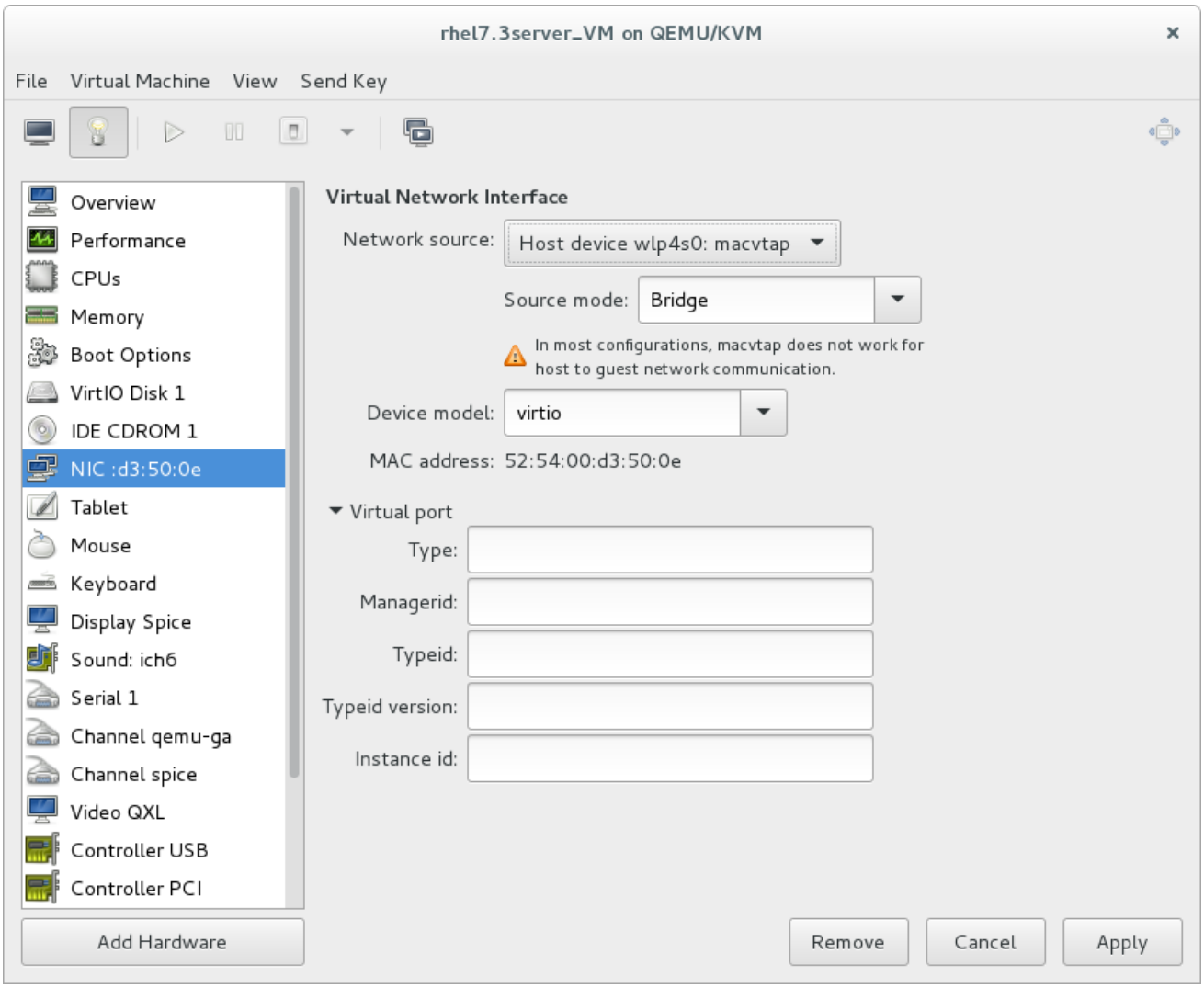
```
<devices>
 ...
 <interface type='direct'>
  <source dev='eth0' mode='private'/>
   <virtualport type='802.1Qbh'>
    <parameters profileid='finance'/>
   </virtualport>
 </interface>
</devices>
...
```

17.12.2. 使用 virt-manager 配置 macvtap

在 Network 源菜单 containerruntime 选择 NIC，选择主机 设备名称: macvtap containerruntime 选择预期的源 模式，打开 虚拟硬件详情窗口 containerruntime 选择预期的源 模式。

然后可在虚拟 端口子菜单中设置虚拟 站接口类型。

图 17.27. 在 virt-manager 中配置 macvtap



17.13. 动态更改附加到虚拟 NIC 的主机物理机器或网络桥接

本节演示如何在不影响客户机虚拟机的情况下将客户机虚拟机的 vNIC 从一个网桥移动到另一个桥接。

1. *使用类似如下的配置准备客户端虚拟机：*

```
<interface type='bridge'>
    <mac address='52:54:00:4a:c9:5e'/>
    <source bridge='virbr0'/>
    <model type='virtio'/>
</interface>
```

2. *为接口更新准备 XML 文件：*

```
# cat br1.xml
```

```
<interface type='bridge'>
    <mac address='52:54:00:4a:c9:5e'/>
    <source bridge='virbr1'/>
    <model type='virtio'/>
</interface>
```

3. 启动 guest 虚拟机，确认 guest 虚拟机的网络功能，并检查 guest 虚拟机的 vnetX 是否已连接到您指定的网桥。

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
virbr0          8000.5254007da9f2       yes             virbr0-nic

vnet0
virbr1          8000.525400682996       yes             virbr1-nic
```

4. 使用以下命令使用新接口参数更新客户机虚拟机的网络：

```
# virsh update-device test1 br1.xml

Device updated successfully
```

5. 在 guest 虚拟机上，运行 服务网络重新启动。客户机虚拟机获取 virbr1 的新 IP 地址。检查 guest 虚拟机的 vnet0 是否已连接到新网桥(virbr1)

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
virbr0          8000.5254007da9f2       yes             virbr0-nic
virbr1          8000.525400682996       yes             virbr1-nic      vnet0
```

17.14. 应用网络过滤

本节介绍 libvirt 的网络过滤器、目标、概念和 XML 格式。

17.14.1. 简介

网络配置的目标是使虚拟化系统的管理员能够对虚拟机配置和强制实施网络流量过滤规则，并且管理虚拟机可以发送或接收的网络流量的参数。当虚拟机启动时，网络流量过滤规则会在主机物理机器上应用。由于过滤规则不能从虚拟机内部绕过，因此从虚拟机用户的角度看来就是必须的。

从客户机虚拟机的角度来看，网络过滤系统允许每个虚拟机的网络流量过滤规则，每个接口单独配

置。这些规则在虚拟机启动时应用到主机物理机器上，并可在虚拟机运行时对其进行修改。可以通过修改网络过滤器的 XML 描述来实现后一种。

多个虚拟机可以使用相同的通用网络过滤器。当修改了这个过滤器时，会更新引用此过滤器的所有正在运行的虚拟机的网络流量过滤规则。未运行的机器将在启动时更新。

如前文所述，可以在为特定类型的网络配置而配置的独立网络接口上实施应用网络流量过滤规则。支持的网络类型包括：

- network
- Ethernet -- 必须在桥接模式中使用
- bridge

例 17.1. 网络过滤示例

接口 XML 用于引用顶级过滤器。在以下示例中，接口描述引用过滤器 clean-traffic。

```
<devices>
 <interface type='bridge'>
   <mac address='00:16:3e:5d:c7:9e'/>
   <filterref filter='clean-traffic'/>
 </interface>
</devices>
```

网络过滤器使用 XML 编写，可能包含：引用其他过滤器、流量过滤规则或两者的组合。以上引用的过滤器 clean-traffic 是一个过滤器，仅包含对其他过滤器的引用，而不包含实际过滤规则。由于可以使用对其他过滤器的引用，因此可以构建过滤器树。可以使用命令查看 clean-traffic 过滤器：# virsh nwfilter-dumpxml clean-traffic。

如前文所述，多个虚拟机可以引用一个网络过滤器。由于接口通常具有与其相应流量过滤规则关联的个别参数，因此可使用变量来常规化过滤器 XML 中描述的规则。在这种情况下，变量名称在过滤器 XML 中使用，并在引用过滤器的位置提供名称和值。

例 17.2. 描述扩展

在以下示例中，接口描述已扩展，参数 IP 和带点的 IP 地址作为值。

```
<devices>
  <interface type='bridge'>
    <mac address='00:16:3e:5d:c7:9e'/>
    <filterref filter='clean-traffic'>
      <parameter name='IP' value='10.0.0.1'/>
    </filterref>
  </interface>
</devices>
```

在这个特定示例中，清理流量过滤器将以 IP 地址参数 10.0.0.1 表示，根据规则规定该接口的所有流量始终都使用 10.0.0.1 作为源 IP 地址，这是此特定过滤器的一种目的。

17.14.2. 过滤链

过滤规则在过滤链中进行组织。这些链可以看作具有树结构，并将规则过滤为单个链(branches)中的条目。

数据包在根链中启动过滤评估，然后在其它链中继续评估，从这些链返回回根链，或者由其中一个遍历链的过滤规则丢弃或接受。

libvirt 的网络过滤系统会自动为每个虚拟机的网络接口创建独立 root 链，用户选择激活流量过滤。用户可以编写过滤规则，这些规则直接以根链中实例化，或者创建特定于协议的过滤链，以便有效地评估协议特定规则。

存在以下链：

- root
- mac
- STP（跨树协议）
- VLAN

- ARP 和 rarp
- ipv4
- ipv6

评估 mac、stp、vlan、arp、rarp、ipv4 或 ipv6 协议的多个链只能使用链名称中的协议名称创建为前缀。

例 17.3. ARP 流量过滤

这个示例允许指定名称 arp-xyz 或 arp-test 的接口，并在这些链中评估其 ARP 协议数据包。

以下过滤器 XML 显示了 arp 链中过滤 ARP 流量的示例。

```
<filter name='no-arp-spoofing' chain='arp' priority='-500'>
  <uuid>f88f1932-debf-4aa1-9fbe-f10d3aa4bc95</uuid>
  <rule action='drop' direction='out' priority='300'>
    <mac match='no' srcmacaddr='$MAC'/>
  </rule>
  <rule action='drop' direction='out' priority='350'>
    <arp match='no' arpsrcmacaddr='$MAC'/>
  </rule>
  <rule action='drop' direction='out' priority='400'>
    <arp match='no' arpsrcipaddr='$IP'/>
  </rule>
  <rule action='drop' direction='in' priority='450'>
    <arp opcode='Reply'/>
    <arp match='no' arpdstmacaddr='$MAC'/>
  </rule>
  <rule action='drop' direction='in' priority='500'>
    <arp match='no' arpdstipaddr='$IP'/>
  </rule>
  <rule action='accept' direction='inout' priority='600'>
    <arp opcode='Request'/>
  </rule>
  <rule action='accept' direction='inout' priority='650'>
    <arp opcode='Reply'/>
  </rule>
  <rule action='drop' direction='inout' priority='1000'/>
</filter>
```

在 arp 链中将 ARP 特定规则放在一个rp 链中，而不是根链中的特定规则，即 ARP 以外的数据包

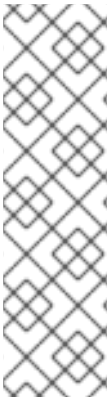
> 协议不需要由 ARP 协议的特定规则评估。这提高了流量过滤的效率。但是，必须注意，必须只注意给定协议的过滤规则到链中，因为不会评估其他规则。例如，没有在 ARP 链中评估 IPv4 规则，因为 IPv4 协议数据包不会遍历 ARP 链。

17.14.3. 过滤链优先级

如前文所述，在创建过滤规则时，所有链都连接到 root 链。这些链的访问顺序按照链的优先级影响。下表显示了可为其分配优先级及其默认优先级的链。

表 17.1. 过滤链默认优先级值

链（前缀）	默认优先级
stp	-810
mac	-800
VLAN	-750
ipv4	-700
ipv6	-600
arp	-500
rarp	-400



注意

在具有较高值值前访问具有较低优先级值的链。

表 17.1 “过滤链默认优先级值” 中列出的链也可以通过将范围 [-1000 到 1000] 范围中的值写入过滤器节点中的优先级(XML)属性来分配自定义优先级。例如，第 17.14.2 节 “过滤链”filter 显示为 arp 链的默认优先级 -500。

17.14.4. 在过滤器中使用变量

网络流量过滤子系统保留了两个变量：MAC 和 IP.

MAC 指定为网络接口的 MAC 地址。引用此变量的过滤规则将自动替换为接口的 MAC 地址。无需明

确提供 MAC 参数，即可执行此操作。尽管可以指定与上述 IP 参数类似的 MAC 参数，但最好不要这样做，因为 libvirt 知道要使用的接口 MAC 地址。

参数 IP 表示虚拟机内操作系统应该在给定接口上使用的 IP 地址。IP 参数在目前是特殊的，因为 libvirt 守护进程将尝试确定在接口上使用的 IP 地址（以及 IP 参数的值），如果未明确提供但引用该参数。对于 IP 地址检测的当前限制，请参阅有关如何使用此功能的限制 第 17.14.12 节 “限制” 以及使用功能的内容。第 17.14.2 节 “过滤链” 中显示的 XML 文件包含过滤器 no-arp-spoofing，这是使用网络过滤器 XML 引用 MAC 和 IP 变量的示例。

请注意，引用的变量始终以字符 $ 前缀。变量的值的格式必须是 XML 中标识的 filter 属性预期的类型。在上例中，IP 参数必须采用标准格式保存法律 IP 地址。如果未提供正确的结构，则会导致过滤器变量没有被一个值替换，并阻止虚拟机启动，或者在使用热插拔时阻止接口附加。每个 XML 属性预期的一些类型显示在示例 例 17.4 “变量类型示例” 中。

例 17.4. 变量类型示例

由于变量可以包含元素列表，变量 IP 可以包含多个在特定接口上有效的 IP 地址，例如：为 IP 变量提供多个元素的表示法：

```
<devices>
  <interface type='bridge'>
    <mac address='00:16:3e:5d:c7:9e'/>
    <filterref filter='clean-traffic'>
      <parameter name='IP' value='10.0.0.1'/>
      <parameter name='IP' value='10.0.0.2'/>
      <parameter name='IP' value='10.0.0.3'/>
    </filterref>
  </interface>
</devices>
```

这个 XML 文件创建过滤器，为每个接口启用多个 IP 地址。每个 IP 地址将导致单独的过滤规则。因此，使用以上 XML 和以下规则，将创建三个单独的过滤规则（每个 IP 地址一个）：

```
<rule action='accept' direction='in' priority='500'>
  <tcp srpipaddr='$IP'/>
</rule>
```

由于可以访问含有元素列表的变量的各个元素，过滤规则（如下所示）访问 DSTPORTS 变量的第二代元素。

```
<rule action='accept' direction='in' priority='500'>
  <udp dstportstart='$DSTPORTS[1]'/>
</rule>
```

例 17.5. 使用各种变量

因为可以使用标记 $VARIABLE[@<iterator id="x">] 创建过滤规则，它们代表来自不同列表中的所有允许规则。以下规则允许虚拟机在 DSTPORTS 中指定的一组端口上接收流量，来自于 SRCIPADDRESSES 中指定的源 IP 地址集合。该规则使用两个独立它器访问其元素，生成变量 DSTPORTS 与 SRCIPADDRESSES 的所有元素组合。

```
<rule action='accept' direction='in' priority='500'>
  <ip srcipaddr='$SRCIPADDRESSES[@1]' dstportstart='$DSTPORTS[@2]'/>
</rule>
```

将具体值分配给 SRCIPADDRESSES 和 DSTPORTS，如下所示：

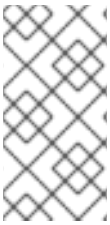
```
SRCIPADDRESSES = [ 10.0.0.1, 11.1.2.3 ]
DSTPORTS = [ 80, 8080 ]
```

使用 $SRCIPADDRESSES[@1] 和 $DSTPORTS[@2] 为变量分配值会导致创建所有变体的地址和端口，如下所示：

- **10.0.0.1, 80**
- **10.0.0.1, 8080**
- **11.1.2.3, 80**
- **11.1.2.3, 8080**

使用单一迭代器访问同一变量，例如使用标记 $SRCIPADDRESSES[@1] 和 $DSTPORTS[@1]，这会导致对这两个列表进行并行访问，并导致以下组合使用：

- **10.0.0.1, 80**
- **11.1.2.3, 8080**



注意

$VARIABLE 是 $VARIABLE[@0] 的缩写。前面的表示法始终假设使用 iterator id="0" 的角色，如本节顶部的"打开段落"中所述。

17.14.5. 自动 IP 地址检测和 DHCP 侦听

本节提供有关自动 IP 地址检测和 DHCP 侦听的信息。

17.14.5.1. 简介

如果引用变量 IP 但不为其分配值，则虚拟机接口中使用的 IP 地址检测会被自动激活。变量 CTRL_IP_LEARNING 可用于指定要使用的 IP 地址学习方法。有效值包括： 任何 、 dhcp 或 none。

值指示 libvirt 使用任何数据包来确定虚拟机使用的地址，这是未设置变量 CTRL_IP_LEARNING 的默认设置。这个方法将仅检测每个接口的单个 IP 地址。检测到客户机虚拟机的 IP 地址后，其 IP 网络流量将被锁定至该地址（例如，其过滤器会阻止 IP 地址 spoofing）。在这种情况下，虚拟机的用户将无法更改客户机虚拟机中的接口的 IP 地址，这将被视为 IP 地址 spoofing。将客户机虚拟机迁移到另一台主机物理计算机或在暂停操作后恢复时，客户机虚拟机发送的第一个数据包将再次确定 guest 虚拟机在特定接口上可以使用的 IP 地址。

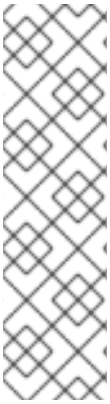
dhcp 的值指示 libvirt 仅遵守具有有效租期的 DHCP 服务器分配地址。此方法支持每个接口检测和使用多个 IP 地址。当客户机虚拟机在暂停操作后恢复时，任何有效的 IP 地址租期都将应用到其过滤器。否则，客户端虚拟机将使用 DHCP 来获取新的 IP 地址。当客户机虚拟机迁移到另一物理主机物理机时，需要客户机虚拟机来重新运行 DHCP 协议。

如果 CTRL_IP_LEARNING 设置为 none，则 libvirt 不会学习 IP 地址并引用 IP，而不为其分配显式值是一个错误。

17.14.5.2. DHCP 侦听

CTRL_IP_LEARNING=dhcp (DHCP snooping)提供额外的反欺骗安全，特别是与过滤器结合使用时，仅允许可信 DHCP 服务器分配 IP 地址。若要启用此功能，请将变量 DHCPSERVER 设置为有效 DHCP 服务器的 IP 地址，并提供使用此变量过滤传入的 DHCP 响应的过滤器。

当 DHCP 侦听启用并且 DHCP 租期过期时，客户机虚拟机将不再能够使用 IP 地址，直到它从 DHCP 服务器获取新的有效租用。如果迁移了客户机虚拟机，它必须获得一个新的有效 DHCP 租用才能使用 IP 地址（例如，通过关闭虚拟机接口并再次启动）。



注意

自动 DHCP 检测侦听客户端虚拟机与基础架构 DHCP 服务器交换的 DHCP 流量。为了避免 libvirt 上的拒绝服务攻击，评估这些数据包的速率限制，这意味着客户机虚拟机每秒发送过多的 DHCP 数据包数量，因此过滤器可能无法改编。假定假定为每秒发送少量 DHCP 数据包。此外，务必要在基础架构中的所有客户机虚拟机设置适当的过滤器，以避免它们发送 DHCP 数据包。因此，客户机虚拟机必须防止将 UDP 和 TCP 流量从端口 67 发送到端口 68，或者所有 guest 虚拟机上应当使用 DHCPSERVER 变量，以仅允许来自受信任的 DHCP 服务器。在子网中的所有客户机虚拟机上必须同时启用反欺骗防止。

例 17.6. 激活 DHCP 侦听的 IP

以下 XML 提供了使用 DHCP 侦听方法学习的 IP 地址激活的示例：

```
<interface type='bridge'>
  <source bridge='virbr0'/>
  <filterref filter='clean-traffic'>
    <parameter name='CTRL_IP_LEARNING' value='dhcp'/>
  </filterref>
</interface>
```

17.14.6. 保留变量

表 17.2 “保留变量” 显示 libvirt 被视为保留和供 libvirt 使用的变量：

表 17.2. 保留变量

变量名称	定义
MAC	接口的 MAC 地址
IP	接口使用的 IP 地址列表
IPV6	当前未实施：接口正在使用的 IPV6 地址列表
DHCPSERVER	可信 DHCP 服务器的 IP 地址列表
DHCPSERVERV6	当前还未实施：可信 DHCP 服务器的 IPv6 地址列表
CTRL_IP_LEARNING	IP 地址检测模式的选择

17.14.7. 元素和属性概述

所有网络过滤器所需的 root 元素均使用两个可能的属性命名 <过滤器> ：name 属性提供给定过滤器的唯一名称。chain 属性是可选的，但允许某些过滤器更好地组织，以便更加有效地由底层主机物理计算机的防火墙子系统处理。目前，系统只支持以下链： root、ipv4、ipv6、arp 和 rarp。

17.14.8. 其他过滤器的引用

任何过滤器都可以包含对其他过滤器的引用。可以在过滤树中多次引用单个过滤器，但过滤器之间的引用不能引入循环。

例 17.7. 清理流量过滤器的示例

以下显示了引用其他过滤器的 clean-traffic 网络过滤器的 XML。

```
<filter name='clean-traffic'>
  <uuid>6ef53069-ba34-94a0-d33d-17751b9b8cb1</uuid>
  <filterref filter='no-mac-spoofing'/>
  <filterref filter='no-ip-spoofing'/>
  <filterref filter='allow-incoming-ipv4'/>
  <filterref filter='no-arp-spoofing'/>
  <filterref filter='no-other-l2-traffic'/>
  <filterref filter='qemu-announce-self'/>
</filter>
```

要引用另一个过滤器，需要在 <过滤器节点中提供 XML 节点过滤器ref>。此节点必须具有属性过滤器，其值必须包含要引用的过滤器的名称。

可以随时定义新的网络过滤器，并且可能包含对 libvirt 未知的网络过滤器的引用。但是，一旦虚拟机启动或引用过滤器的网络接口才是热插，过滤器树中的所有网络过滤器都必须可用。否则，虚拟机将不会启动，或者无法附加网络接口。

17.14.9. 过滤规则

以下 XML 显示了一个简单的网络流量示例，在传出 IP 数据包中的 IP 地址（通过变量 IP 的值提供）时，会过滤实施规则以丢弃流量，从而防止虚拟机 IP 地址欺骗。

例 17.8. 网络流量过滤示例

```
<filter name='no-ip-spoofing' chain='ipv4'>
  <uuid>fce8ae33-e69e-83bf-262e-30786c1f8072</uuid>
  <rule action='drop' direction='out' priority='500'>
```

```
    <ip match='no' srcipaddr='$IP'/>
  </rule>
</filter>
```

流量过滤规则以规则节点开头。此节点可能包含以下三个属性：

- 操作是必须的：
  - drop（匹配规则会静默地丢弃数据包而不进行进一步分析）
  - reject（匹配规则会生成 ICMP 拒绝消息，而不进行进一步分析）
  - 接受（匹配规则接受不进一步分析的数据包）
  - 返回（匹配规则通过此过滤器，但返回到调用过滤器以便进一步分析）
  - 继续（匹配规则进入下一规则以便进一步分析）
- 方向是强制值：
  - 中用于传入流量
  - 出站流量
  - 传入和传出流量
- 优先级是可选的。规则的优先级控制规则相对于其他规则的实例化顺序。具有较低值的规则将在具有更高值的规则前实例化。有效值在 -1000 到 1000 范围内。如果未提供此属性，则默认分配优先级 500。请注意，根据优先级，在 root 链中过滤规则按其优先级对连接到 root 链的过滤器进行排序。这允许交集过滤规则，并可访问 来过滤链。如需更多信息，请参阅 第 17.14.3 节 “过滤链优先级”。

- statematch 是可选的。可能的值有 '0' 或 'false'，用于关闭底层连接状态。默认设置为 'true' 或 1

如需更多信息，请参阅 第 17.14.11 节 “高级过滤器配置主题”。

上面的 例 17.7 “清理流量过滤器的示例” 示例表示，类型为 ip 的流量将 与链 ipv4 关联，该规则将具有 priority=500。例如，如果一个过滤器被引用，其 类型为 ip 的流量也与 chain ipv4 关联，那么该过滤器的规则将按照所示规则的 priority=500 进行排序。

规则可以包含单个规则，用于过滤流量。以上示例显示要过滤类型为 ip 的流量。

17.14.10. 支持的协议

以下小节列出了并提供了有关网络过滤子系统支持的协议的一些详细信息。这类流量规则作为嵌套节点在规则节点中提供。根据规则过滤的流量类型，属性会有所不同。上例演示了 ip 流量过滤节点中有效的单一属性 srcipaddr。以下小节展示了哪些属性有效，以及它们期望的数据类型。可用的数据类型如下：

- UINT8：8 位整数；范围 0-255
- UINT16：16 位整数；范围 0-65535
- MAC_ADDR:以点十进制格式的 MAC 地址，如 00:11:22:33:44:55
- MAC_MASK：MAC 地址格式的 MAC 地址掩码，如 FF:FF:FF:FC:00:00
- IP_ADDR:以带点十进制格式的 IP 地址，如 10.1.2.3
- IP_MASK:IP 地址掩码使用点十进制格式(255.255.248.0)或 CIDR 掩码(0-32)
- IPV6_ADDR:以数字格式的 IPv6 地址，如 FFFF::1

- *IPV6_MASK:以数字格式的 IPv6 掩码(FFFF:FFFF:FC00::)或 CIDR 掩码(0-128)*

- *字符串：一个字符串*

- *BOOLEAN: 'true', 'yes', '1' or 'false', 'no', '0'*

- *IPSETFLAGS：最多 6 个 'src' 或 'dst' 元素的 ipset 的源和目的地标志选择来自数据包标头的源或目标部分的功能，例如： src,src,dst。这里提供的 'selectors' 数量取决于引用的 ipset 类型*

除了 IP_MASK 或 IPV6_MASK 之外的每个属性外，可以使用值 no 的 match 属性来求反。多个指定属性可以分组在一起。以下 XML 片段显示使用抽象属性的示例。

```
[...]
  <rule action='drop' direction='in'>
    <protocol match='no' attribute1='value1' attribute2='value2'/>
    <protocol attribute3='value3'/>
  </rule>
[...]
```

规则的行为会评估规则，并在给定协议属性的边界内查看它。因此，如果单个属性值与规则中提供的值不匹配，则在评估过程中将跳过整个规则。因此，在上述示例中，仅当协议属性 attribute1 不匹配 value1 和 protocol 属性属性2 不匹配 value 2 且协议属性 属性3 与 value3 匹配时，才会丢弃传入流量。

17.14.10.1. MAC(Ethernet)

协议 ID：mac

这个类型的规则应该进入 root 链。

表 17.3. MAC 协议类型

属性名称	datatype	定义
srcmacaddr	MAC_ADDR	发件人的 MAC 地址
srcmacmask	MAC_MASK	应用到发送者的 MAC 地址的掩码

属性名称	datatype	定义
dstmacaddr	MAC_ADDR	目的地的 MAC 地址
dstmacmask	MAC_MASK	应用到目的地的 MAC 地址的掩码
protocolid	UINT16(0x600-0x selfLink), STRING	第 3 层协议 ID。有效字符串包括 [arp, rarp, ipv4, ipv6]
注释	字符串	最多 256 个字符的文本字符串

过滤器可以编写如下：

```
[...]
<mac match='no' srcmacaddr='$MAC'/>
[...]
```

17.14.10.2. VLAN (802.1Q)

*协议 **ID：vlan***

此类型的规则应该进入 root 或 vlan 链。

表 17.4. VLAN 协议类型

属性名称	datatype	定义
srcmacaddr	MAC_ADDR	发件人的 MAC 地址
srcmacmask	MAC_MASK	应用到发送者的 MAC 地址的掩码
dstmacaddr	MAC_ADDR	目的地的 MAC 地址
dstmacmask	MAC_MASK	应用到目的地的 MAC 地址的掩码
vlan-id	UINT16 (0x0-0xfff, 0 - 4095)	VLAN ID
encap-protocol	UINT16(0x03c-0xfff), String	封装的第 3 协议 ID, 有效字符串是 arp, ipv4, ipv6
注释	字符串	最多 256 个字符的文本字符串

17.14.10.3. STP（Spanning Tree 协议）

协议 ID：sp

这个类型的规则应该进入 root 或 stp 链。

表 17.5. STP 协议类型

属性名称	datatype	定义
srcmacaddr	MAC_ADDR	发件人的 MAC 地址
srcmacmask	MAC_MASK	应用到发送者的 MAC 地址的掩码
type	UINT8	网桥协议数据单元(BPDU)类型
flags	UINT8	BPDU 标志dstmacmask
root-priority	UINT16	根优先级范围开始
root-priority-hi	UINT16 (0x0-0xfff, 0 - 4095)	根优先级范围末尾
root-address	MAC _ADDRESS	根 MAC 地址
root-address-mask	MAC _MASK	根 MAC 地址掩码
roor-cost	UINT32	根路径成本（范围开始）
root-cost-hi	UINT32	根路径成本范围结束
sender-priority-hi	UINT16	发件人优先级范围结束
sender-address	MAC_ADDRESS	BPDU 发件人 MAC 地址
sender-address-mask	MAC_MASK	BPDU 发件人 MAC 地址掩码
端口	UINT16	端口标识符（范围启动）
port_hi	UINT16	端口标识符范围结束
msg-age	UINT16	Message age timer（范围开始）
msg-age-hi	UINT16	消息年龄计时器范围结束
max-age-hi	UINT16	最长年龄时间范围结束

属性名称	datatype	定义
hello-time	UINT16	hello 时间计时器（范围开始）
hello-time-hi	UINT16	hello 时间计时器范围结束
forward-delay	UINT16	转发延迟（范围开始）
forward-delay-hi	UINT16	转发延迟范围结束
注释	字符串	最多 256 个字符的文本字符串

17.14.10.4. ARP/RARP

*协议 **ID : arp 或 rarp***

*这个类型的规则应该进入 **root 或 arp/rarp 链。***

表 17.6. ARP 和 RARP 协议类型

属性名称	datatype	定义
srcmacaddr	MAC_ADDR	发件人的 MAC 地址
srcmacmask	MAC_MASK	应用到发送者的 MAC 地址的掩码
dstmacaddr	MAC_ADDR	目的地的 MAC 地址
dstmacmask	MAC_MASK	应用到目的地的 MAC 地址的掩码
hwtype	UINT16	硬件类型
protocoltype	UINT16	协议类型
opcode	UINT16, 字符串	opcode 有效字符串有：请求, Reply, Request_Reverse, Reply_Reverse, DRARP_Request, DRARP_Reply, DRARP_Error, InARP_Request, ARP_NAK
arpsrcmacaddr	MAC_ADDR	ARP/RARP 数据包中的源 MAC 地址
arpdstmacaddr	MAC _ADDR	ARP/RARP 数据包中的目的地 MAC 地址

属性名称	datatype	定义
arpsrcipaddr	IP_ADDR	ARP/RARP 数据包中的源 IP 地址
arpdstipaddr	IP_ADDR	ARP/RARP 数据包中的目的地 IP 地址
gratuitous	布尔值	布尔值指明是否检查一个 gratuitus ARP 数据包
注释	字符串	最多 256 个字符的文本字符串

17.14.10.5. IPv4

*协议 **ID：ip***

*这个类型的规则应该进入 **root 或 ipv4 链。***

表 17.7. IPv4 协议类型

属性名称	datatype	定义
srcmacaddr	MAC_ADDR	发件人的 MAC 地址
srcmacmask	MAC_MASK	应用到发送者的 MAC 地址的掩码
dstmacaddr	MAC_ADDR	目的地的 MAC 地址
dstmacmask	MAC_MASK	应用到目的地的 MAC 地址的掩码
srcipaddr	IP_ADDR	源 IP 地址
srcipmask	IP_MASK	应用到源 IP 地址的掩码
dstipaddr	IP_ADDR	目标 IP 地址
dstipmask	IP_MASK	应用到目标 IP 地址的掩码
协议	UINT8, 字符串	第 4 层协议标识符。协议的有效字符串有：tcp、udp、udplite、esp、ah、icmp、igmp、sctp
srcportstart	UINT16	有效源端口的范围开始；需要协议

属性名称	datatype	定义
srcportend	UINT16	有效源端口范围结束 ; 需要协议
dstportstart	UNIT16	有效目的端口的范围开始；需要协议
dstportend	UNIT16	有效目的端口范围结束 ; 需要协议
注释	字符串	最多 256 个字符的文本字符串

17.14.10.6. IPv6

协议 ID： ipv6

这个类型的规则应该进入 root 或 ipv6 链。

表 17.8. IPv6 协议类型

属性名称	datatype	定义
srcmacaddr	MAC_ADDR	发件人的 MAC 地址
srcmacmask	MAC_MASK	应用到发送者的 MAC 地址的掩码
dstmacaddr	MAC_ADDR	目的地的 MAC 地址
dstmacmask	MAC_MASK	应用到目的地的 MAC 地址的掩码
srcipaddr	IP_ADDR	源 IP 地址
srcipmask	IP_MASK	应用到源 IP 地址的掩码
dstipaddr	IP_ADDR	目标 IP 地址
dstipmask	IP_MASK	应用到目标 IP 地址的掩码
协议	UINT8, 字符串	第 4 层协议标识符。协议的有效字符串有：tcp、udp、udplite、esp、ah、icmpv6、sctp
scrportstart	UNIT16	有效源端口的范围开始；需要协议
srcportend	UINT16	有效源端口范围结束 ; 需要协议

属性名称	datatype	定义
dstportstart	UNIT16	有效目的端口的范围开始；需要协议
dstportend	UNIT16	有效目的端口范围结束 ; 需要协议
注释	字符串	最多 256 个字符的文本字符串

17.14.10.7. TCP/UDP/SCTP

*协议 **ID：tcp、udp、sctp***

*对于这种类型的流量，**chain** 参数将被忽略，并且应省略或设置为 **root**。*

表 17.9. TCP/UDP/SCTP 协议类型

属性名称	datatype	定义
srcmacaddr	MAC_ADDR	发件人的 MAC 地址
srcipaddr	IP_ADDR	源 IP 地址
srcipmask	IP_MASK	应用到源 IP 地址的掩码
dstipaddr	IP_ADDR	目标 IP 地址
dstipmask	IP_MASK	应用到目标 IP 地址的掩码
scripto	IP_ADDR	源 IP 地址的范围开始
srcipfrom	IP_ADDR	源 IP 地址的范围结束
dstipfrom	IP_ADDR	目标 IP 地址的范围开始
dstipto	IP_ADDR	目标 IP 地址的范围结束
scrportstart	UNIT16	有效源端口的范围开始；需要协议
srcportend	UINT16	有效源端口范围结束 ; 需要协议
dstportstart	UNIT16	有效目的端口的范围开始；需要协议

属性名称	datatype	定义
dstportend	UNIT16	有效目的端口范围结束；需要协议
注释	字符串	最多 256 个字符的文本字符串
状态	字符串	以逗号分隔的 NEW、ESTABLISHED、RELATED、INVALID 或 NONE 的列表
flags	字符串	仅限 TCP-only：带有掩码和标记的掩码/标记格式，各自是以逗号分开的 SYN、ACK、URG、PSH、FIN、RST 或 NONE 或 ALL 的列表
ipset	字符串	在 libvirt 之外管理的 IPSet 的名称
ipsetflags	IPSETFLAGS	IPSet 的标记；需要 ipset 属性

17.14.10.8. ICMP

*协议 **ID：icmp***

备注：对于这种类型的流量，chain 参数将被忽略，并且应省略或设置为 root。

表 17.10. ICMP 协议类型

属性名称	datatype	定义
srcmacaddr	MAC_ADDR	发件人的 MAC 地址
srcmacmask	MAC_MASK	掩码应用到发件人的 MAC 地址
dstmacaddr	MAD_ADDR	目的地的 MAC 地址
dstmacmask	MAC_MASK	应用到目的地的 MAC 地址的掩码
srcipaddr	IP_ADDR	源 IP 地址
srcipmask	IP_MASK	应用到源 IP 地址的掩码
dstipaddr	IP_ADDR	目标 IP 地址
dstipmask	IP_MASK	应用到目标 IP 地址的掩码

属性名称	datatype	定义
srcipfrom	IP_ADDR	源 IP 地址的范围开始
scripto	IP_ADDR	源 IP 地址的范围结束
dstipfrom	IP_ADDR	目标 IP 地址的范围开始
dstipto	IP_ADDR	目标 IP 地址的范围结束
type	UNIT16	ICMP 类型
代码	UNIT16	ICMP 代码
注释	字符串	最多 256 个字符的文本字符串
状态	字符串	以逗号分隔的 NEW、ESTABLISHED、RELATED、INVALID 或 NONE 的列表
ipset	字符串	在 libvirt 之外管理的 IPSet 的名称
ipsetflags	IPSETFLAGS	IPSet 的标记；需要 ipset 属性

17.14.10.9. IGMP, ESP, AH, UDPLITE, 'ALL'

*协议 **ID：igmp, esp, ah, udplite, all***

对于这种类型的流量，chain 参数将被忽略，并且应省略或设置为 root。

表 17.11. IGMP, ESP, AH, UDPLITE, 'ALL'

属性名称	datatype	定义
srcmacaddr	MAC_ADDR	发件人的 MAC 地址
srcmacmask	MAC_MASK	掩码应用到发件人的 MAC 地址
dstmacaddr	MAD_ADDR	目的地的 MAC 地址
dstmacmask	MAC_MASK	应用到目的地的 MAC 地址的掩码
srcipaddr	IP_ADDR	源 IP 地址

属性名称	datatype	定义
srcipmask	IP_MASK	应用到源 IP 地址的掩码
dstipaddr	IP_ADDR	目标 IP 地址
dstipmask	IP_MASK	应用到目标 IP 地址的掩码
srcipfrom	IP_ADDR	源 IP 地址的范围开始
scripto	IP_ADDR	源 IP 地址的范围结束
dstipfrom	IP_ADDR	目标 IP 地址的范围开始
dstipto	IP_ADDR	目标 IP 地址的范围结束
注释	字符串	最多 256 个字符的文本字符串
状态	字符串	以逗号分隔的 NEW、ESTABLISHED、RELATED、INVALID 或 NONE 的列表
ipset	字符串	在 libvirt 之外管理的 IPSet 的名称
ipsetflags	IPSETFLAGS	IPSet 的标记；需要 ipset 属性

17.14.10.10. TCP/UDP/SCTP over IPV6

*协议 ID：**tcp-ipv6**、**udp-ipv6**、**sctp-ipv6***

对于这种类型的流量，chain 参数将被忽略，并且应省略或设置为 root。

表 17.12. TCP、UDP、SCTP over IPv6 协议类型

属性名称	datatype	定义
srcmacaddr	MAC_ADDR	发件人的 MAC 地址
srcipaddr	IP_ADDR	源 IP 地址
srcipmask	IP_MASK	应用到源 IP 地址的掩码
dstipaddr	IP_ADDR	目标 IP 地址

属性名称	datatype	定义
dstipmask	IP_MASK	应用到目标 IP 地址的掩码
srcipfrom	IP_ADDR	源 IP 地址的范围开始
scripto	IP_ADDR	源 IP 地址的范围结束
dstipfrom	IP_ADDR	目标 IP 地址的范围开始
dstipto	IP_ADDR	目标 IP 地址的范围结束
srcportstart	UINT16	有效源端口的范围
srcportend	UINT16	有效源端口的范围结束
dstportstart	UINT16	有效目的端口的范围开始
dstportend	UINT16	有效目标端口的范围结束
注释	字符串	最多 256 个字符的文本字符串
状态	字符串	以逗号分隔的 NEW、ESTABLISHED、RELATED、INVALID 或 NONE 的列表
ipset	字符串	在 libvirt 之外管理的 IPSet 的名称
ipsetflags	IPSETFLAGS	IPSet 的标记；需要 ipset 属性

17.14.10.11. ICMPv6

*协议 **ID：icmpv6***

对于这种类型的流量，chain 参数将被忽略，并且应省略或设置为 root。

表 17.13. ICMPv6 协议类型

属性名称	datatype	定义
srcmacaddr	MAC_ADDR	发件人的 MAC 地址
srcipaddr	IP_ADDR	源 IP 地址

属性名称	datatype	定义
srcipmask	IP_MASK	应用到源 IP 地址的掩码
dstipaddr	IP_ADDR	目标 IP 地址
dstipmask	IP_MASK	应用到目标 IP 地址的掩码
srcipfrom	IP_ADDR	源 IP 地址的范围开始
scripto	IP_ADDR	源 IP 地址的范围结束
dstipfrom	IP_ADDR	目标 IP 地址的范围开始
dstipto	IP_ADDR	目标 IP 地址的范围结束
type	UINT16	ICMPv6 类型
代码	UINT16	ICMPv6 代码
注释	字符串	最多 256 个字符的文本字符串
状态	字符串	以逗号分隔的 NEW、ESTABLISHED、RELATED、INVALID 或 NONE 的列表
ipset	字符串	在 libvirt 之外管理的 IPSet 的名称
ipsetflags	IPSETFLAGS	IPSet 的标记；需要 ipset 属性

17.14.10.12. IGMP, ESP, AH, UDPLITE, 'ALL' over IPv6

协议 ID：igmp-ipv6, esp-ipv6, ah-ipv6, udplite-ipv6, all-ipv6

对于这种类型的流量，chain 参数将被忽略，并且应省略或设置为 root。

表 17.14. IGMP, ESP, AH, UDPLITE, 'ALL' over IPv6 协议类型

属性名称	datatype	定义
srcmacaddr	MAC_ADDR	发件人的 MAC 地址
srcipaddr	IP_ADDR	源 IP 地址

属性名称	datatype	定义
srcipmask	IP_MASK	应用到源 IP 地址的掩码
dstipaddr	IP_ADDR	目标 IP 地址
dstipmask	IP_MASK	应用到目标 IP 地址的掩码
srcipfrom	IP_ADDR	源 IP 地址的范围开始
scripto	IP_ADDR	源 IP 地址的范围结束
dstipfrom	IP_ADDR	目标 IP 地址的范围开始
dstipto	IP_ADDR	目标 IP 地址的范围结束
注释	字符串	最多 256 个字符的文本字符串
状态	字符串	以逗号分隔的 NEW、ESTABLISHED、RELATED、INVALID 或 NONE 的列表
ipset	字符串	在 libvirt 之外管理的 IPSet 的名称
ipsetflags	IPSETFLAGS	IPSet 的标记；需要 ipset 属性

17.14.11. 高级过滤器配置主题

以下部分讨论高级过滤配置主题。

17.14.11.1. 连接跟踪

网络过滤子系统（在 Linux 中）利用连接跟踪支持 IP 表。这有助于强制网络流量（状态匹配）的方向，以及计算和限制与客户机虚拟机的同时连接数量。例如，如果 guest 虚拟机作为服务器打开了 TCP 端口 8080，客户端可以在端口 8080 上连接到 guest 虚拟机。连接跟踪和执行方向，然后阻止客户机虚拟机发起连接（TCP 客户端）端口 8080 到主机物理机重新到远程主机物理机器。更为重要的是，跟踪可帮助远程攻击者建立回客户机虚拟机的连接。例如，如果客户机虚拟机中的用户建立了与攻击者站点上的端口 80 的连接，则攻击者将无法从 TCP 端口 80 发起连接回客户机虚拟机。默认情况下，连接状态与启用连接跟踪，然后打开流量方向的强制实施。

> **例 17.9. 关闭到 TCP 端口的 XML 示例**
>
> 下面显示了一个示例 XML 片段，此特性已被关闭，以便传入到 TCP 端口 12345。

```
[...]
 <rule direction='in' action='accept' statematch='false'>
   <cp dstportstart='12345'/>
 </rule>
[...]
```

现在，允许传入流量到 TCP 端口 12345，但也会从虚拟机中启用从（客户端）TCP 端口 12345（可能可能也可能不取）的启动。

17.14.11.2. 限制连接数量

要限制客户机虚拟机可以建立的连接数量，必须提供规则来设置给定流量类型的连接限制。例如，如果虚拟机应该一次只能 ping 另外一个 IP 地址，应该一次只有一个活跃的 ssh 连接。

例 17.10. 设定到连接的 XML 示例文件

以下 XML 片段可用于限制连接

```
[...]
<rule action='drop' direction='in' priority='400'>
  <tcp connlimit-above='1'/>
</rule>
<rule action='accept' direction='in' priority='500'>
  <tcp dstportstart='22'/>
</rule>
<rule action='drop' direction='out' priority='400'>
  <icmp connlimit-above='1'/>
</rule>
<rule action='accept' direction='out' priority='500'>
  <icmp/>
</rule>
<rule action='accept' direction='out' priority='500'>
  <udp dstportstart='53'/>
</rule>
<rule action='drop' direction='inout' priority='1000'>
  <all/>
</rule>
[...]
```

注意

在接受流量的规则之前，必须在 XML 中列出限制规则。根据 例 17.10 “设定到连接的 XML 示例文件” 中的 XML 文件，添加允许发送到端口 22 的 DNS 流量进入客户端虚拟机的额外规则，以避免 ssh 会话没有因为 ssh 守护进程 DNS 查找失败的原因而建立。离开此规则可能会导致 ssh 客户端意外挂起，因为它尝试连接。对于处理与跟踪流量相关的超时相关的超时，应格外小心。用户在客户机虚拟机中可能已经终止的 ICMP ping 可能在主机物理计算机的连接跟踪系统中有较长的超时，因此不允许另一个 ICMP ping 进行访问。

最佳解决方案是使用以下命令调整主机物理机器 sysfs 中的超时：# echo 3 > /proc/sys/net/netfilter/nf_conntrack_icmp_timeout。此命令将 ICMP 连接跟踪超时设置为 3 秒。这样做的效果是，一旦 ping 被终止，另一个 ping 在 3 秒后可以启动。

如果因为任何原因客户机虚拟机没有正确关闭其 TCP 连接，则会在较长时间内保持打开的连接，特别是在主机物理机器上为大量时间设置 TCP 超时值。另外，任何闲置连接都可能会导致连接跟踪系统中的超时，在数据包交换后可以重新激活。

但是，如果设置限制太低，新启动的连接可能会强制闲置连接到 TCP backoff。因此，应设置连接限制，而不是高，以便新 TCP 连接中波动的流量行为不会导致与闲置连接的关系。

17.14.11.3. 命令行工具

virsh 通过对网络过滤器的生命周期支持进行延长。与网络过滤子系统相关的所有命令都以前缀 nwfilter 开头。可用的命令如下：

- nwfilter-list ：列出所有网络过滤器的 UUID 和名称
- nwfilter-define ：定义一个新的网络过滤器或更新现有网络（必须提供名称）
- nwfilter-undefine ：删除指定的网络过滤器（必须提供名称）。不要删除当前正在使用的网络过滤器。
- nwfilter-dumpxml ：显示指定的网络过滤器（必须提供名称）
- nwfilter-edit ：编辑指定的网络过滤器（必须提供名称）

17.14.11.4. 预先存在的网络过滤器

以下是使用 libvirt 自动安装的示例网络过滤器：

表 17.15. ICMPv6 协议类型

协议名称	描述
allow-arp	可接受到客户机虚拟机的所有传入和出站地址解析协议(ARP)流量。
no-arp-spoofing,**no-arp-mac-spoofing** , 和 **no-arp-ip-spoofing**	这些过滤器可防止客户机虚拟机欺骗 ARP 流量。另外，它们仅允许 ARP 请求和回复消息，并强制这些数据包包含： ● no-arp-spoofing - 客户机的 MAC 和 IP 地址 ● no-arp-mac-spoofing - 客户端的 MAC 地址 ● no-arp-ip-spoofing - 客户端的 IP 地址
low-dhcp	允许客户机虚拟机通过 DHCP 请求 IP 地址（来自任何 DHCP 服务器）。
low-dhcp-server	允许客户机虚拟机从指定的 DHCP 服务器请求 IP 地址。DHCP 服务器的十进制 IP 地址必须在对此过滤器的引用中提供。变量的名称必须是 *DHCPSERVER*。
low-ipv4	接受虚拟机的所有传入和传出 IPv4 流量。
low-incoming-ipv4	仅接受虚拟机的传入 IPv4 流量。此过滤器是 **clean-traffic** 过滤器的一部分。
no-ip-spoofing	防止客户机虚拟机发送源 IP 地址与数据包内不同的 IP 地址的 IP 数据包。此过滤器是 **clean-traffic** 过滤器的一部分。
no-ip-multicast	防止客户机虚拟机发送 IP 多播数据包。
no-mac-broadcast	防止将 IPv4 传出流量到指定的 MAC 地址。此过滤器是 **clean-traffic** 过滤器的一部分。
no-other-l2-traffic	防止除由网络使用的其他过滤器指定的流量之外的所有第 2 层网络流量。此过滤器是 **clean-traffic** 过滤器的一部分。
no-other-rarp-traffic,**qemu-announce-self**,**qemu-announce-self-rarp**	这些过滤器允许 QEMU 自助式地址解析协议(RARP)数据包，但阻止所有其他 RARP 流量。它们也包含在 **clean-traffic** 过滤器中。

协议名称	描述
clean-traffic	防止 MAC、IP 和 ARP 欺骗。此过滤器将其他几个过滤器作为构建块引用。

这些过滤器只是构建块，需要与其他过滤器结合使用来提供有用的网络流量过滤。上方列表中最常用的一个是 clean-traffic 过滤器。例如，此过滤器本身可以与 no-ip-multicast 过滤器相结合，以防止虚拟机在数据包欺骗之上发送 IP 多播流量。

17.14.11.5. 编写您自己的过滤器

由于 libvirt 仅提供了几个示例网络过滤器，您可能会考虑自行编写。在计划进行操作时，您可能需要了解网络过滤子系统及其内部工作的方式。当然，您还必须了解并理解您需要过滤的协议，以便不再比您需要通过的服务外任何流量，而且事实上您要允许的流量通过。

网络过滤子系统目前仅适用于 Linux 主机物理机器上，且仅适用于 QEMU 和 KVM 虚拟机类型。在 Linux 上，它建立在对 ebtables、iptables 和 ip6tables 的支持之上，并利用了其功能。考虑在 第 17.14.10 节 “支持的协议” 中找到的列表，可使用 ebtables 实施以下协议：

- *mac*
- *STP（跨树协议）*
- *vlan (802.1Q)*
- *ARP, rarp*
- *ipv4*
- *ipv6*

任何通过 IPv4 运行的协议都会使用 iptables 支持，通过 IPv6 上的协议使用 ip6tables 来实施。

通过使用 Linux 主机物理机器，由 libvirt 网络过滤子系统创建的所有流量过滤规则首先通过 ebtables 实施的过滤支持，之后仅通过 iptables 或 ip6tables 过滤器实施。如果过滤器树具有协议的规则，其中包括： mac、stp、vlan arp、rarp、ipv4 或 ipv6；将首先自动使用列出的 ebtable 规则和值。

可以为同一协议创建多个链。链的名称必须具有之前枚举协议的前缀。要创建处理 ARP 流量的额外链，可以使用名称 arp-test 的链，如指定。

例如，可以使用 ip protocol 过滤器根据源和目标端口过滤 UDP 流量，并为要接受的 UDP 数据包指定属性、源和目标 IP 地址和端口。这允许使用 ebtables 进行早期过滤 UDP 流量。但是，一旦 IP 或 IPv6 数据包（如 UDP 数据包）已通过 ebtables 层，且在过滤树中至少有一个规则实例化 iptables 或 ip6tables 规则，也需要为这些过滤层提供 UDP 数据包通过。这可以通过包含相应 udp 或 udp-ipv6 流量过滤节点的规则来实现。

例 17.11. 创建自定义过滤器

假设需要过滤器来满足以下要求列表：

- 防止虚拟机的接口来自 MAC、IP 和 ARP 欺骗
- 仅打开虚拟机接口的 TCP 端口 22 和 80
- 允许虚拟机从接口发送 ping 流量，但不允许虚拟机在接口上 ping
- 允许虚拟机进行 DNS 查找（UDP 为端口 53）

防止欺骗网络过滤器的要求由现有的 清理流量网络过滤器实现，因此要执行此操作的方式是从自定义过滤器引用它。

要启用 TCP 端口 22 和 80 的流量，添加了两条规则来启用这种类型的流量。要允许客户机虚拟机发送 ping 流量，为 ICMP 流量添加一条规则。出于简单原因，允许从客户机虚拟机启动常规 ICMP 流量，不会指定给 ICMP 回显请求和响应消息。所有其他流量将阻止访问或由客户机虚拟机启动。要执行此操作，则会添加丢弃所有其他流量的规则。假设 guest 虚拟机名为 test，并且要将我们过滤器与 的接口名为 eth0，系统将创建一个名为 test-eth0 的过滤器。

这些注意事项的结果是以下网络过滤 XML：

```
<filter name='test-eth0'>
  <!- - This rule references the clean traffic filter to prevent MAC, IP and ARP spoofing. By not
providing an IP address parameter, libvirt will detect the IP address the guest virtual machine is
using. - ->
  <filterref filter='clean-traffic'/>

  <!- - This rule enables TCP ports 22 (ssh) and 80 (http) to be reachable - ->
  <rule action='accept' direction='in'>
    <tcp dstportstart='22'/>
  </rule>

  <rule action='accept' direction='in'>
    <tcp dstportstart='80'/>
  </rule>

  <!- - This rule enables general ICMP traffic to be initiated by the guest virtual machine including
ping traffic - ->
  <rule action='accept' direction='out'>
    <icmp/>
  </rule>>

  <!- - This rule enables outgoing DNS lookups using UDP - ->
  <rule action='accept' direction='out'>
    <udp dstportstart='53'/>
  </rule>

  <!- - This rule drops all other traffic - ->
  <rule action='drop' direction='inout'>
    <all/>
  </rule>

</filter>
```

17.14.11.6. 自定义过滤器示例

虽然上述 XML 中的其中一个规则包含客户机虚拟机的 IP 地址作为源或目标地址，但过滤流量可以正常工作。其原因是，而规则的评估在各个接口在内部进行，而规则将根据发送或将接收数据包，而不是其源或目标 IP 地址可以进行额外评估。

例 17.12. 网络接口描述的 XML 示例

测试客户机虚拟机的域 XML 中可能出现的网络接口描述 XML 片段类似如下：

```
[...]
  <interface type='bridge'>
    <source bridge='mybridge'/>
    <filterref filter='test-eth0'/>
  </interface>
[...]
```

要更严格地控制 ICMP 流量，并且强制只能从客户机虚拟机发送 ICMP 回显请求，并且只有 ICMP 回显响应才能被客户机虚拟机接收，以上 ICMP 规则可以替换为以下两条规则：

```
<!- - enable outgoing ICMP echo requests- ->
<rule action='accept' direction='out'>
  <icmp type='8'/>
</rule>
```

```
<!- - enable incoming ICMP echo replies- ->
<rule action='accept' direction='in'>
  <icmp type='0'/>
</rule>
```

例 17.13. 第二个自定义过滤器示例

本例演示如何像上面示例所示构建类似的过滤器，但通过位于 guest 虚拟机中的 ftp 服务器扩展要求列表。这个过滤器的要求如下：

- 防止客户机虚拟机的接口来自 MAC、IP 和 ARP 欺骗
- 在客户机虚拟机接口中仅打开 TCP 端口 22 和 80
- 允许客户机虚拟机从接口发送 ping 流量，但不允许客户机虚拟机在接口上 ping
- 允许客户机虚拟机执行 DNS 查找（UDP 至端口 53）
- 启用 FTP 服务器（在活动模式中），以便它可以在客户机虚拟机中运行

额外的要求允许 FTP 服务器在客户机虚拟机中运行，允许允许端口 21 访问 FTP 控制流量，同时使客户机虚拟机能够建立源自客户机虚拟机的 TCP 端口 20 到 FTP 客户端（FTP 活跃模式）的传出 TCP 连接。本例中如何编写此过滤器和两个可能的解决方案有几种方法。

第一种解决方案使用 TCP 协议的 state 属性，为 Linux 主机物理机器的连接跟踪框架提供 hook。对于 guest 虚拟机初始化的 FTP 数据连接（FTP 活动模式） RELATED 状态，使用 RELATED 状态启用检测客户机虚拟机发起的 FTP 数据连接会导致（或与' 的关系）现有 FTP 控制连接，从而使它通过防火墙。然而， RELATED 状态仅对 FTP 数据路径传出 TCP 连接的完全第一个数

据包有效。之后，状态是 ESTABLISHED，然后同样适用于传入和传出方向。所有这些都与源自客户机虚拟机的 TCP 端口 20 的 FTP 数据流量相关。然后，这会导致以下解决方案：

```
<filter name='test-eth0'>
  <!- - This filter (eth0) references the clean traffic filter to prevent MAC, IP, and ARP spoofing. By not providing an IP address parameter, libvirt will detect the IP address the guest virtual machine is using. - ->
  <filterref filter='clean-traffic'/>

  <!- - This rule enables TCP port 21 (FTP-control) to be reachable - ->
  <rule action='accept' direction='in'>
    <tcp dstportstart='21'/>
  </rule>

  <!- - This rule enables TCP port 20 for guest virtual machine-initiated FTP data connection related to an existing FTP control connection - ->
  <rule action='accept' direction='out'>
    <tcp srcportstart='20' state='RELATED,ESTABLISHED'/>
  </rule>

  <!- - This rule accepts all packets from a client on the FTP data connection - ->
  <rule action='accept' direction='in'>
    <tcp dstportstart='20' state='ESTABLISHED'/>
  </rule>

  <!- - This rule enables TCP port 22 (SSH) to be reachable - ->
  <rule action='accept' direction='in'>
    <tcp dstportstart='22'/>
  </rule>

  <!- -This rule enables TCP port 80 (HTTP) to be reachable - ->
  <rule action='accept' direction='in'>
    <tcp dstportstart='80'/>
  </rule>

  <!- - This rule enables general ICMP traffic to be initiated by the guest virtual machine, including ping traffic - ->
  <rule action='accept' direction='out'>
    <icmp/>
  </rule>

  <!- - This rule enables outgoing DNS lookups using UDP - ->
  <rule action='accept' direction='out'>
    <udp dstportstart='53'/>
  </rule>

  <!- - This rule drops all other traffic - ->
  <rule action='drop' direction='inout'>
    <all/>
  </rule>

</filter>
```

在尝试使用 RELATED 状态的过滤器前，您必须确定已将适当的连接跟踪模块加载到主机物理计

算机的内核。根据内核的版本，您必须在建立与客户机虚拟机的 FTP 连接前运行以下命令之一：

- modprobe nf_conntrack_ftp - 其中可用 OR
- modprobe ip_conntrack_ftp 如果上述版本不可用

如果 FTP 以外的协议与 RELATED 状态结合使用，则必须加载对应的模块。模块可用于协议：ftp、tftp、irc、sip、sctp 和 amanda。

第二个解决方案利用了超过之前解决方案的连接的状态标志。此解决方案利用了事实：当检测到流量非常第一个数据包时，连接的 NEW 状态会有效。因此，如果接受非常第一个流的数据包，流会变为连接，因此进入 ESTABLISHED 状态。因此，可以编写常规规则以便允许 ESTABLISHED 连接到达 guest 虚拟机或由客户机虚拟机发送。这是为由 NEW 状态标识的非常第一次数据包编写特定规则，并规定接受数据的端口。所有未明确接受的封包将被丢弃，因此不会到达 ESTABLISHED 状态。从该端口发送的所有后续数据包也会被丢弃。

```
<filter name='test-eth0'>
  <!- - This filter references the clean traffic filter to prevent MAC, IP and ARP spoofing. By not
providing and IP address parameter, libvirt will detect the IP address the VM is using. - ->
  <filterref filter='clean-traffic'/>

  <!- - This rule allows the packets of all previously accepted connections to reach the guest virtual
machine - ->
  <rule action='accept' direction='in'>
    <all state='ESTABLISHED'/>
  </rule>

  <!- - This rule allows the packets of all previously accepted and related connections be sent from
the guest virtual machine - ->
  <rule action='accept' direction='out'>
    <all state='ESTABLISHED,RELATED'/>
  </rule>

  <!- - This rule enables traffic towards port 21 (FTP) and port 22 (SSH)- ->
  <rule action='accept' direction='in'>
    <tcp dstportstart='21' dstportend='22' state='NEW'/>
  </rule>

  <!- - This rule enables traffic towards port 80 (HTTP) - ->
  <rule action='accept' direction='in'>
    <tcp dstportstart='80' state='NEW'/>
  </rule>

  <!- - This rule enables general ICMP traffic to be initiated by the guest virtual machine, including
ping traffic - ->
  <rule action='accept' direction='out'>
    <icmp state='NEW'/>
```

```
  </rule>

  <!- - This rule enables outgoing DNS lookups using UDP - ->
  <rule action='accept' direction='out'>
    <udp dstportstart='53' state='NEW'/>
  </rule>

  <!- - This rule drops all other traffic - ->
  <rule action='drop' direction='inout'>
    <all/>
  </rule>

</filter>
```

17.14.12. 限制

下表列出了当前已知的网络过滤子系统的限制：

- 只有在目标主机物理机器上也提供了 guest 虚拟机顶级过滤器引用的整个过滤器树时，才支持虚拟机迁移。示例的网络过滤器 清理流量 应该在所有 libvirt 安装中可用，从而启用在引用此过滤器的客户机虚拟机迁移。为了确保版本兼容性不是一个问题，请确定您通过定期更新软件包来使用 libvirt 的最新版本。

- 迁移必须在 0.8.1 或更高版本的 libvirt 之间发生，以防止丢失与接口关联的网络流量过滤器。

- VLAN(802.1Q)数据包如果由客户机虚拟机发送，则无法使用协议 ID、rarp、ipv4 和 ipv6 的规则过滤。它们只能使用协议 ID、MAC 和 VLAN 进行过滤。因此，过滤器清理流量 例 17.1 “网络过滤示例” 无法正常工作。

17.15. 创建 TUNNELS

本节将演示如何实施不同的隧道情景。

17.15.1. 创建多播 Tunnels

多播组被设置为代表虚拟网络。任何网络设备位于相同多播组中的虚拟客户机都可以相互通信，即使在主机物理机器也是如此。此模式也可供非特权用户使用。没有默认的 DNS 或 DHCP 支持，且不进行传出网络访问。为提供传出网络访问，其中一个客户机虚拟机应具有第二个 NIC，该 NIC 连接到前四个网络类型之一，从而提供适当的路由。多播协议兼容客户机虚拟机用户模式。请注意，您提供的源地址必须是用于多播地址块的地址。

要创建多播隧道，请将以下 XML 详情放在 <devices> 元素中：

图 17.28. 多播隧道域 XMl 示例

```
...
<devices>
  <interface type='mcast'>
    <mac address='52:54:00:6d:90:01'>
    <source address='230.0.0.1' port='5558'/>
  </interface>
</devices>
...
```

17.15.2. 创建 TCP Tunnels

TCP 客户端-服务器架构提供虚拟网络。在此配置中，一个 guest 虚拟机提供网络的服务器结尾，而所有其他客户机虚拟机则配置为客户端。所有网络流量都通过客户机虚拟机服务器在客户机虚拟机客户端之间路由。此模式也可供非特权用户使用。请注意，这个模式不提供默认的 DNS 或 DHCP 支持，且不会提供传出网络访问。为提供传出网络访问，其中一个客户机虚拟机应具有第二个 NIC，该 NIC 连接到前四个网络类型之一，从而提供适当的路由。

要创建 TCP 隧道，请将以下 XML 详情放在 <devices> 元素中：

图 17.29. TCP 隧道域 XMl 示例

```
    ...
<devices>
  <interface type='server'>
    <mac address='52:54:00:22:c9:42'>
    <source address='192.168.0.1' port='5558'/>
  </interface>
  ...
  <interface type='client'>
    <mac address='52:54:00:8b:c9:51'>
    <source address='192.168.0.1' port='5558'/>
  </interface>
</devices>
...
```

17.16. 设置 VLAN 标签

使用 virsh net-edit 命令添加虚拟局域网(vLAN) 标签。此标签也可用于 SR-IOV 设备的 PCI 设备分配。如需更多信息，请参阅 第 16.2.3 节 “使用 SR-IOV 设备配置 PCI 分配”。

图 17.30. vSetting VLAN 标签（仅支持网络类型）

```
<network>
  <name>ovs-net</name>
  <forward mode='bridge'/>
  <bridge name='ovsbr0'/>
  <virtualport type='openvswitch'>
    <parameters interfaceid='09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f'/>
  </virtualport>
  <vlan trunk='yes'>
    <tag id='42' nativeMode='untagged'/>
    <tag id='47'/>
  </vlan>
  <portgroup name='dontpanic'>
    <vlan>
      <tag id='42'/>
    </vlan>
  </portgroup>
</network>
```

如果（且只有）网络类型支持对客户机透明的 vlan 标记，可选的 <vlan> 元素可以指定一个或多个 vlan 标签，以应用于使用此网络的所有虚拟客户机的流量。（openvswitch 和 type='hostdev 的 SR-IOV 网络支持透明的 VLAN 标记，则包括标准 Linux 网桥和 libvirt 自身的虚拟网络等内容。802.1Qbh(vn-link)和 802.1Qbg(VEPA)交换机提供自己的方法（在 libvirt 外），以标记到特定 vlans 的客户机流量。） 如预期一样，tag 属性指定要使用的 vlan 标签。如果网络定义了多个 <vlan> 元素，则假定用户希望利用所有指定标签进行 VLAN 中继。如果需要使用单一标签的 vlan 中继，可选属性 trunk='yes' 可以添加到 vlan 元素。

对于使用 openvswitch 的网络连接，可以配置 'native-tagged' 和 'native-tagged' vlan 模式。这会在 <tag> 元素中使用可选的 nativeMode 属性： nativeMode 可以被设置为 'tagged' 或 'untagged'。元素的 id 属性设置原生 vlan。

VLAN 元素也可以在 <端口> 组元素中指定，也可直接在域的 <interface> 元素中指定。<>如果在多个位置上指定了 vlan 标签，<接口> 中的设置将具有优先权，后跟接口配置选择的 <portgroup> 中的设置。只有在 <端口> 组或 <接口> 中未给定任何时，才会选择 <vlan> in <network>。

17.17. 将 QOS 应用到您的虚拟网络

服务质量(QoS) 指的是资源控制系统，保证网络上所有用户的最佳体验，确保没有延迟、jitter 或数据包丢失。QoS 可以是特定于应用程序的或用户 / 组。如需更多信息，请参阅 第 23.17.8.14 节 “服务质量(QoS)”。

第 18 章 客户机的远程管理

本节介绍如何远程管理您的客户机。

18.1. 传输模式

对于远程管理，libvirt 支持以下传输模式：

传输层安全性(TLS)

传输层安全性 TLS 1.0(SSL 3.1)经过验证和加密 TCP/IP 套接字，通常在公共端口号上侦听。要使用此功能，您需要生成客户端和服务器证书。标准端口为 16514。具体步骤请查看 第 18.3 节 “通过 TLS 和 SSL 进行远程管理”。

SSH

通过安全 Shell 协议(SSH)连接进行传输。libvirt 守护进程(libvirtd)必须在远程计算机上运行。端口 22 必须处于打开状态才能进行 SSH 访问。您应该使用某种类型的 SSH 密钥管理（例如，ssh-agent 实用程序），或者会提示您输入密码。具体步骤请查看 第 18.2 节 “使用 SSH 进行远程管理”。

Unix 套接字

UNIX 域套接字只能在本地机器上访问。套接字未加密，使用 UNIX 权限或 SELinux 进行身份验证。标准套接字名称是 /var/run/libvirt/libvirt-sock 和 /var/run/libvirt/libvirt-sock-ro （用于只读连接）。

ext

ext 参数用于任何可以在 libvirt 范围外面连接到远程机器的外部程序。不支持此参数。

TCP

未加密的 TCP/IP 套接字.不建议在生产环境中使用，这通常是禁用的，但管理员可以使它进行测试或在可信网络中使用。默认端口为 16509。

如果没有指定其他默认传输，则默认传输是 TLS。

远程 URI

virsh 和 libvirt 使用统一资源标识符(URI)来连接远程主机。URI 也可搭配 --connect 参数用于 virsh 命令，以便在远程主机上执行单个命令或迁移。通过采用普通本地 URI 和添加主机名或传输名称来形成远

程 URI。作为特殊情况，使用 URI 方案 'remote' 将指示远程 libvirtd 服务器探测到最佳管理程序驱动程序。这等同于为本地连接传递 NULL URI

libvirt URI 采用常规形式（用方括号"[]"内容代表可选功能）：

```
driver[+transport]://[username@][hostname][:port]/path[?extraparameters]
```

请注意，如果虚拟机监控程序（驱动程序）是 QEMU，则必须路径。

以下是有效的远程 URI 示例：

- qemu://hostname/

必须提供传输方法或主机名以外部位置为目标。如需更多信息，请参阅 libvirt 上游文档。

远程管理参数示例

- 使用 SSH 传输和 SSH 用户名 virtuser 连接到名为 host2 的远程 KVM 主机。每个连接的 connect 命令都是 connect [URI] [--readonly]。有关 virsh connect 命令的详情请参考 第 20.4 节 “使用 virsh Connect 连接到管理程序”

```
qemu+ssh://virtuser@host2/
```

- 使用 TLS 连接到名为 host2 的主机上的远程 KVM 管理程序。

```
qemu://host2/
```

测试示例

- 使用非标准 UNIX 套接字连接到本地 KVM 管理程序。本例中明确提供了 UNIX 套接字的完整路径。

```
qemu+unix:///system?socket=/opt/libvirt/run/libvirt/libvirt-sock
```

- 使用非加密 TCP/IP 连接到服务器，在端口 5000 上使用 IP 地址 10.1.1.10 连接到 libvirt 守护进程。这会使用带有默认设置的测试驱动程序。

```
test+tcp://10.1.1.10:5000/default
```

额外的 URI 参数

可以将额外参数附加到远程 URI。下表介绍了可识别的参数。所有其他参数将被忽略。请注意，参数值必须是 URI 转义（即问号(?)），在参数和特殊字符转换为 URI 格式之前。

表 18.1. 额外 URI 参数

Name	传输模式	描述	用法示例
名称	所有模式	传递给远程 **virConnectOpen** 功能的名称。名称通常通过从远程 URI 中删除 ***传输、主机名、端口号、用户名*** 和额外参数而形成，但是在某些非常复杂的情况下，最好地提供名称。	name=qemu:///system
命令	SSH 和 ext	external 命令.对于 ext 传输来说，这是必需的。对于 ssh，默认为 ssh。为该命令搜索 PATH。	command=/opt/openssh/bin/ssh
socket	UNIX 和 ssh	UNIX 域套接字的路径，它将覆盖默认值。对于 ssh 传输，这会被传递给远程 **netcat** 命令（请参见 netcat）。	socket=/opt/libvirt/run/libvirt/libvirt-sock
no_verify	tls	如果设置为非零值，这会禁用客户端检查服务器证书。请注意，要禁用客户端的证书或 IP 地址的服务器检查，您必须更改 libvirtd 配置。	no_verify=1
no_tty	ssh	如果设置为非零值，这会停止 ssh。如果无法登录远程机器，则停止 ssh。当您无法访问终端 时，使用它。	no_tty=1

18.2. 使用 SSH 进行远程管理

ssh 软件包提供了一个加密的网络协议，可安全地将管理功能发送到远程虚拟化服务器。下面描述的方法使用 libvirt 管理连接（通过 SSH 连接安全地隧道）来管理远程机器。所有身份验证均使用由本地 SSH 代理收集的 SSH 公钥加密和密码或密码短语来完成。另外，每个 guest 的 VNC 控制台通过 SSH 隧道。

当使用 SSH 远程管理虚拟机时，请注意以下问题：

- 您需要 root 登录对远程机器以管理虚拟机的访问权限。
- 初始连接设置过程可能很慢。
- 没有在所有主机或客户机上撤销用户密钥的标准或简单的方法。
- SSH 无法通过更多远程机器进行很好的扩展。

注意

Red Hat Virtualization 支持远程管理大量虚拟机。详情请查看 Red Hat Virtualization 文档。

SSH 访问需要以下软件包：

- openssh
- openssh-askpass
- openssh-clients
- openssh-server

为 virt-manager配置无密码或密码管理的 SSH 访问

下列说明假设您从头开始，并且尚未设置 SSH 密钥。如果您将 SSH 密钥设置并复制到其他系统，您可以跳过这个过程。

重要

SSH 密钥独立于用户，只能由其所有者使用。密钥的所有者是生成该密钥的用户。密钥可能不会在不同用户间共享。

virt-manager 必须通过拥有密钥来连接远程主机的用户运行。这意味着，如果远程系统由非 root 用户管理，virt-manager 必须以非特权模式运行。如果远程系统由本地 root 用户管理，则 SSH 密钥必须由 root 拥有和创建。

您不能使用 virt-manager 以非特权用户管理本地主机。

1. 可选：更改用户

   根据需要更改用户。这个示例使用本地 root 用户来远程管理其他主机和本地主机。

   ```
   $ su -
   ```

2. 生成 SSH 密钥对

   在使用 virt-manager 的机器上生成一个公钥对。这个示例在 ~/.ssh/ 目录中使用默认密钥位置。

   ```
   # ssh-keygen -t rsa
   ```

3. 将密钥复制到远程主机

   不使用密码或密码短语进行远程登录，要求将 SSH 密钥分发到受管理的系统。使用 ssh-copy-id 命令，在提供的系统地址（示例中为 root@host2.example.com）将密钥复制到 root 用户。

   ```
   # ssh-copy-id -i ~/.ssh/id_rsa.pub root@host2.example.com
   root@host2.example.com's password:
   ```

   之后，尝试登录到机器，并检查 .ssh/authorized_keys 文件，以确保还没有添加意外的密钥：

   ```
   ssh root@host2.example.com
   ```

根据需要为其他系统重复此操作。

4. 可选：将密码短语添加到 ssh-agent

如果需要，将 SSH 密钥的 pass-phrase 添加到 ssh-agent。在本地主机中，使用以下命令添加 pass-phrase（如果有）以启用无密码登录。

```
# ssh-add ~/.ssh/id_rsa
```

如果 ssh-agent 未在运行，这个命令将无法运行。为避免错误或冲突，请确保您的 SSH 参数设置正确。如需更多信息，请参阅 Red Hat Enterprise System Administration Guide。

libvirt 守护进程（libvirtd）

libvirt 守护进程提供了管理虚拟机的接口。您必须已在您要以这种方式管理的每个远程主机上安装并运行 libvirtd 守护进程。

```
$ ssh root@somehost
# systemctl enable libvirtd.service
# systemctl start libvirtd.service
```

配置 libvirtd 和 SSH 后，您应该能够远程访问和管理虚拟机。此时，也应该能够通过 VNC 访问您的 guest。

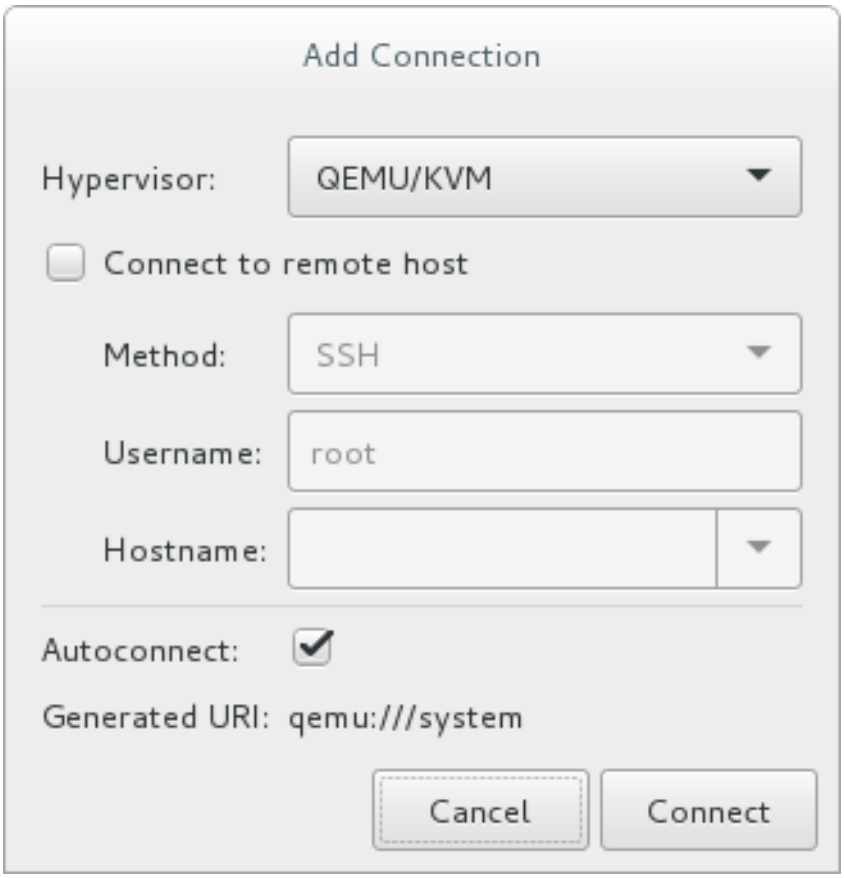
使用 virt-manager 访问远程主机

可以使用 virt-manager GUI 工具管理远程主机。SSH 密钥必须属于执行 virt-manager 的用户，以便免密码登录正常工作。

1. 启动 virt-manager。

2. 打开 File zfcp Add Connection 菜单。

图 18.1. 添加连接菜单



3. 使用下拉菜单选择虚拟机监控程序类型，然后单击 **Connect to remote host** 复选框以打开连接方法（本例中为 **Remote tunnel over SSH**），输入 **User name** 和 **Hostname**，然后单击 **Connect**。

18.3. 通过 TLS 和 SSL 进行远程管理

您可以使用 TLS 和 SSL 协议管理虚拟机。TLS 和 SSL 提供更大的可扩展性，但比 SSH 复杂（请参阅 第 18.2 节 “使用 SSH 进行远程管理”）。TLS 和 SSL 是 Web 浏览器用于安全连接的相同。libvirt 管理连接为传入连接打开一个 TCP 端口，该连接会根据 x509 证书安全地加密和验证。以下流程提供了为 TLS 和 SSL 管理创建和部署身份验证证书的说明。

过程 18.1. 为 TLS 管理创建证书颁发机构(CA)密钥

1. 开始之前，确认已安装了 **gnutls-utils**。如果没有安装，请安装它：

```
# yum install gnutls-utils
```

2. 使用以下命令生成私钥：

```
# certtool --generate-privkey > cakey.pem
```

3. 生成密钥后，创建签名文件，以便密钥可以自签名。要做到这一点，创建一个带有签名详情的文件，并将其命名为 ca.info。此文件应包含以下内容：

```
cn = Name of your organization
ca
cert_signing_key
```

4. 使用以下命令生成自签名证书：

```
# certtool --generate-self-signed --load-privkey cakey.pem --template ca.info --outfile
cacert.pem
```

生成文件后，可以使用 rm 命令删除 ca.info 文件。生成进程的结果的文件名为 cacert.pem。此文件是公钥(certificate)。加载的文件 cakey.pem 是私钥。为了安全起见，该文件应保持私有，而不能驻留在共享空间中。

5. 在 /etc/pki/CA/ cacert.pem 目录中的所有客户端和服务器上安装 cacert.pem CA 证书文件，以便他们知道您的 CA 发布的证书可以信任。要查看此文件的内容，请运行：

```
# certtool -i --infile cacert.pem
```

这是设置您的 CA 所需要的。保持 CA 的私钥安全，因为您需要它才能为客户端和服务器签发证书。

过程 18.2. 发出服务器证书

此流程演示了如何将 X.509 通用名称(CN)字段设置为服务器主机名来签发证书。CN 必须与客户端用来连接到服务器的主机名匹配。在本例中，客户端将使用 URI: qemu://mycommonname/system 连接到服务器，因此 CN 字段应当相同，例如"mycommoname"。

1. 为服务器创建私钥。

```
# certtool --generate-privkey > serverkey.pem
```

2. 首先创建名为 server.info 的模板文件，为 CA 的私钥生成签名。确保将 CN 设置为与服务器

的主机名相同：

```
organization = Name of your organization
cn = mycommonname
tls_www_server
encryption_key
signing_key
```

3. 创建证书：

```
# certtool --generate-certificate --load-privkey serverkey.pem --load-ca-certificate cacert.pem
--load-ca-privkey cakey.pem \ --template server.info --outfile servercert.pem
```

这会导致生成两个文件：

- ServerKey.pem - 服务器的私钥
- servercert.pem - 服务器的公钥

4. 确保保留私钥 secret 的位置。要查看文件的内容，请使用以下命令：

```
# certtool -i --infile servercert.pem
```

打开此文件时，CN= 参数应当与之前设置的 CN 相同。例如，mycommonname。

5. 在以下位置安装两个文件：

- ServerKey.pem - 服务器的私钥。将这个文件放在以下位置：
/etc/pki/libvirt/private/serverkey.pem
- servercert.pem - 服务器的证书。将它安装到服务器上的以下位置：
/etc/pki/libvirt/servercert.pem

过程 18.3. 发出客户端证书

1.

对于每个客户端（例如，与 libvirt 链接的任何程序（如 virt-manager），您需要向证书发出 X.509 区分名称(DN)字段，设置为合适的名称。这需要在企业层面上决定。

例如，会使用以下信息：

```
C=USA,ST=North Carolina,L=Raleigh,O=Red Hat,CN=name_of_client
```

2. 创建一个私钥：

```
# certtool --generate-privkey > clientkey.pem
```

3. 首先创建名为 client.info 的模板文件，为 CA 的私钥生成签名。文件应当包含以下内容（字段应自定义来反映您的地区/位置）：

```
country = USA
state = North Carolina
locality = Raleigh
organization = Red Hat
cn = client1
tls_www_client
encryption_key
signing_key
```

4. 使用以下命令签署证书：

```
# certtool --generate-certificate --load-privkey clientkey.pem --load-ca-certificate cacert.pem \
--load-ca-privkey cakey.pem --template client.info --outfile clientcert.pem
```

5. 在客户端机器上安装证书：

```
# cp clientkey.pem /etc/pki/libvirt/private/clientkey.pem
# cp clientcert.pem /etc/pki/libvirt/clientcert.pem
```

18.4. 配置 VNC 服务器

要在主机和使用虚拟网络计算(VNC)计算机之间进行图形桌面共享，必须在您要连接的客户机上配置 VNC 服务器。为此，必须将 VNC 指定为客户机 XML 文件的 devices 元素中的图形类型。有关详情请参考 第 23.17.11 节 “图形帧缓冲”。

要连接到 VNC 服务器，请使用 virt-viewer 实用程序或 virt-manager 接口。

18.5. 使用 NSS 增强虚拟机远程管理

在 Red Hat Enterprise Linux 7.3 及更新的版本中，您可以使用 libvirt Network Security Services(NSS)模块更轻松地使用 SSH、TLS、SSL 和其他远程登录服务连接到 guest。另外，该模块还的好处是使用主机名转换的实用程序，如 ping。

要使用这个功能，请安装 libvirt-nss 软件包：

```
# yum install libvirt-nss
```

注意

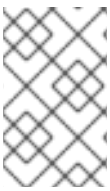
如果安装 libvirt-nss 失败，请确保启用了 Red Hat Enterprise Linux 的 Optional 软件仓库。具体步骤请查看 系统管理员指南。

最后，通过在 /etc/nsswitch.conf 文件的 hosts 行中添加 libvirt_guest 来启用模块，例如：

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

在 hosts 行中列出的模块的顺序决定了这些模块被查询的顺序来查找指定的远程客户端。因此，libvirt 的 NSS 模块被添加到将主机名转换为 IP 地址的模块中。这个示例启用在 NAT 模式中连接到远程客户端，而不设置静态 IP 地址，且只使用客户机的名称：

```
# ssh root@guestname
root@guestname's password:
Last login: Thu Aug 10 09:12:31 2017 from 192.168.122.1
[root@guestname ~]#
```



注意

如果您不知道客户端的名称，您可以使用 virsh list --all 命令获取它。

第 19 章 使用虚拟机管理器(VIRT-MANAGER)管理客户机.

本章论述了虚拟机管理器(virt-manager)窗口、对话框和各种 GUI 控制。

virt-manager 提供主机系统上和远程主机上的系统管理程序和客户机的图形视图。virt-manager 可以执行虚拟化管理任务，包括：

- 定义和创建客户机，
- 分配内存，
- 分配虚拟 CPU、
- 监控运营性能，
- 保存和恢复、暂停和恢复，以及关闭和启动客户机，
- 文本和图形控制台链接，以及
- 实时和离线迁移。

重要

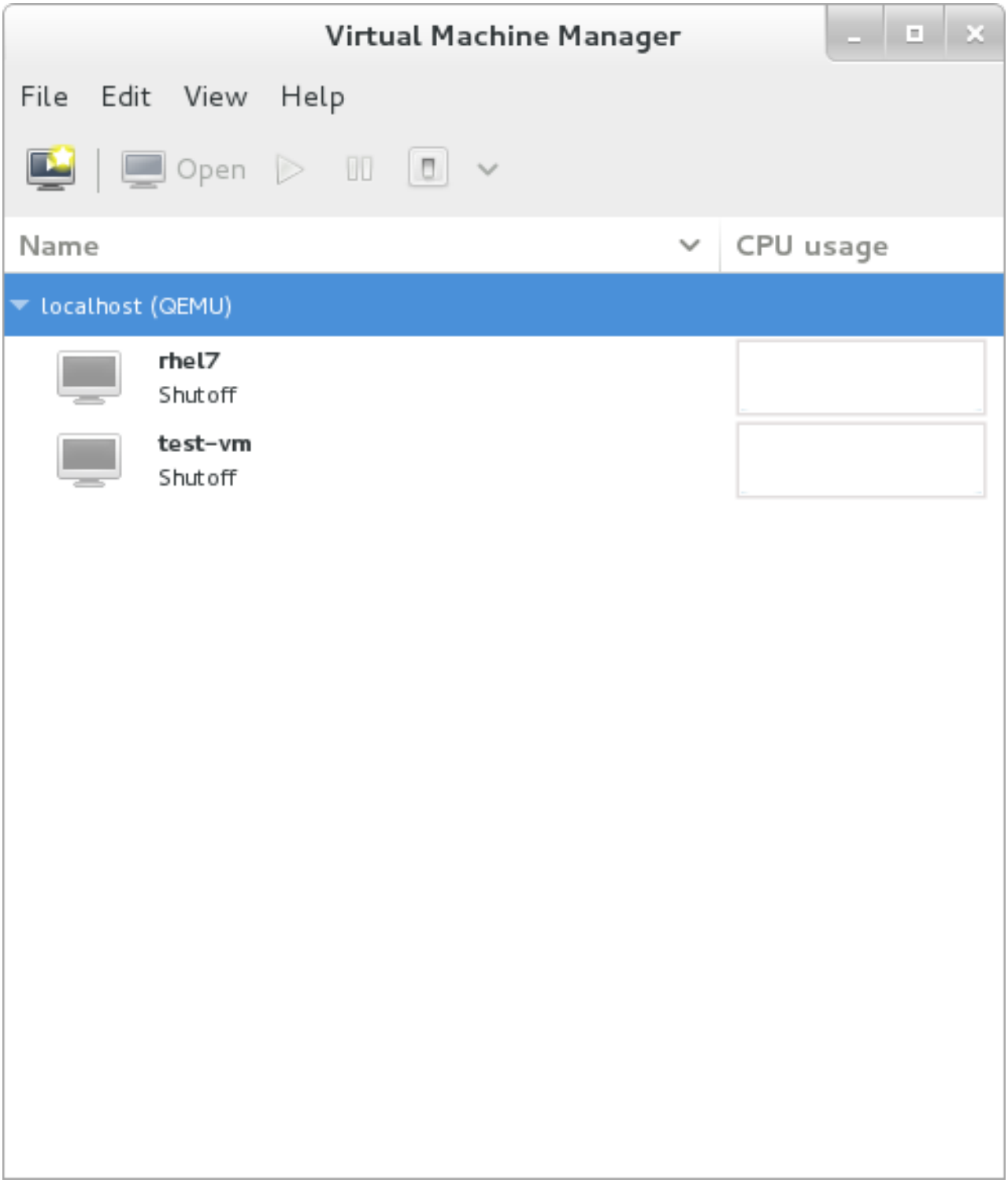
请注意您使用的用户很重要。如果您使用一个用户创建客户机虚拟机，您将无法使用另一个用户检索有关它的信息。这在 virt-manager 中创建虚拟机时尤其重要。在这种情况下，默认用户为 root，除非另外指定。如果您无法使用 virsh list --all 命令列出虚拟机，则很可能是因为使用与您用于创建虚拟机的不同用户运行该命令。

19.1. 启动 VIRT-MANAGER

要启动 virt-manager 会话打开" 应用程序 "菜单，然后 "系统工具 "菜单并选择" 虚拟机管理器"(virt-manager)。

*此时会出现 **virt-manager** 主窗口。*

*图 **19.1.** 启动 **virt-manager***



*或者，可在以下命令中使用 **ssh** 远程启动 **virt-manager** ：*

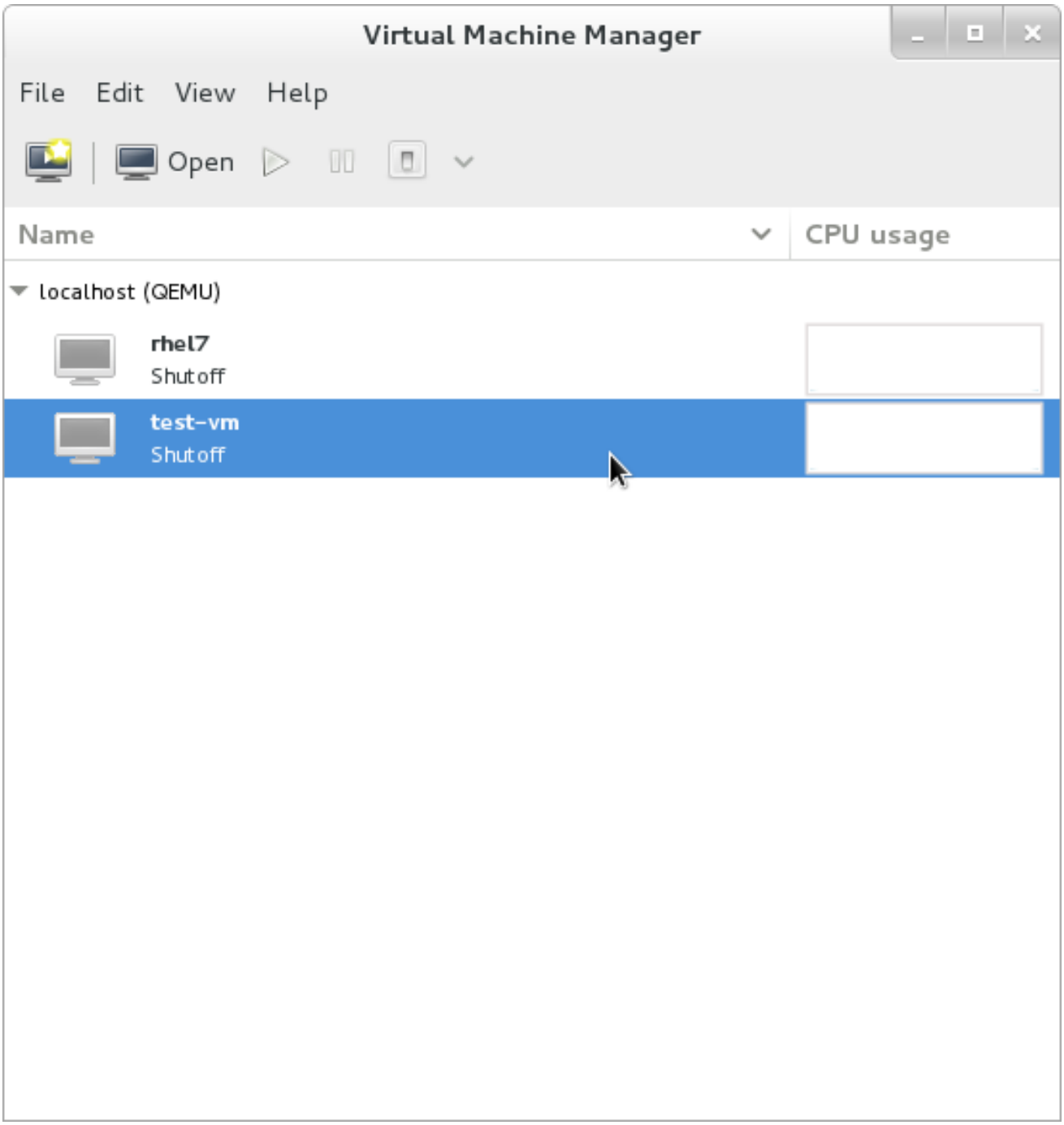
```
# ssh -X host's address
[remotehost]# virt-manager
```

在 第 18.2 节 “使用 SSH 进行远程管理” 中进一步讨论使用 ssh 管理虚拟机和主机。

19.2. VIRTUAL MACHINE MANAGER MAIN 窗口

这个主窗口显示客户机使用的所有正在运行的客户机和资源。通过双击 guest 的名称选择 guest。

图 19.2. 虚拟机管理器主窗口

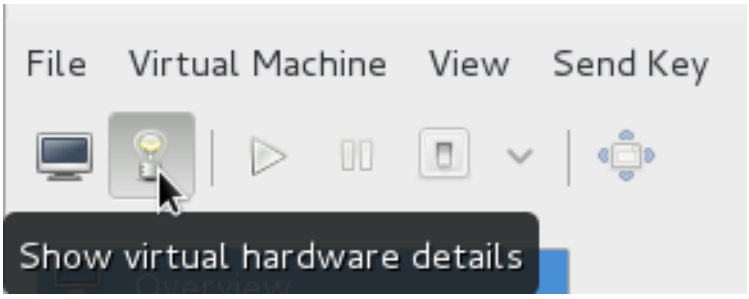


19.3. VIRTUAL HARDWARE DETAILS 窗口

虚拟硬件详细信息窗口显示有关为客户机配置的虚拟硬件的信息。在此窗口中，可以添加、删除和修

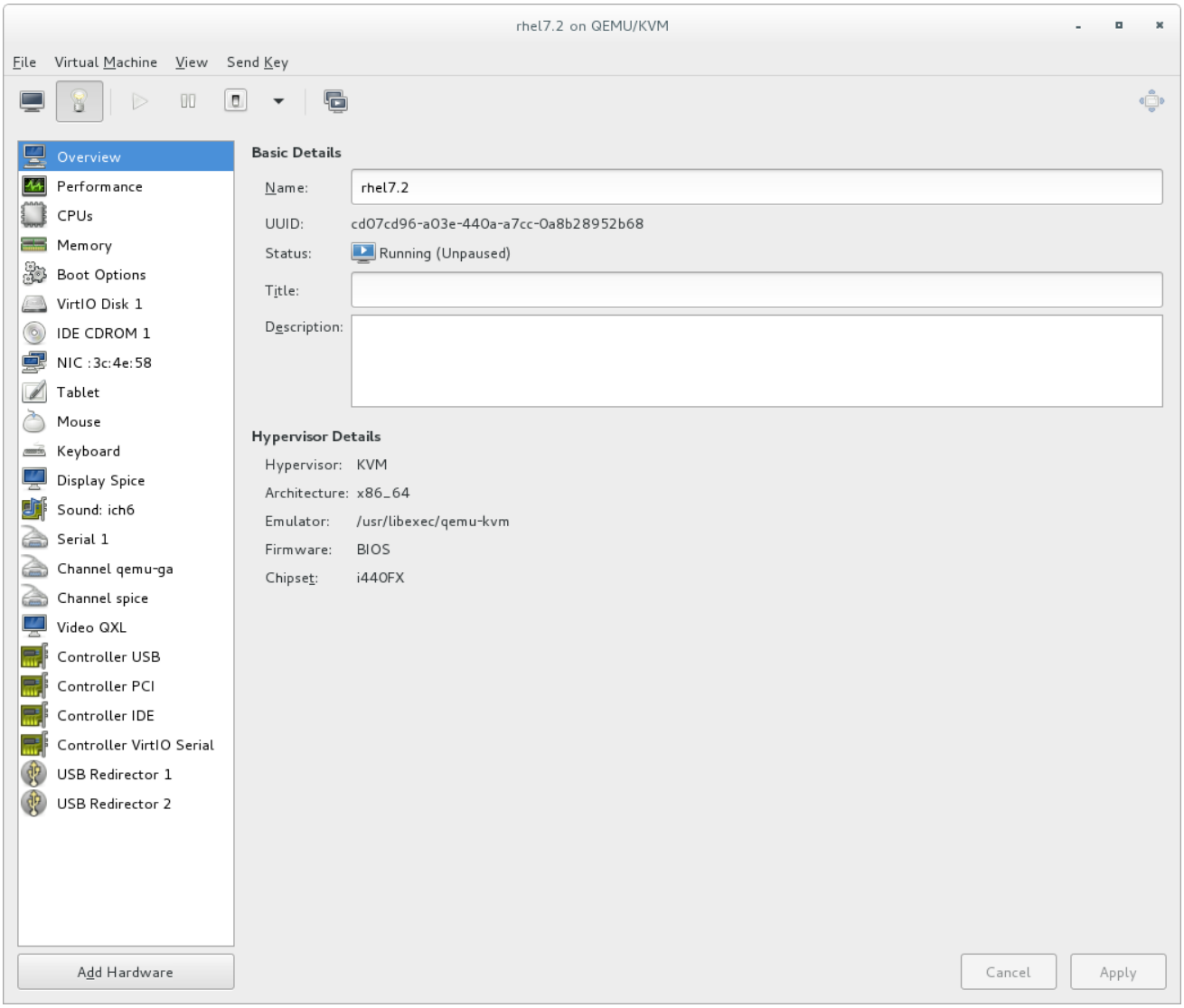
改虚拟硬件资源。要访问虚拟硬件详情窗口，请点击工具栏中的图标。

图 19.3. 虚拟硬件详情图标



单击图标可显示虚拟硬件详细信息窗口。

图 19.4. 虚拟硬件详情窗口



19.3.1. 将引导选项应用到客户机虚拟机

通过使用 virt-manager，您可以选择将在引导时如何操作 guest。在 guest 虚拟机重启前，引导选项

不会生效。在进行任何更改之前，您可以关闭虚拟机，或者之后可以重新启动虚拟机。如果没有这些选项，则下一次客户端重启时将会发生更改。

过程 19.1. 配置引导选项

1. 在 Virtual Machine Manager Edit 菜单中，选择 Virtual Machine Details。

2. 在侧边面板中，选择 Boot Options，然后完成以下可选步骤：

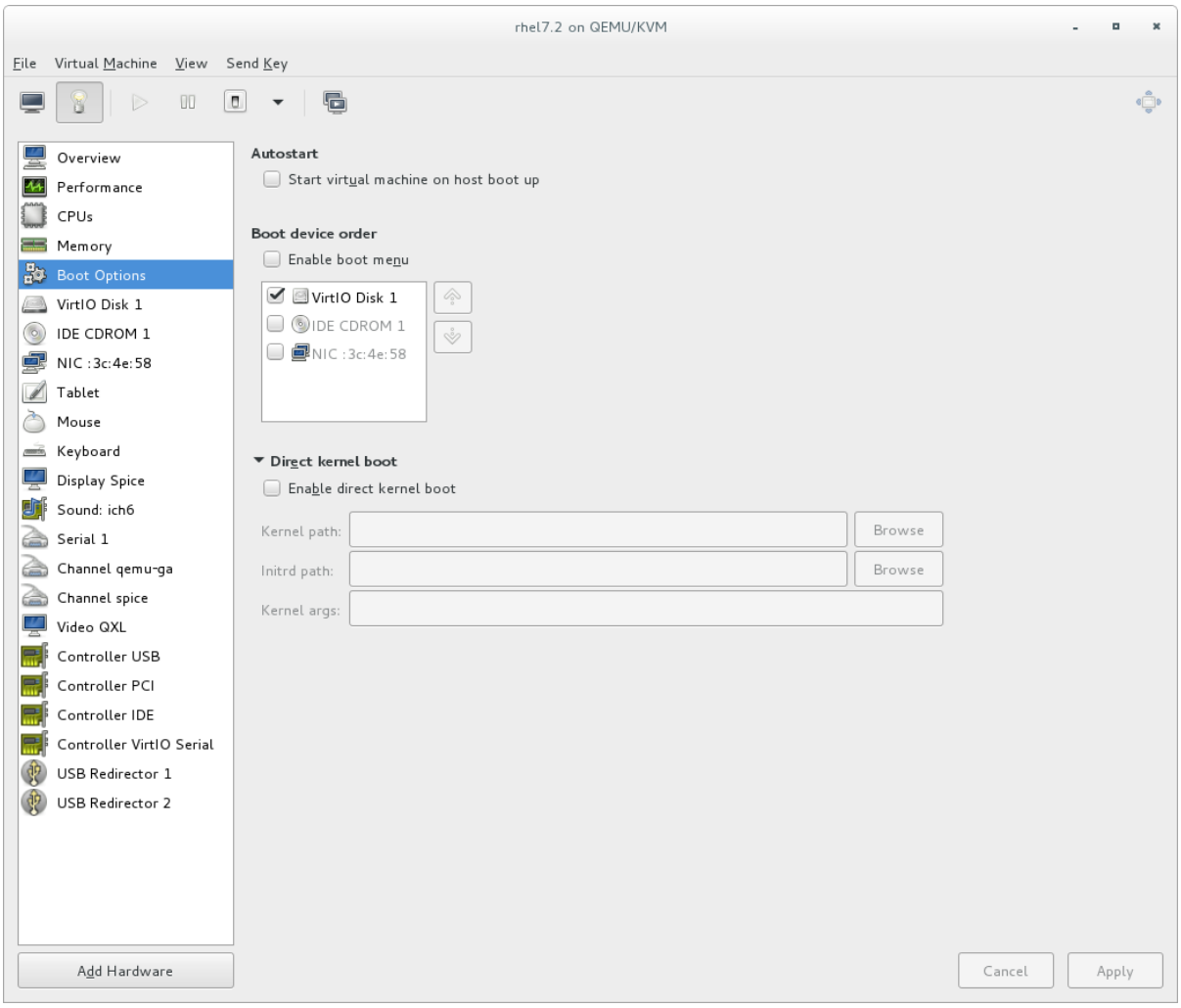
   a. 若要指明此 guest 虚拟机应在每次主机物理计算机启动时启动，请选择 Autostart 复选框。

   b. 要指明应启动 guest 虚拟机的顺序，请单击 Enable boot 菜单 复选框。选中此项后，您可以检查要从引导的设备，并使用箭头键更改客户机虚拟机在引导时将使用的顺序。

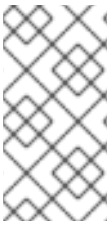
   c. 如果要直接从 Linux 内核引导，展开 Direct 内核 引导菜单。填写您要使用的 内核路径、Initrd 路径 和 内核参数。

3. 点应用。

图 19.5. 配置引导选项



19.3.2. 将 USB 设备附加到虚拟机



注意

要将 USB 设备附加到客户机虚拟机，您必须首先将其附加到主机物理计算机，并确认该设备正常工作。如果客户机正在运行，则需要在继续前将其关闭。

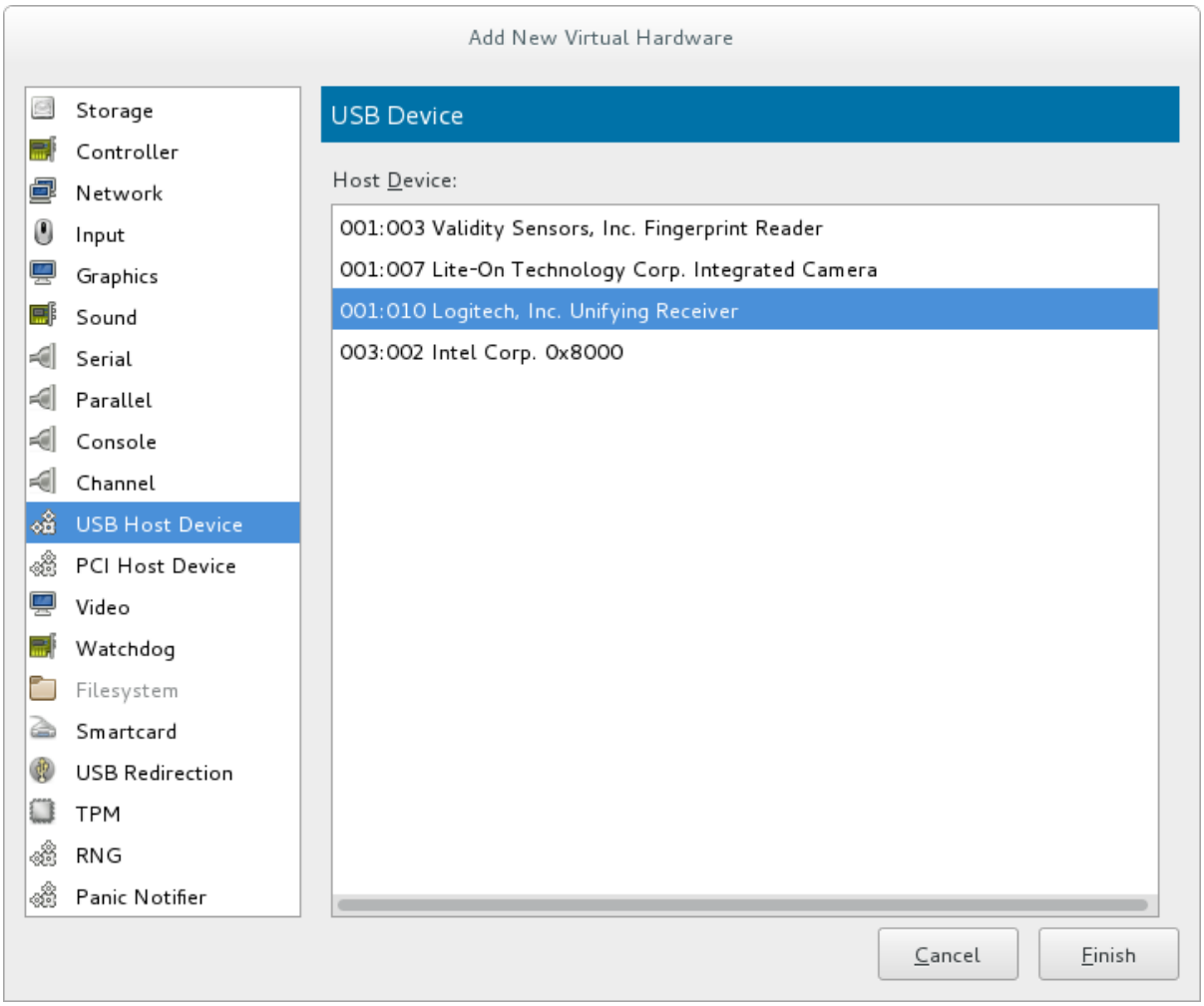
过程 19.2. 使用 Virt-Manager 附加 USB 设备

1. ***打开 guest 虚拟机的 Virtual Machine Details 屏幕。***

2. ***点 Add Hardware***

3. ***在 Add New Virtual Hardware popup 中，选择 USB Host Device，从列表中选择您要附加的设备，然后单击 Finish。***

图 19.6. 添加 USB 设备



4. *要在客户机虚拟机中使用 USB 设备，请启动 guest 虚拟机。*

19.3.3. USB 重定向

如果有一个主机物理机器在数据中心中运行，则最好使用 USB 重定向。用户从本地机器或瘦客户端连接到其/及客户机虚拟机。此本地机器上有 SPICE 客户端。用户可以将任何 USB 设备附加到瘦客户端，SPICE 客户端会将设备重定向到数据中心中的主机物理机器，以便供瘦客户端上运行的虚拟机使用。

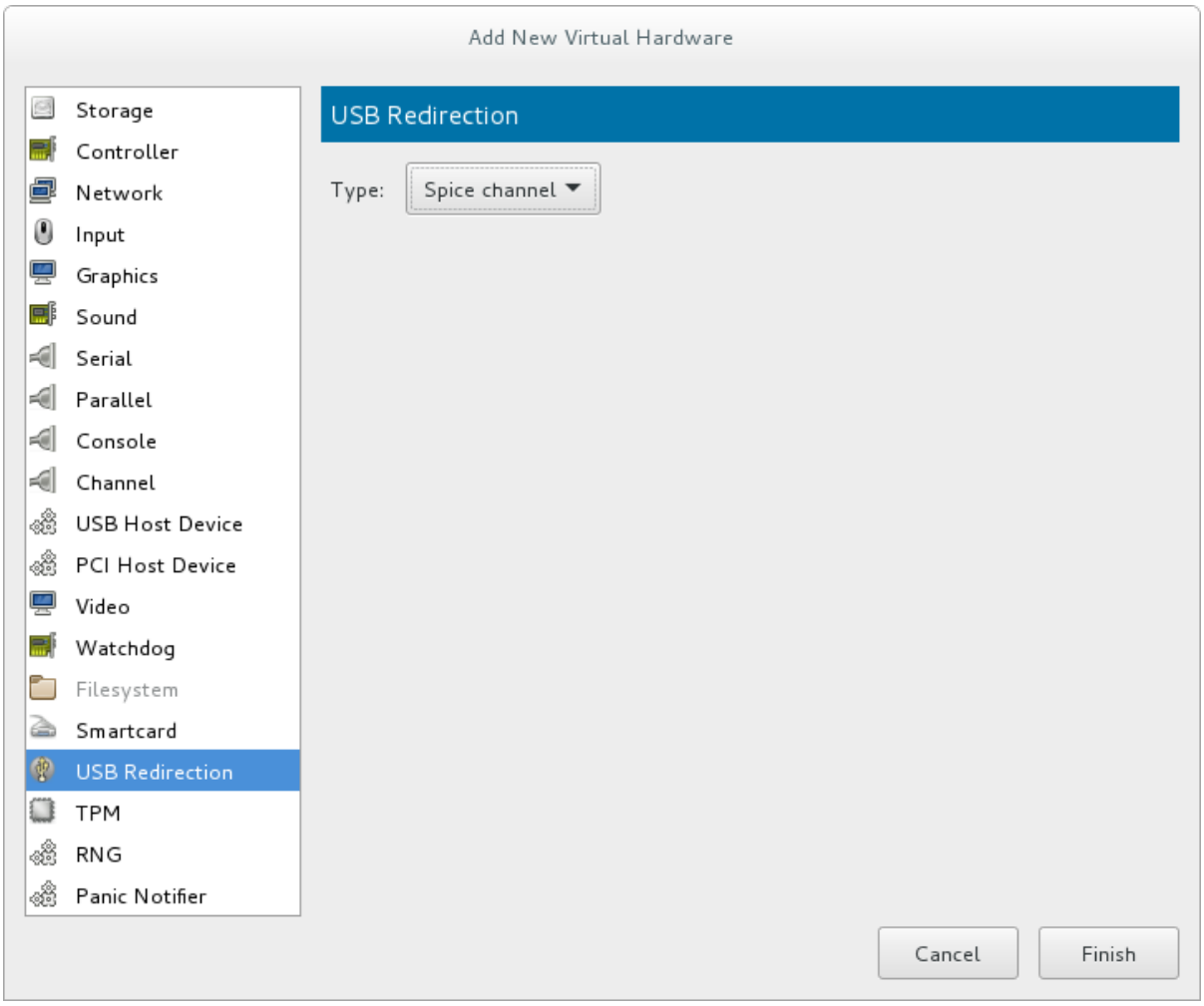
过程 19.3. 重定向 USB 设备

1. *打开 guest 虚拟机的 Virtual Machine Details 屏幕。*

2. *点 Add Hardware*

3. *在 Add New Virtual Hardware 弹出窗口中，选择 USB 重定向。确保从" 类型 "下拉菜单中选择 Spice 频道，并单击" 完成"。*

图 19.7. 添加新虚拟硬件窗口



4. *打开 Virtual Machine 菜单并选择 Redirect USB 设备。此时会打开一个带有 USB 设备列表的弹出窗口。*

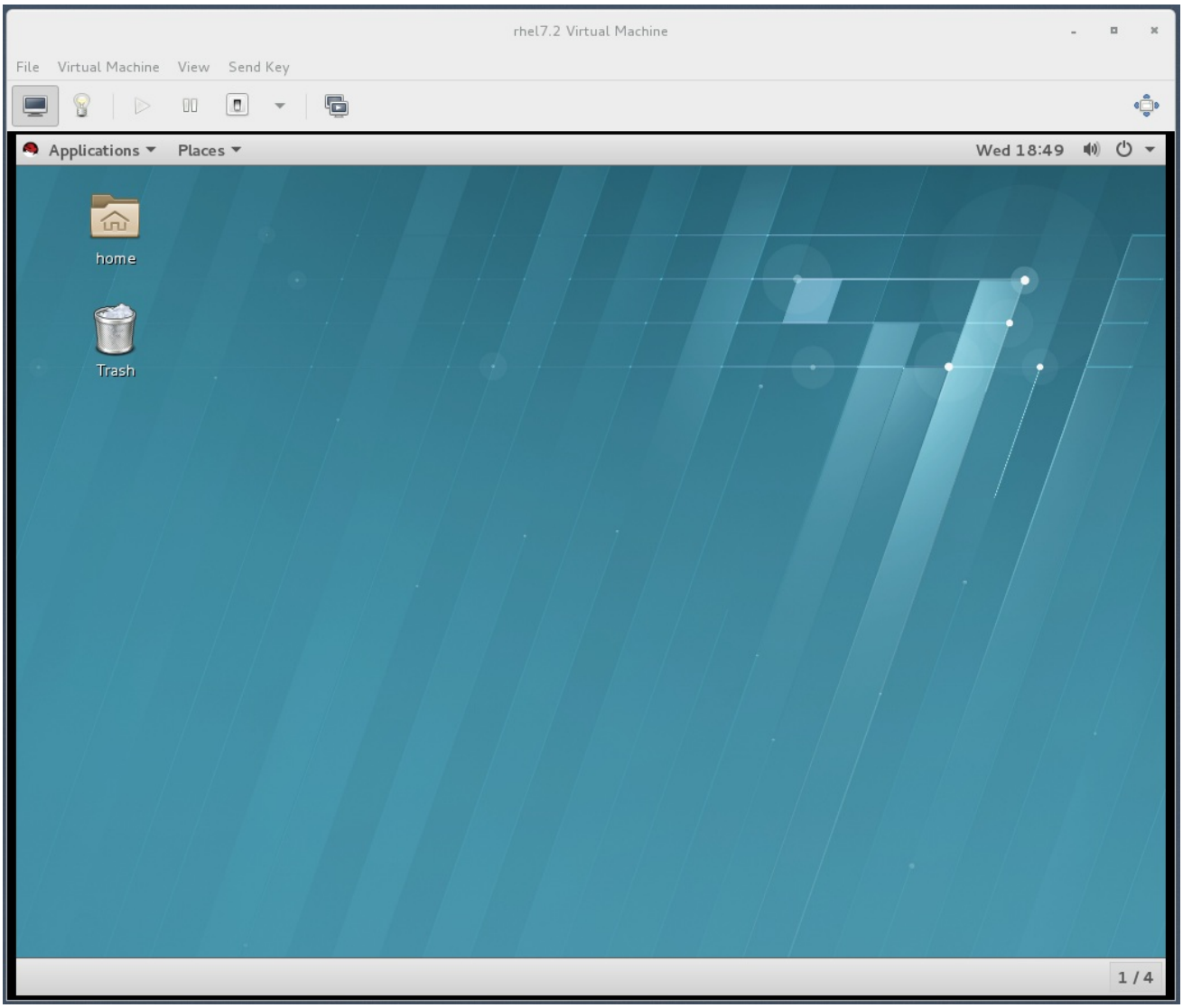
图 19.8. 选择 USB 设备

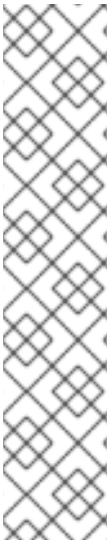
5. *选中其复选框并单击确定，为重定向选择 USB 设备。*

19.4. 虚拟机图形控制台

此窗口显示 guest 的图形控制台。客户机可以使用多种不同的协议导出其图形帧缓冲区： virt-manager 支持 VNC 和 SPICE。如果您的虚拟机设置为需要身份验证，则 Virtual Machine 图形控制台会在显示前提示输入密码。

图 19.9. 图形控制台窗口





注意

VNC 被许多安全专家视为不安全，但进行了一些更改，在 Red Hat Enterprise Linux 上启用了 VNC 的安全使用。客户机计算机仅侦听本地主机的回环地址(127.0.0.1)。这样可保证主机上具有 shell 特权的那些可以通过 VNC 访问 virt-manager 和虚拟机。虽然 virt-manager 配置为监听其他公共网络接口和其它方法，但不建议这样做。

可以通过 SSH 隧道通过 SSH 来执行远程管理，从而对流量进行加密。出于安全原因，可以将 VNC 配置为无需通过 SSH 进行远程访问，但出于安全原因，不建议这样做。要远程管理客户端，请按照 第 18 章 客户机的远程管理 中的说明操作。TLS 可以为管理客户机和主机系统提供企业级安全性。

您的本地桌面可以截获组合键（例如，按 Ctrl+Alt+F1）以防止它们发送到客户机计算机。您可以使用 Send key 菜单选项发送这些序列。在 guest 计算机窗口中，单击 Send key 菜单并选择要发送的键序列。另外，在此菜单中，您还可以捕获屏幕输出。

SPICE 是可用于 Red Hat Enterprise Linux 的 VNC 的替代方案。

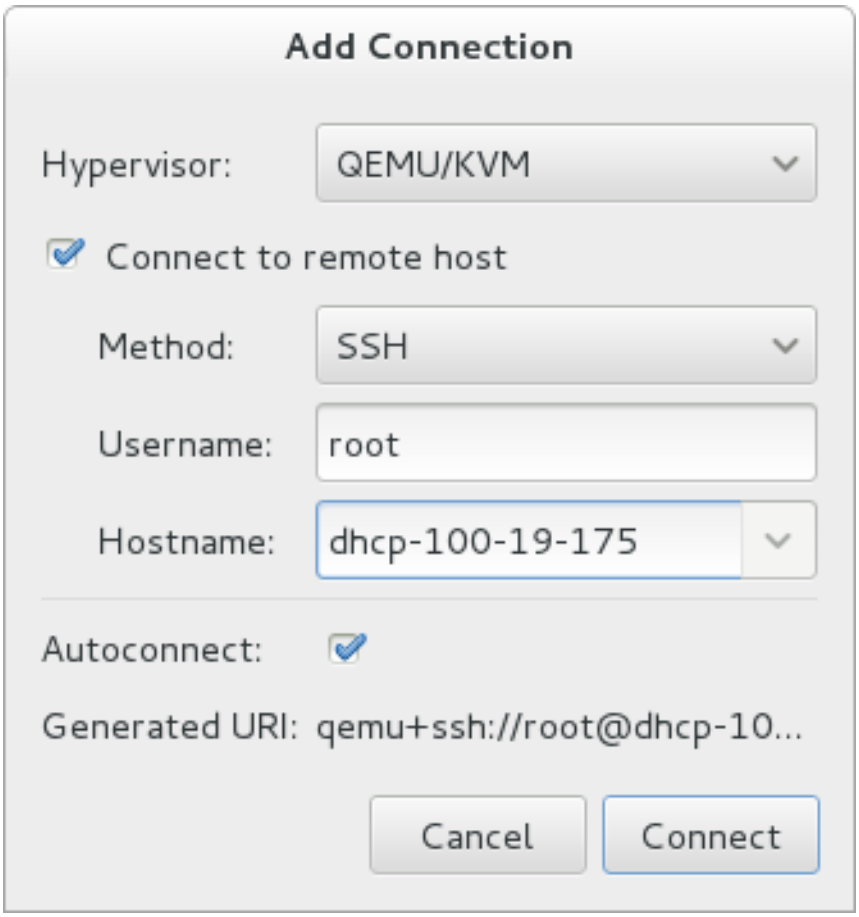
19.5. 添加远程连接

此流程论述了如何使用 virt-manager 设置到远程系统的连接。

1. 若要创建新连接，可打开 File 菜单，再选择 Add Connection 菜单项。

2. 此时会出现 Add Connection 向导。选择系统管理程序。对于 Red Hat Enterprise Linux 7，系统选择 QEMU/KVM。为本地系统选择 Local，或者其中一个的远程连接选项并点 Connect。这个示例使用通过 SSH 进行远程隧道，用于默认安装。有关配置远程连接的详情，请参考 第 18 章 客户机的远程管理

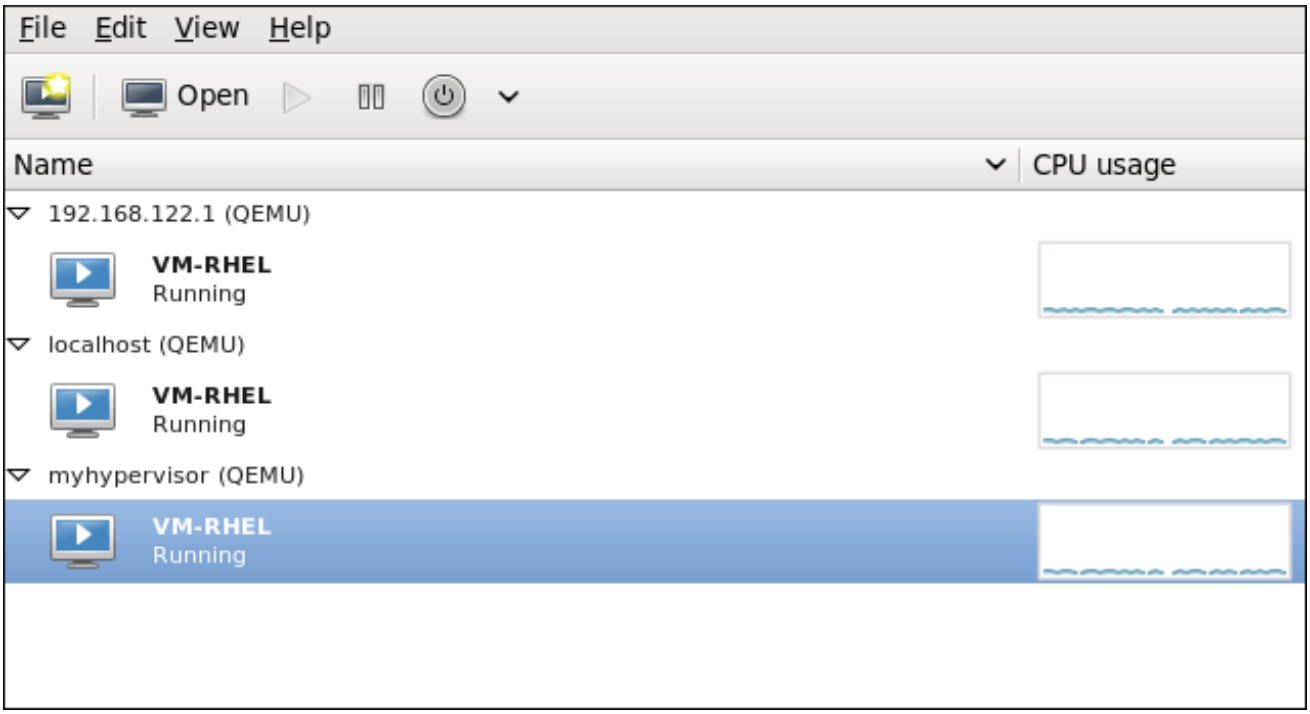
   图 19.10. 添加连接



3. 出现提示时，输入所选主机的 root 密码。

远程主机现已连接，并出现在主 virt-manager 窗口中。

图 19.11. 主 virt-manager 窗口中的远程主机



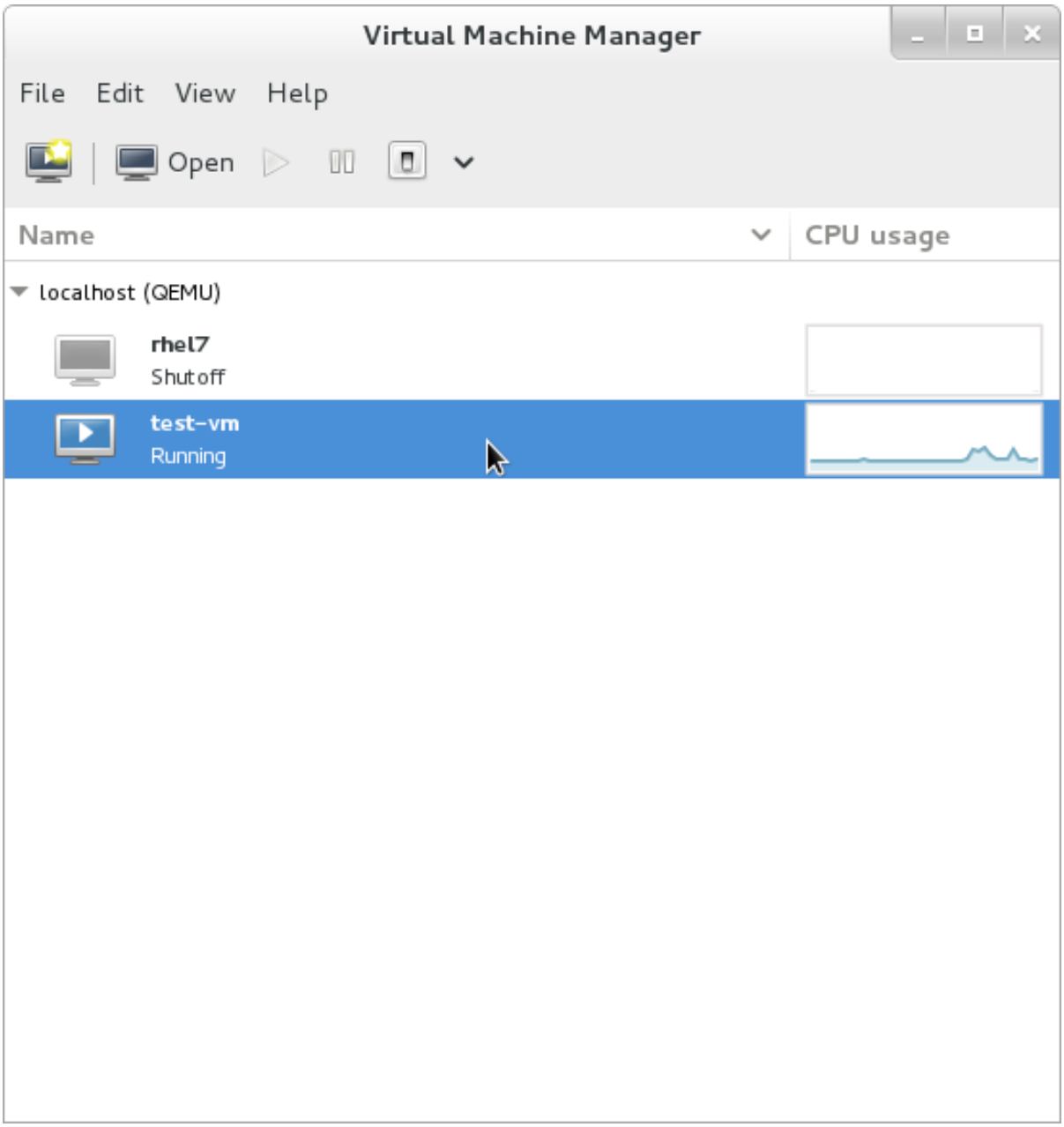
19.6. 显示客户机详情

您可以使用 Virtual Machine Monitor 来查看系统中任何虚拟机的活动信息。

查看虚拟系统的详情：

1. *在 Virtual Machine Manager 主窗口中，突出显示您要查看的虚拟机。*

图 19.12. 选择要显示的虚拟机

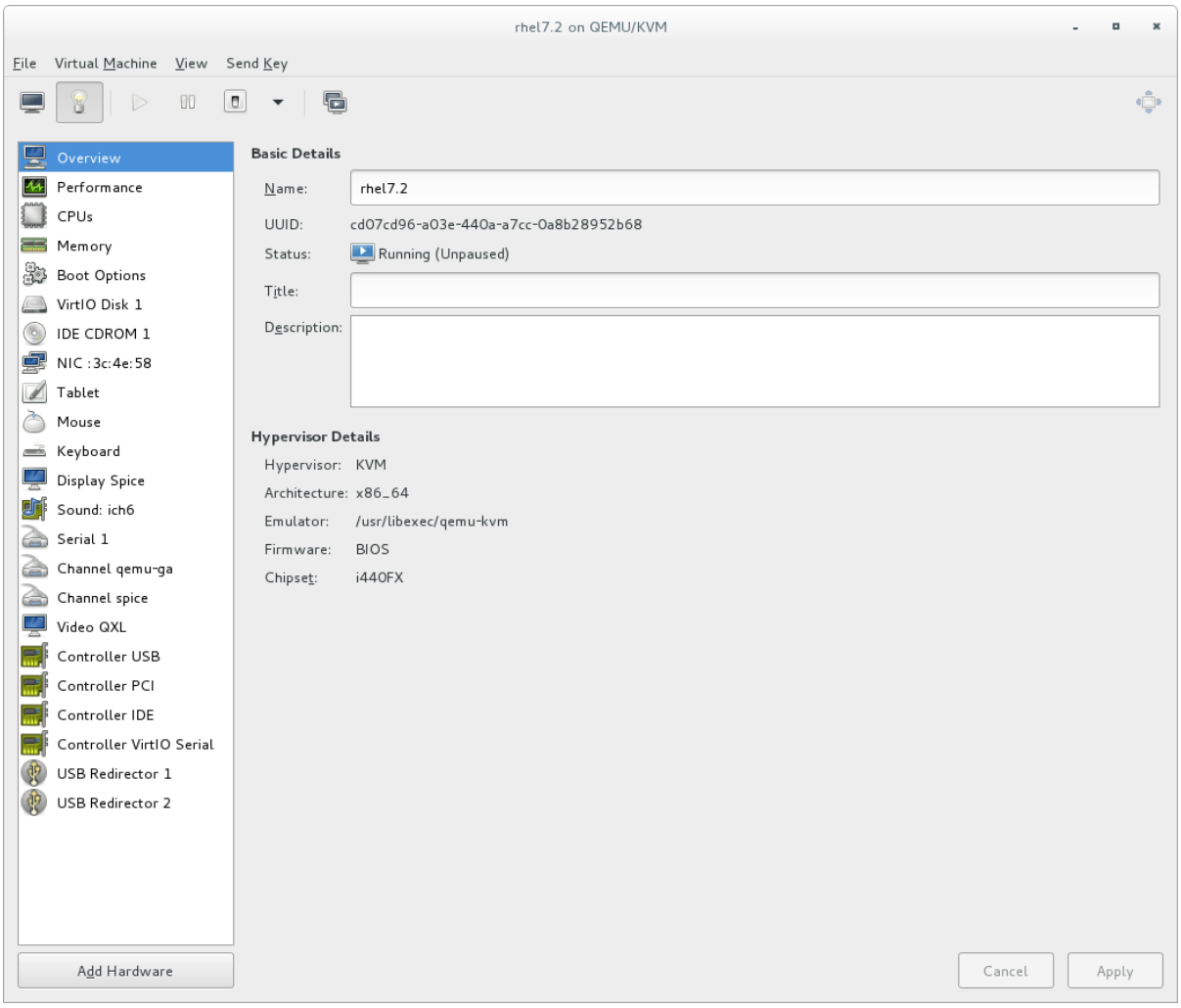


2. *在 Virtual Machine Manager Edit 菜单中，选择 Virtual Machine Details。*

当 Virtual Machine details 窗口打开时，可能会显示控制台。如果发生这种情况，请单击 View，然后选择 Details。Overview 窗口默认打开。要返回此窗口，请从左侧的导航窗格中选择 Overview。

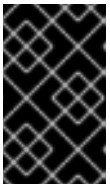
Overview 视图显示客户机的配置详情概述。

图 19.13. 显示客户机详情概述



3. *从左侧的导航窗格中选择 CPU。CPU 视图允许您查看或更改当前的处理器分配。*

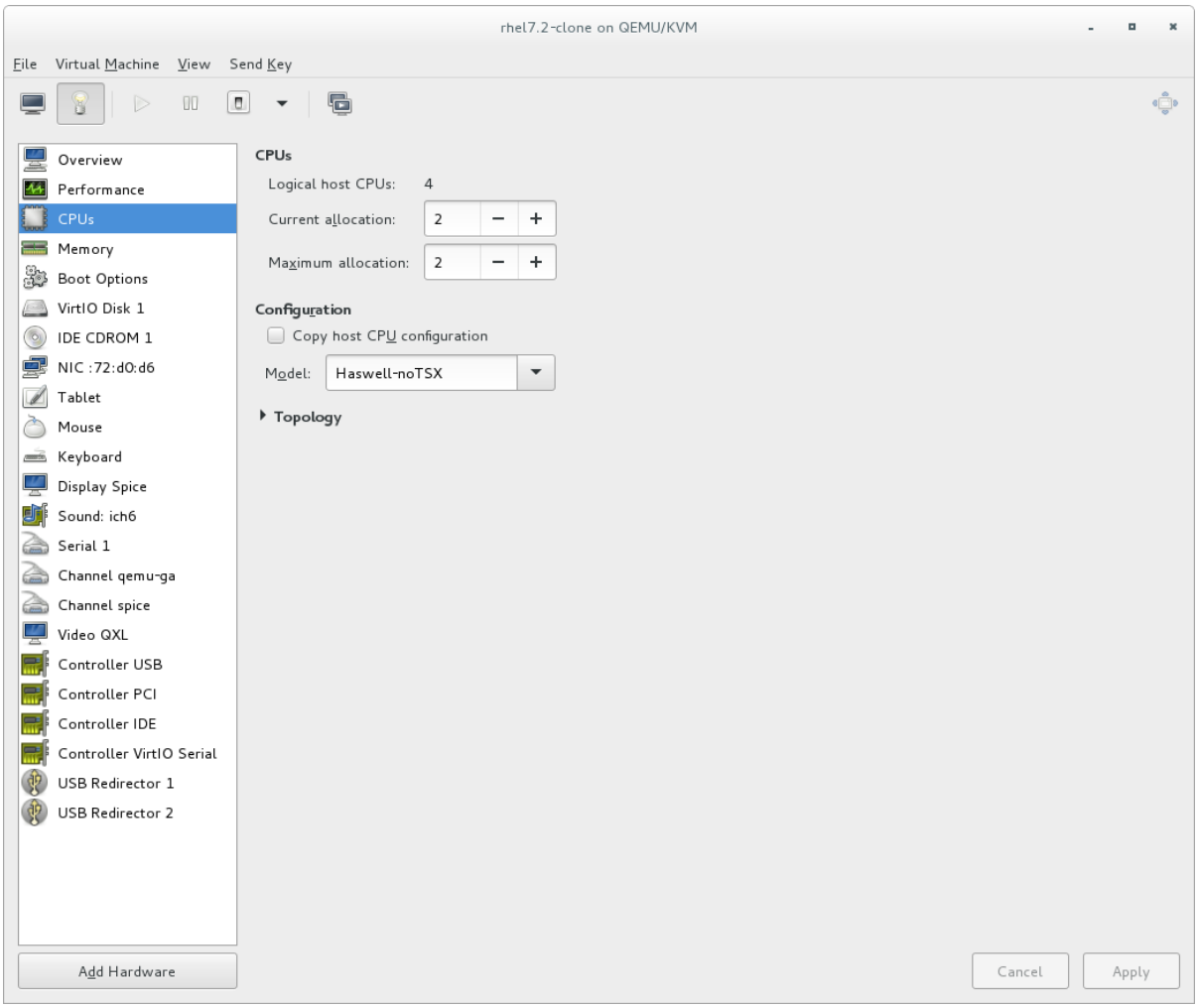
   也可以在虚拟机运行时增加虚拟 CPU(vCPU)的数量，这被称为 热插拔。



重要

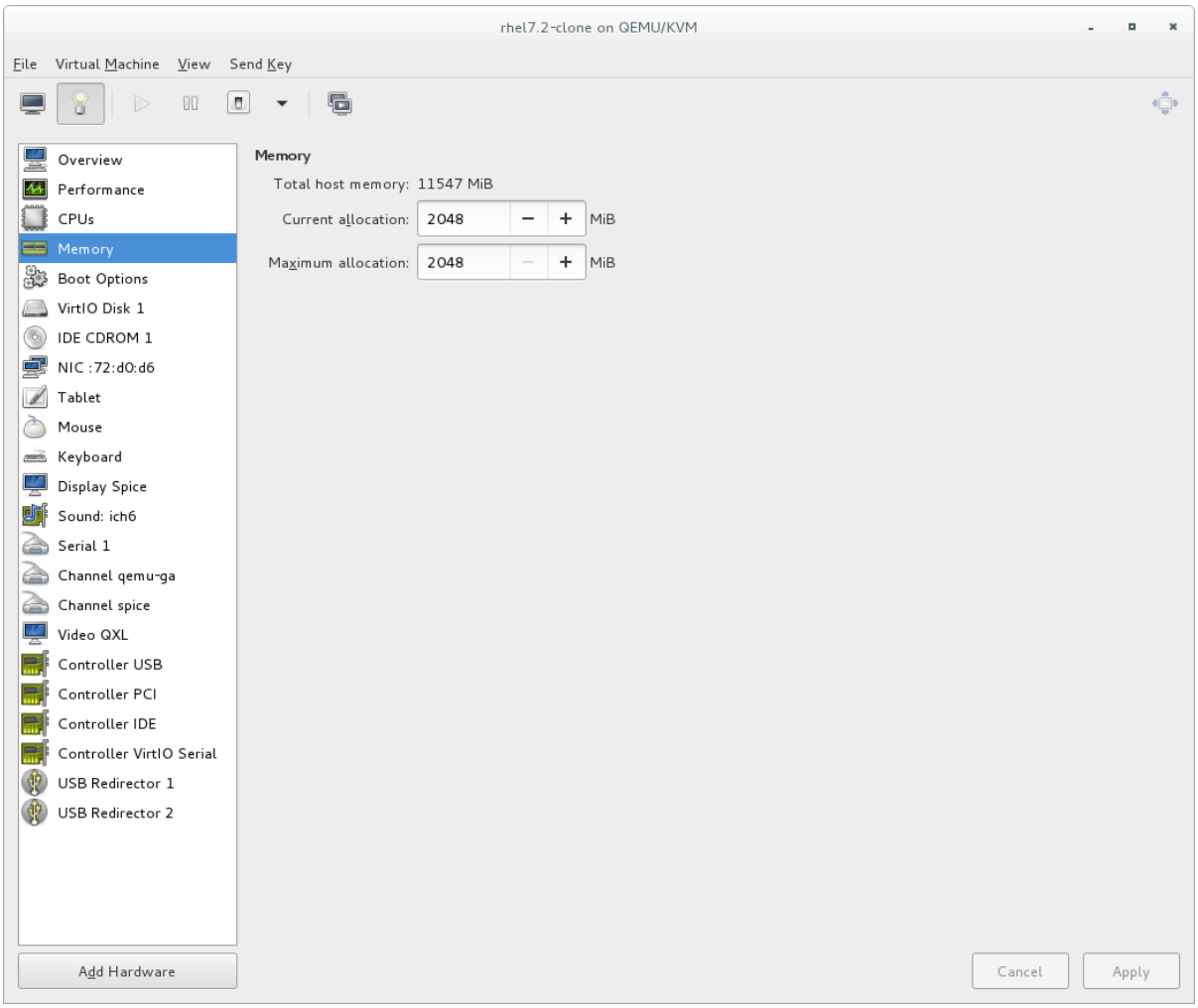
Red Hat Enterprise Linux 7 不支持 热拔 vCPU。

图 19.14. 处理器分配面板



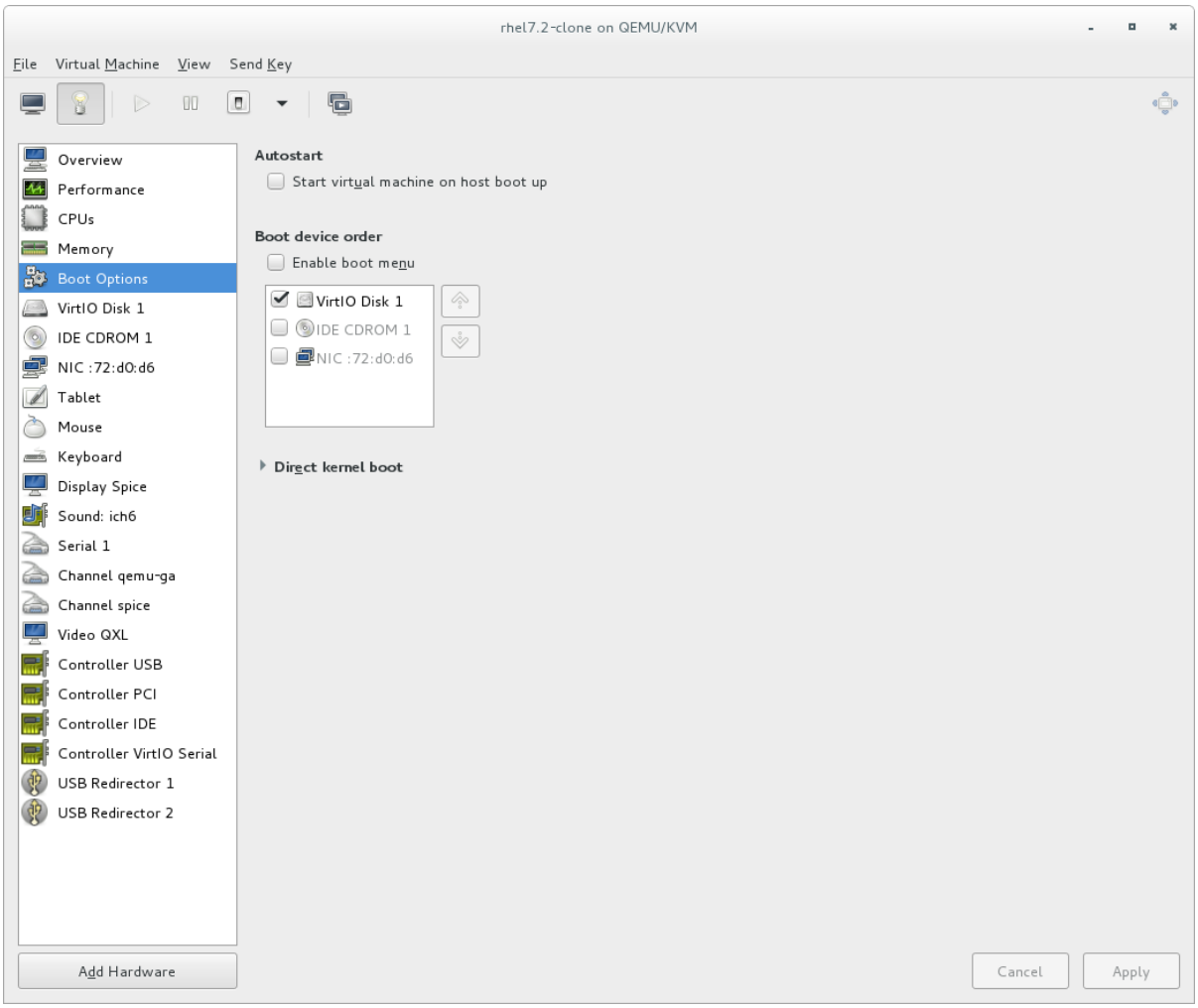
4. *从左侧的导航窗格中，选择 Memory。Memory 视图允许您查看或更改当前的内存分配。*

图 19.15. 显示内存分配



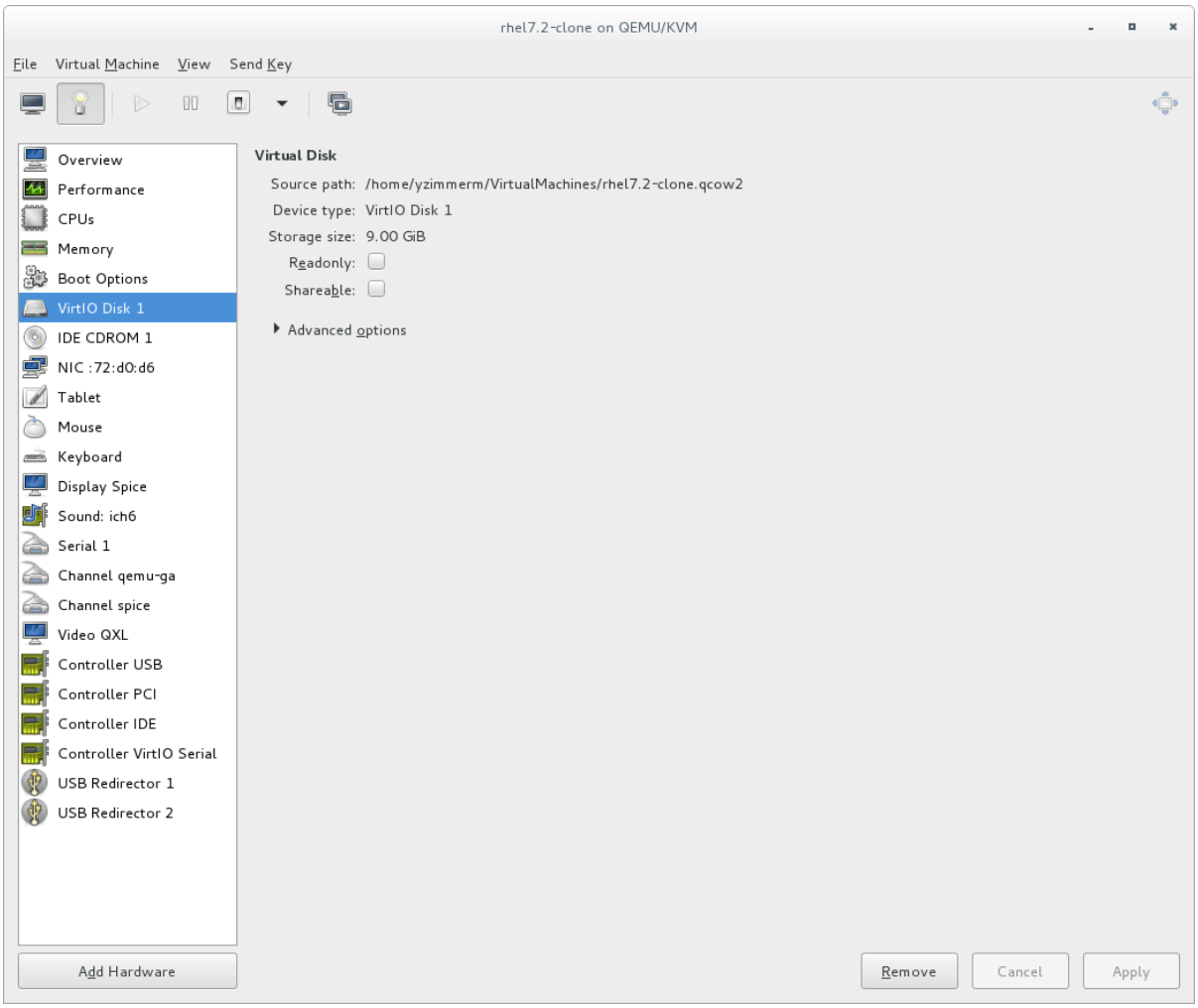
5. ***从左侧的导航窗格中，选择"引导选项"。通过 Boot Options 视图，您可以查看或更改引导选项，包括虚拟机是否在主机引导时启动，以及虚拟机的引导设备顺序。***

图 19.16. 显示引导选项



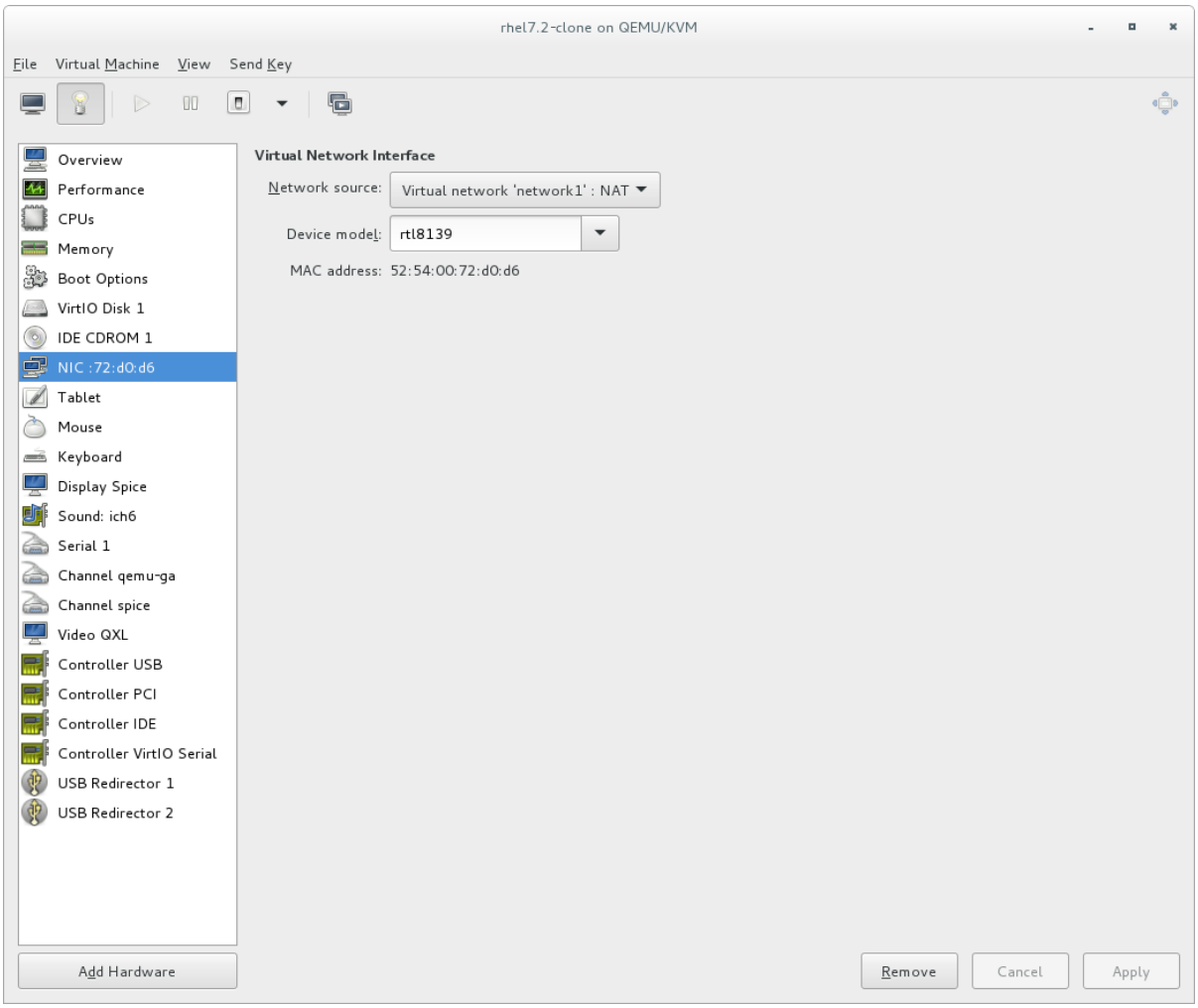
6. *在导航窗格中显示附加到虚拟机的每个虚拟磁盘。点击虚拟磁盘来修改或删除它。*

图 19.17. 显示磁盘配置



7. *在导航窗格中显示附加到虚拟机的每个虚拟网络接口。点击虚拟网络接口修改或删除它。*

图 19.18. 显示网络配置



19.7. 管理快照

使用 virt-manager，可以创建、运行和删除客户机 快照。快照是 guest 硬盘、内存和设备状态在单一时间点的保存映像。快照创建之后，可以随时将 guest 返回到快照的配置。

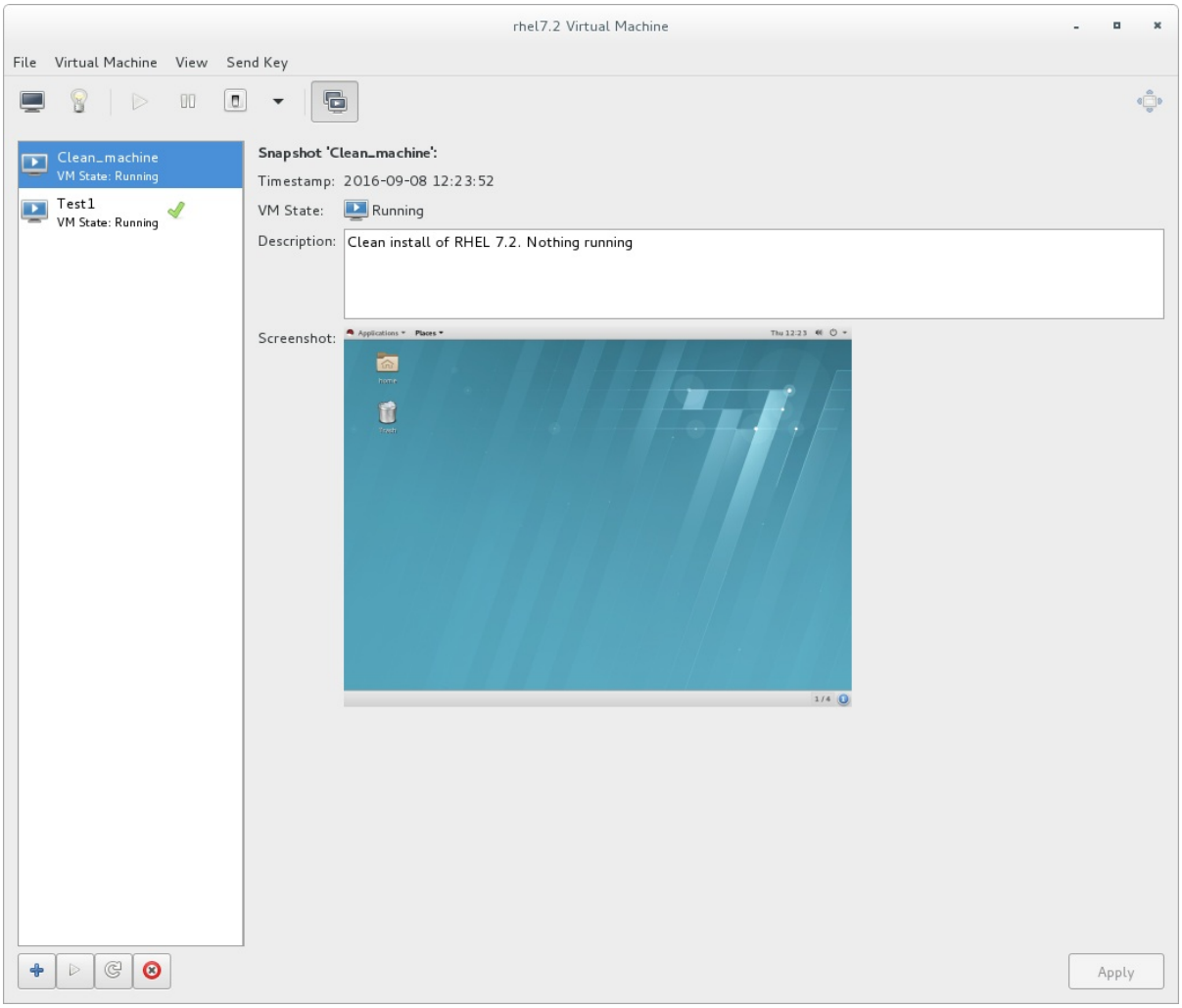
重要

红帽建议使用外部快照，因为当由其他虚拟化工具处理时，它们更灵活且更可靠。但是，目前无法在 virt-manager 中创建外部快照。

要创建外部快照，请使用 virsh snapshot-create-as 命令和 --diskspec vda,snapshot=external 选项。如需更多信息，请参阅 第 A.13 节 “使用 libvirt 创建外部快照的临时解决方案”。

- *要在 virt-manager 中管理快照，请点击客户端控制台中的*

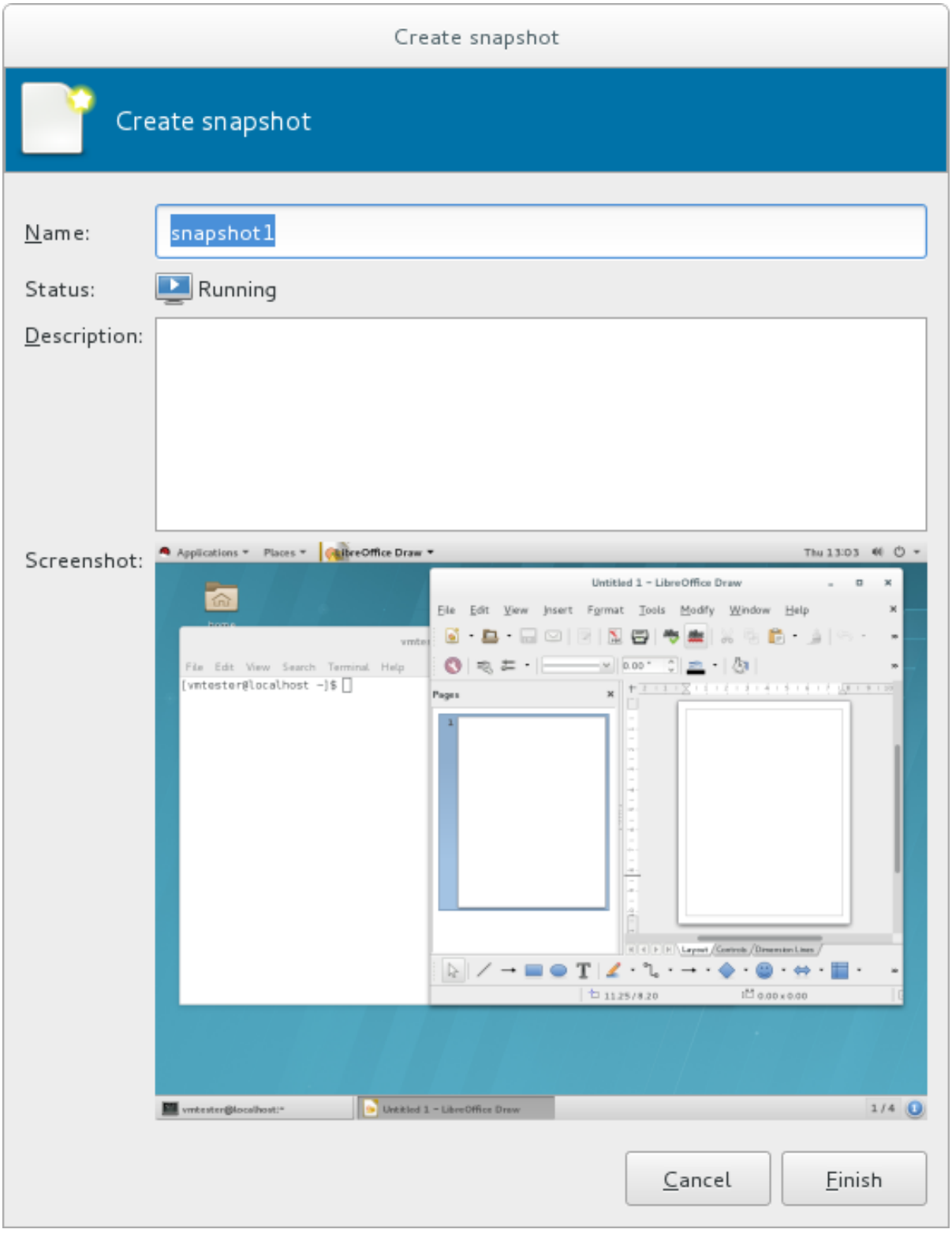
打开快照管理界面。



- *要创建新快照，请点击快照列表下的*

  。在快照创建界面中，输入快照的名称，以及可选的描述，然后单击" 完成"。



- ***要将客户机恢复到快照的配置，*** *请选择**快照并**单击*

- ***要删除所选快照，点***

警告

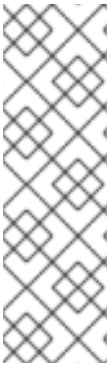
在虚拟机正在运行时创建和加载快照（也称为 实时快照）仅支持 qcow2 磁盘映像。

要更加深入的快照管理，请使用 virsh snapshot-create 命令。有关使用 virsh 管理快照的详情，请参考 第 20.39 节 “管理快照”。

第 20 章 使用 VIRSH 管理客户机虚拟机

virsh 是用于管理客户机虚拟机的命令行界面工具，并可作为在 Red Hat Enterprise Linux 7 中控制虚拟化的主要方法。virsh 命令行工具在 libvirt 管理 API 上构建，可用于创建、部署和管理客户机虚拟机。virsh 实用程序是创建虚拟化管理脚本的理想选择，并且没有 root 特权的用户可在只读模式中使用它。virsh 软件包使用 yum 作为 libvirt-client 软件包的一部分安装。

有关安装说明，请参阅 第 2.2.1 节 “手动安装虚拟化软件包”。有关 virsh 的一般介绍，包括实际演示，请参阅 虚拟化入门指南 一章的剩余部分涵盖了基于逻辑顺序的 virsh 命令集。



注意

请注意，在使用帮助或读取 man page 时，将使用术语"domain" 而不是术语 guest 虚拟机。这是 libvirt 使用这一术语。如果显示屏幕输出以及使用 'domain' 的提示，则不会切换到 guest 或 guest 虚拟机。在所有示例中，将使用 guest 虚拟机 'guest1'。所有情况下，都应用 guest 虚拟机的名称替换它。当为客户机虚拟机创建名称时，您应该使用简短的方法来记住整数(0,1,2.)）、文本字符串名称，或者在所有情况下，您也可以使用虚拟机的完整 UUID。

重要

请注意您使用的用户很重要。如果您使用一个用户创建客户机虚拟机，您将无法使用另一个用户检索有关它的信息。这在 virt-manager 中创建虚拟机时尤其重要。在这种情况下，默认用户为 root，除非另外指定。如果您无法使用 virsh list --all 命令列出虚拟机，则很可能是因为使用与您用于创建虚拟机的不同用户运行该命令。如需更多信息，请参阅 重要。

20.1. 虚拟机状态和类型

几个 virsh 命令受到客户机虚拟机状态的影响：

- 瞬态 - 临时客户机无法重新引导.
- persistent - 持久的 guest 虚拟机会重新引导，并在删除前最后。

在虚拟机生命周期中，libvirt 将把客户机指定为以下状态之一：

-

undefined - 这是尚未定义或创建的客户机虚拟机。因此，libvirt 不了解此状态内的任何 guest，不会报告此状态下的客户机虚拟机。

- **关闭 - 这是一个已定义但未在运行的客户机虚拟机。只有持久的客户机可以考虑关闭。因此，当临时的 guest 虚拟机进入这个状态时，它就会存在。**

- **Running - 此状态下的客户机虚拟机已定义并且正在工作。此状态可用于持久和临时的客户机虚拟机。**

- **paused - 已暂停系统管理程序上的 guest 虚拟机执行，或者其状态已暂时存储，直到恢复为止。处于此状态的 guest 虚拟机不知道它们已被暂停，请注意在恢复后不会发现经过的时间。**

- **saved - 此状态与暂停状态类似，但客户机虚拟机的配置被保存到持久性存储。任何处于这个状态的客户机虚拟机都不知道它已被暂停，且不会注意到恢复后经过的时间。**

20.2. 显示 VIRSH 版本

virsh version 命令显示当前 libvirt 版本并显示有关本地 virsh 客户端的信息。例如：

```
$ virsh version
Compiled against library: libvirt 1.2.8
Using library: libvirt 1.2.8
Using API: QEMU 1.2.8
Running hypervisor: QEMU 1.5.3
```

virsh 版本 --daemon 可以用来获取有关 libvirtd 版本和软件包信息的信息，包括有关主机上运行的 libvirt 守护进程的信息。

```
$ virsh version --daemon
Compiled against library: libvirt 1.2.8
Using library: libvirt 1.2.8
Using API: QEMU 1.2.8
Running hypervisor: QEMU 1.5.3
Running against daemon: 1.2.8
```

20.3. 使用 ECHO 发送命令

virsh echo [-shell][--xml] 参数命令以指定格式显示 指定的参数。您可以使用的格式是 --shell 和 --xml。查询的每个参数都以空格分开。--shell 选项生成输出（如果需要使用单引号），因此它适合将其作

为命令复制和粘贴至 bash 模式。如果使用 --xml 参数，则会在 XML 文件中格式化输出以供使用，然后保存或用于客户机配置。

20.4. 使用 VIRSH CONNECT 连接到管理程序

virsh connect [hostname-or-URI] [--readonly] 命令开始使用 virsh 的本地管理程序会话。第一次运行此命令后，每次 virsh shell 运行时将自动运行该命令。虚拟机监控程序连接 URI 指定如何连接到管理程序。最常用的 URI 是：

- QEMU:///system - 作为 root 用户在本地连接到 KVM 管理程序上客户机虚拟机的守护进程。
- QEMU:///session - 以用户本地方式连接到使用 KVM 管理程序的用户本地机器的集合。
- lxc:/// - 连接到本地 Linux 容器。

可以使用以下方式运行该命令，目标 guest 通过其机器名称（主机名）或虚拟机监控程序的 URL（virsh uri 命令的输出）指定，如下所示：

```
$ virsh uri
qemu:///session
```

例如，要建立一个会话以连接到一组客户机虚拟机，以本地用户身份为您提供：

```
$ virsh connect qemu:///session
```

要启动只读连接，请使用 --readonly 附加上述命令。有关 URI 的更多信息，请参阅 远程 URI。如果您不确定 URI，virsh uri 命令会显示它：

20.5. 显示客户机虚拟机和虚拟机监控程序的信息

virsh list 命令将列出连接到您的管理程序（适合请求的搜索参数）的客户机虚拟机。 命令的输出在表中有 3 列。每个 guest 虚拟机均列出其 ID、名称和状态。

有多种搜索参数可用于 virsh list。这些选项位于 man page 中，运行 man virsh 或运行 virsh list --help 命令。

注意

请注意，如果此命令仅显示由 root 用户创建的 guest 虚拟机。如果没有显示您创建的虚拟机，则不能以 root 身份创建虚拟机。

使用 virt-manager 界面创建的客户机默认由 root 创建。

例 20.1. 如何列出所有本地连接的虚拟机

以下示例列出了您的虚拟机监控程序连接到的所有虚拟机。请注意，这个命令列出了持久性和 临时 虚拟机。

```
# virsh list --all

 Id          Name                   State
----------------------------------------------------
 8  guest1   running
 22  guest2   paused
 35  guest3   shut off
 38           guest4   shut off
```

例 20.2. 如何列出不活跃的虚拟机

以下示例列出了当前不活跃或没有运行的客户机。请注意，该列表仅包含永久虚拟机。

```
# virsh list --inactive

 Id          Name                   State
----------------------------------------------------
 35  guest3   shut off
 38  guest4   shut off
```

另外，使用以下命令也可以用来显示有关管理程序的基本信息：

- **# virsh hostname** - 显示 hypervisor 的主机名，例如：

```
# virsh hostname
dhcp-2-157.eus.myhost.com
```

- **# virsh sysinfo - 显示 hypervisor 系统信息的 XML 表示，例如：**

```
# virsh sysinfo
<sysinfo type='smbios'>
  <bios>
    <entry name='vendor'>LENOVO</entry>
    <entry name='version'>GJET71WW (2.21 )</entry>
[...]
```

20.6. 启动、恢复和恢复虚拟机

20.6.1. 启动客户机虚拟机

virsh start domain; [--console] [--paused] [--autodestroy] [--bypass-cache] [--force-boot] 命令启动一个已定义的非活动虚拟机，但状态自上次受管或新启动以来处于不活跃状态。默认情况下，如果域由 **virsh managedsave** 命令保存，则域将恢复到之前的状态。否则，它将被引导。该命令可采用以下参数，且需要虚拟机的名称。

- **--console - 将运行 virsh 的终端连接到域的控制台设备。这是运行级别 3。**
- **--paused - 如果驱动程序支持，它将以暂停状态启动客户机虚拟机**
- **--autodestroy - 在 virsh 断开连接时自动销毁客户机虚拟机**
- **--bypass-cache - 如果客户机虚拟机处于 managedsave，则使用**
- **--force-boot - 丢弃任何 受管保存 选项，并导致新启动发生**

例 20.3. 如何启动虚拟机

以下示例启动您已创建的 **guest1** 虚拟机，并且当前处于 **inactive** 状态。另外，命令会将客户机的控制台附加到运行 **virsh** 的终端：

```
# virsh start guest1 --console
Domain guest1 started
Connected to domain guest1
Escape character is ^]
```

–

20.6.2. 将虚拟机配置为在引导时自动启动

当主机引导时，virsh autostart [--disable] domain 命令将自动启动 guest 虚拟机。在这个命令中添加 --disable 参数可禁用 autostart。在这种情况下，主机物理机引导时不会自动启动 guest。

> **例 20.4. 主机物理机启动时，如何自动启动虚拟机**
>
> 以下示例将您创建 guest1 虚拟机设置为在主机引导时自动启动：
>
> ```
> # virsh autostart guest1
> ```

20.6.3. 重新引导客户机虚拟机

使用 virsh reboot 域 [-mode modename] 命令重新引导 guest 虚拟机。请记住，该操作将仅在执行重新启动后返回，因此可能从该点清除一次，直到 guest 虚拟机实际重新启动为止。您可以通过修改 guest 虚拟机的 XML 配置文件中的 on_reboot 元素来控制重新引导客户机虚拟机的行为。默认情况下，虚拟机监控程序会尝试自动选择合适的关闭方法。要指定替代方法，-- mode 参数可以指定以逗号分隔的列表，其中包括 acpi 和 agent。驱动将尝试每个模式的顺序是未定义的，并且与 virsh 中指定的顺序无关。若要严格控制排序，请一次使用单一模式并重复该命令。

> **例 20.5. 如何重新引导客户机虚拟机**
>
> 以下示例重新引导名为 guest1 的客户机虚拟机。在这个示例中，重启使用 initctl 方法，但您可以选择任何适合您的需要的模式。
>
> ```
> # virsh reboot guest1 --mode initctl
> ```

20.6.4. 恢复客户机虚拟机

virsh restore <file> [--bypass-cache] [--xml /path/to/file] [--running] [--paused] 命令会恢复之前通过 virsh save 命令保存的客户机虚拟机。有关 virsh save 命令的详情请参考 第 20.7.1 节 “保存客户机虚拟机的配置”。恢复操作重启保存的客户机虚拟机，可能需要一些时间。客户机虚拟机的名称和 UUID 被保留，但 ID 不一定与保存虚拟机时具有的 ID。

virsh restore 命令可使用以下参数：

- --bypass-cache - 导致恢复以避免文件系统缓存，但请注意，使用这个标记可能会减慢恢复

操作的速度。

- -- XML - 这个参数必须与 XML 文件名一起使用。虽然通常省略这个参数，但它可以提供备用的 XML 文件，以便在恢复的客户机虚拟机中使用，且仅在域 XML 中更改主机特定部分。例如，由于保存客户机后执行的磁盘快照，它可用于考虑底层存储中的文件命名差异。

- --running - 覆盖在保存镜像中记录的状态，以便启动客户机虚拟机正在运行。

- --paused - 覆盖在保存镜像中记录的状态，以便启动客户机虚拟机暂停。

> 例 20.6. 如何恢复客户机虚拟机
>
> 以下示例恢复客户机虚拟机及其运行的配置文件 guest1-config.xml:
>
> ```
> # virsh restore guest1-config.xml --running
> ```

20.6.5. 恢复客户机虚拟机

virsh 恢复 域 命令重新启动已暂停的域的 CPU。此操作是即时的。客户机虚拟机从暂停的时间点恢复执行。请注意，此操作不会恢复已未定义的客户机虚拟机。此操作不会恢复 临时 虚拟机，且仅在持久的虚拟机中工作。

> 例 20.7. 如何恢复已暂停的客户机虚拟机
>
> 以下示例恢复 guest1 虚拟机：
>
> ```
> # virsh resume guest1
> ```

20.7. 管理虚拟机配置

本节提供有关管理虚拟机配置的信息。

20.7.1. 保存客户机虚拟机的配置

virsh save [--bypass-cache] 域 文件 [--xml 字符串] [--running] [--paused] [--verbose] 命令会停止

指定域，将 guest 虚拟机的系统内存的当前状态保存到指定文件中。这可能需要大量时间，具体取决于客户机虚拟机使用的内存量。您可以使用 virsh restore (第 20.6.4 节 “恢复客户机虚拟机”)命令恢复客户机虚拟机的状态。

virsh save 命令和 virsh suspend 命令之间的区别是 virsh suspend 会停止域 CPU，但让域的 qemu 进程运行及其内存映像驻留在主机系统中。如果主机系统重启，这个内存镜像将会丢失。

virsh save 命令将域的状态存储在主机系统的硬盘中，终止 qemu 进程。这可让从保存的状态重启域。

您可以使用 virsh domjobinfo 命令监控 virsh save 的过程，并使用 virsh domjobabort 命令取消它。

virsh save 命令可使用以下参数：

- --bypass-cache - 导致恢复以避免文件系统缓存，但请注意，使用这个标记可能会减慢恢复操作的速度。
- -- XML - 这个参数必须与 XML 文件名一起使用。虽然通常省略这个参数，但它可以提供备用的 XML 文件，以便在恢复的客户机虚拟机中使用，且仅在域 XML 中更改主机特定部分。例如，由于保存客户机后执行的磁盘快照，它可用于考虑底层存储中的文件命名差异。
- --running - 覆盖在保存镜像中记录的状态，以便启动客户机虚拟机正在运行。
- --paused - 覆盖在保存镜像中记录的状态，以便启动客户机虚拟机暂停。
- --verbose - 显示保存的进度。

例 20.8. 如何保存运行配置的客户机虚拟机

以下示例将 guest1 虚拟机的运行配置保存到 guest1-config.xml 文件中：

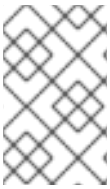
```
# virsh save guest1 guest1-config.xml --running
```

20.7.2. 使用 XML 文件定义客户机虚拟机

virsh 定义 filename 命令可从 XML 文件定义 guest 虚拟机。此例中的客户机虚拟机定义是注册但未启动。如果 guest 虚拟机已在运行，则更改将在域关闭并重新启动后生效。

> **例 20.9. 如何从 XML 文件创建客户机虚拟机**
>
> 以下示例从预先存在的 guest1-config.xml XML 文件创建一个虚拟机，其中包含虚拟机的配置：
>
> ```
> # virsh define guest1-config.xml
> ```

20.7.3. 更新用于恢复客户机虚拟机的 XML 文件



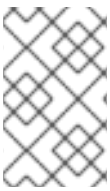
注意

这个命令只用于恢复 guest 虚拟机未正确运行的情况。它并不适用于一般用途。

virsh save-image-define filename [--xml /path/to/file] [--running] [--paused] 命令会更新 guest 虚拟机的 XML 文件，该文件将在 virsh restore 命令中恢复时使用。--xml 参数必须是 XML 文件名，其中包含客户机虚拟机 XML 的替代 XML 元素。例如，它可用于考虑文件命名差别，从而在保存客户机后创建底层存储的磁盘快照。如果 guest 虚拟机应该恢复到正在运行或暂停的状态，则保存镜像记录。使用参数 --running 或 --paused 会指示要使用的状态。

> **例 20.10. 如何保存客户机虚拟机的运行配置**
>
> 以下示例使用相应运行的客户机状态更新 guest1-config.xml 配置文件：
>
> ```
> # virsh save-image-define guest1-config.xml --running
> ```

20.7.4. 提取客户机虚拟机 XML 文件



注意

这个命令只用于恢复 guest 虚拟机未正确运行的情况。它并不适用于一般用途。

virsh save-image-dumpxml 文件 --security-info 命令将提取在保存的状态文件（在 virsh save 命

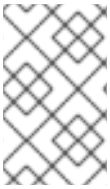
令中使用）时生效的客户机虚拟机 XML 文件。使用 --security-info 参数会在文件中包含安全敏感信息。

例 20.11. 如何从最后保存拉取 XML 配置

以下示例会触发最后一次 保存 客户机虚拟机的配置文件的转储。在本例中，生成的转储文件名为 guest1-config-xml ：

```
# virsh save-image-dumpxml guest1-config.xml
```

20.7.5. 编辑客户机虚拟机配置



注意

这个命令只用于恢复 guest 虚拟机未正确运行的情况。它并不适用于一般用途。

virsh save-image-edit <file> [--running] [--paused] 命令编辑由 virsh save 命令创建的 XML 配置文件。有关 virsh save 命令的详情请参考 第 20.7.1 节 “保存客户机虚拟机的配置”。

保存客户端虚拟机时，生成的镜像文件将指示虚拟机是否应恢复到 --running 或 --paused 状态。如果在 save-image-edit 命令中不使用这些参数，则状态由镜像文件本身决定。通过选择 --running （选择 running 状态）或 --paused （选择暂停状态），您可以覆盖 virsh restore 应使用的状态。

例 20.12. 如何编辑客户机虚拟机的配置并将虚拟机恢复到运行状态

以下示例打开名为 guest1-config.xml 的客户机虚拟机配置文件，以便在默认编辑器中进行编辑。保存编辑后，虚拟机将使用新设置引导：

```
# virsh save-image-edit guest1-config.xml --running
```

20.8. 关闭、关闭、重新启动和关闭客户机虚拟机的关闭

20.8.1. 关闭客户机虚拟机

virsh shutdown domain [--mode modename] 命令关闭 guest 虚拟机。您可以通过修改 guest 虚拟机配置文件中的 on_shutdown 参数来控制客户机虚拟机如何重启的行为。对 on_shutdown 参数的任何更改都将在域关闭并重新启动后生效。

virsh shutdown 命令可使用以下可选参数：

- --mode 选择关闭模式。这可以是 acpi、agent、initctl、信号（） 或泛虚拟化。

例 20.13. 如何关闭客户机虚拟机

以下示例使用 acpi 模式关闭 guest1 虚拟机：

```
# virsh shutdown guest1 --mode acpi
Domain guest1 is being shutdown
```

20.8.2. 挂起客户机虚拟机

virsh suspend domain 命令挂起客户机虚拟机。

当客户机虚拟机处于挂起状态时，它会消耗系统 RAM，而不是处理器资源。在客户机虚拟机暂停时，不会发生磁盘和网络 I/O。此操作是即时的，guest 虚拟机只能使用 virsh resume 命令重启。在 临时 虚拟机中运行这个命令会删除它。

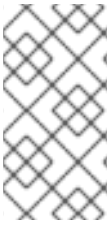
例 20.14. 如何挂起客户机虚拟机

以下示例挂起 guest1 虚拟机：

```
# virsh suspend guest1
```

20.8.3. 重置虚拟机

virsh reset domain 立即重置 guest 虚拟机，无需关闭任何 guest。重置会在计算机上模拟重置按钮，其中所有客户机硬件都看到 RST 行并重新初始化内部状态。请注意，没有 guest 虚拟机操作系统关闭，则数据丢失的风险。



注意

重置虚拟机不会应用任何待处理的域配置更改。对域配置的更改仅在完全关闭并重新启动域后生效。

例 20.15. 如何重置客户机虚拟机

以下示例重置 guest1 虚拟机：

```
# virsh reset guest1
```

20.8.4. 在 Order 中停止正在运行的客户机虚拟机以重启它

virsh managedsave domain --bypass-cache --running | --paused | --verbose 命令保存并销毁（停止）正在运行的客户端虚拟机，以便稍后从同一状态重启它。与 **virsh start** 命令一起使用时，它将从此保存点自动启动。如果与 **--bypass-cache** 参数一起使用，则保存将避免文件系统缓存。请注意，这个选项可能会减慢保存进程速度，并使用 **--verbose** 选项显示转储进程的进度。在正常情况下，受管保存将决定在使用正在运行或暂停状态，具体由客户机虚拟机在保存完成时的状态决定。但是，可以使用 **--running** 选项覆盖它，以指示它必须保留为 **running** 状态，或使用 **--paused** 选项将其保留为暂停状态。要删除受管保存状态，请使用 **virsh managedsave-remove** 命令，该命令强制客户端虚拟机在下次启动时进行完全引导。请注意，整个管理的保存过程可以通过 **domjobinfo** 命令监控，也可使用 **domjobabort** 命令取消。

例 20.16. 如何停止正在运行的虚拟客户机并保存其配置

以下示例停止 guest1 虚拟机并保存其正在运行的配置设置，以便重启它：

```
# virsh managedsave guest1 --running
```

20.9. 删除和删除虚拟机

20.9.1. 取消定义虚拟机

virsh undefine domain [--managed-save] [storage] [--remove-all-storage] [--wipe-storage] [--snapshots-metadata] [---nvram] 命令取消定义域。如果域不活跃，则配置将完全删除。如果域处于活动状态（正在运行），它将转换为 临时 域。当客户机虚拟机变为不活动状态时，配置将完全删除。

此命令可以采用以下参数：

- **--managed-save** - 此参数可确保同时清理任何受管保存镜像。如果不使用这个参数，尝试取消定义具有受管保存的客户机虚拟机将失败。
-

--snapshots-metadata - 此参数保证在未保护不活跃的虚拟机虚拟机时，也能够清理任何快照（如带有 snapshot-list所示）。请注意，任何尝试不使用快照元数据取消定义不活跃的客户机虚拟机都将会失败。如果使用这个参数且 guest 虚拟机处于活跃状态，则忽略它。

- --storage - 使用这个参数要求将卷目标名称或源路径的逗号分隔列表与未定义域一起删除。此操作将在删除存储卷前取消定义它。请注意，这只能通过不活动的虚拟机完成，并且这只能使用由 libvirt 管理的存储卷。

- --remove-all-storage - 除了取消保护客户机虚拟机外，还会删除所有关联的存储卷。如果您要删除虚拟机，只有在没有其他虚拟机使用相同的相关存储时才选择这个选项。另一种方法是 virsh vol-delete。如需更多信息，请参阅 第 20.31 节 “删除存储卷”。

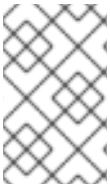
- --wipe-storage - 除了删除存储卷外，内容也会擦除。

例 20.17. 如何删除客户机虚拟机并删除其存储卷

以下示例取消定义 guest1 虚拟机并删除所有相关存储卷。未定义的客户机会变得 临时，因此会在关闭后删除：

```
# virsh undefine guest1 --remove-all-storage
```

20.9.2. 强制客户机虚拟机停止



注意

只有在您无法通过任何其他方法关闭 guest 虚拟机时，才应使用这个命令。

virsh destroy 命令启动一个立即非正常关闭，并停止指定的客户机虚拟机。使用 virsh destroy 可破坏 guest 虚拟机文件系统。仅在客户机虚拟机无响应时使用 virsh destroy 命令。使用 --graceful 选项的 virsh destroy 命令尝试在关闭虚拟机前清除磁盘镜像文件的缓存。

例 20.18. 如何使用硬关闭来立即关闭客户机虚拟机

以下示例会立即关闭 guest1 虚拟机，因为它可能不响应：

```
# virsh destroy guest1
```

您可能想通过 virsh undefine 命令对此进行操作。请查看 例 20.17 “如何删除客户机虚拟机并删除其存储卷”

20.10. 连接客户机虚拟机的 SERIAL CONSOLE

virsh 控制台 域 [--devname devicename] [--force] [--safe] 命令连接客户机虚拟机的虚拟串行控制台。例如，对于不提供 VNC 或 SPICE 协议的虚拟机（因此不提供 GUI 工具 的视频显示），并且没有网络连接（因此无法使用 SSH 进行交互），这非常有用。

可选的 --devname 参数指的是为客户机虚拟机配置的备用控制台、串行或并行设备的设备别名。如果省略此参数，则打开主控制台。如果指定了 --safe 选项，则只有在驱动程序支持安全控制台处理时才尝试连接。此选项指定服务器必须确保对控制台设备的独占访问权限。（可选）可以指定 force 选项，它可请求断开任何现有会话，例如在连接断开的情况下。

> **例 20.19. 如何在控制台模式中启动客户机虚拟机**
>
> 以下示例启动之前创建的 guest1 虚拟机，以便使用安全控制台处理连接到串行控制台：
>
> ```
> # virsh console guest1 --safe
> ```

20.11. 注入不可屏蔽的中断

virsh inject-nmi 域 向客户机虚拟机注入不可屏蔽中断(NMI)消息。这在响应时间至关重要时使用，如不可恢复的硬件错误。另外，virsh inject-nmi 可用于在 Windows 客户端中触发崩溃转储。

> **例 20.20. 如何将 NMI 注入到客户机虚拟机**
>
> 以下示例将 NMI 发送到 guest1 虚拟机：
>
> ```
> # virsh inject-nmi guest1
> ```

20.12. 检索有关您的虚拟机的信息

20.12.1. 显示设备块统计信息

默认情况下，virsh domblkstat 命令显示为域定义的第一个块设备块统计信息。要查看其他块设备的

统计信息，请使用 **virsh domblklist domain** 命令列出所有块设备，然后选择一个特定的块设备并通过指定域名后从 **virsh domblklist** 命令输出的 **Target** 或 **Source name** 来显示它。请注意，不是每个虚拟机监控程序都可以显示每个字段。为确保输出在其最明的表单中显示，请使用 **--human** 参数。

例 20.21. 如何显示客户机虚拟机的块统计信息

以下示例显示了为 **guest1** 虚拟机定义的设备，然后列出该设备的块设备。

```
# virsh domblklist guest1

Target     Source
------------------------------------------------
vda        /VirtualMachines/guest1.img
hdc        -

# virsh domblkstat guest1 vda --human
Device: vda
 number of read operations:      174670
 number of bytes read:           3219440128
 number of write operations:     23897
 number of bytes written:        164849664
 number of flush operations:     11577
 total duration of reads (ns):   1005410244506
 total duration of writes (ns):  1085306686457
 total duration of flushes (ns): 340645193294
```

20.12.2. 检索网络接口统计信息

virsh domifstat 域 **interface-device** 命令显示在给定客户机虚拟机上运行的指定设备的网络接口统计信息。

要确定为域定义了哪些接口设备，请使用 **virsh domiflist** 命令并使用 **Interface** 列中的输出。

例 20.22. 如何显示客户机虚拟机的网络统计信息

以下示例获取为 **guest1** 虚拟机定义的网络接口，然后显示所获取接口(**macvtap0**)上的网络统计信息：

```
# virsh domiflist guest1
Interface  Type       Source     Model       MAC
-------------------------------------------------------
macvtap0   direct     em1        rtl8139     12:34:00:0f:8a:4a
# virsh domifstat guest1 macvtap0
macvtap0 rx_bytes 51120
```

```
macvtap0 rx_packets 440
macvtap0 rx_errs 0
macvtap0 rx_drop 0
macvtap0 tx_bytes 231666
macvtap0 tx_packets 520
macvtap0 tx_errs 0
macvtap0 tx_drop 0
```

20.12.3. 修改客户机虚拟机虚拟接口的链路状态

virsh domif-setlink 域 interface-device state 命令将指定接口设备链接状态配置为 **up** 或 **down**。要确定为域定义了哪些接口设备，请使用 **virsh domiflist** 命令，并使用 **Interface** 或 **MAC** 列作为接口设备选项。默认情况下，**virsh domif-setlink** 更改正在运行的域的链接状态。若要修改域的永久配置，请使用 **--config** 参数。

例 20.23. 如何启用客户机虚拟机接口

以下示例显示确定 **rhel7** 域的接口设备，然后将链接设置为 **down**，最后设置为 ：

```
# virsh domiflist rhel7
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet0      network    default    virtio      52:54:00:01:1d:d0

# virsh domif-setlink rhel7 vnet0 down
Device updated successfully

# virsh domif-setlink rhel7 52:54:00:01:1d:d0 up
Device updated successfully
```

20.12.4. 列出客户机虚拟机虚拟接口的链接状态

virsh domif-getlink 域 interface-device 命令检索指定的接口设备链接状态。要确定为域定义了哪些接口设备，请使用 **virsh domiflist** 命令，并使用 **Interface** 或 **MAC** 列作为接口设备选项。默认情况下，**virsh domif-getlink** 检索正在运行的域的链接状态。若要检索域的永久配置，请使用 **--config** 选项。

例 20.24. 如何显示客户机虚拟机接口的链接状态

以下示例显示确定 **rhel7** 域的接口设备，然后确定其状态为 **up**，然后将状态更改为 **down**，然后验证更改是否成功：

```
# virsh domiflist rhel7
```

```
 Interface  Type       Source     Model       MAC
 -------------------------------------------------------
 vnet0      network    default    virtio      52:54:00:01:1d:d0

 # virsh domif-getlink rhel7 52:54:00:01:1d:d0
 52:54:00:01:1d:d0 up

 # virsh domif-setlink rhel7 vnet0 down
 Device updated successfully

 # virsh domif-getlink rhel7 vnet0
 vnet0 down
```

20.12.5. 设置网络接口带宽参数

virsh domiftune 域 interface-device 命令可检索或设置指定的域的接口带宽参数。要确定为域定义了哪些接口设备，请使用 **virsh domiflist** 命令，并使用 **Interface** 或 **MAC** 列作为接口设备选项。应使用以下格式：

```
# virsh domiftune domain interface [--inbound] [--outbound] [--config] [--live] [--current]
```

--config、**--live** 和 **--current** 选项包括在 第 20.43 节 “设置计划参数” 中。如果没有指定 **--inbound** 或 **--outbound** 选项，**virsh domiftune** 查询指定的网络接口并显示带宽设置。通过指定 **--inbound** 或 **--outbound**，以及平均、峰值和突发值，**virsh domiftune** 可设置带宽设置。至少需要平均值。要清除带宽设置，请提供 **0**（零）。有关平均、峰值和突发值的描述，请参阅 第 20.27.6.2 节 “附加接口设备”。

例 20.25. 如何设置客户机虚拟机网络接口参数

以下示例为名为 **guest1** 的客户机虚拟机设置 **eth0** 参数：

```
# virsh domiftune guest1 eth0 outbound --live
```

20.12.6. 检索内存统计信息

virsh dommemstat 域 [<period in seconds>] [--config] [--live] [--current] 命令显示正在运行的客户机虚拟机的内存统计信息。使用可选的 **period** 开关需要以秒为单位的时间。将此选项设置为大于 **0** 的值将允许 **balloon** 驱动程序返回其他统计信息，这些统计信息将通过运行后续 **dommemstat** 命令显示。将 **period** 选项设置为 **0**，停止 **balloon** 驱动程序集合，但不会清除 **balloon** 驱动程序中已存在的统计信息。您不能在不设置 **period** 选项的情况下使用 **--live**、**--config** 或 **--current** 选项。如果指定了 **--live** 选项，则只会收集客户机运行统计。如果使用 **--config** 选项，它将收集持久客户机的统计信息，但仅在下次启动后。如果使用 **--current** 选项，它将收集当前的统计信息。

可以使用 --live 和 --config 选项，但 --current 是独占的。如果没有指定标志，则客户机的状态将指示统计收集（运行或非运行）的行为。

例 20.26. 如何收集正在运行的客户机虚拟机的内存统计信息

以下示例显示了 rhel7 域中的内存统计信息：

```
# virsh dommemstat rhel7
actual 1048576
swap_in 0
swap_out 0
major_fault 2974
minor_fault 1272454
unused 246020
available 1011248
rss 865172
```

20.12.7. 在块设备中显示错误

virsh domblkerror domain 命令列出了处于错误状态的所有块设备，并在每个设备上检测到错误。在 virsh domstate 命令报告由于 I/O 错误而暂停 guest 虚拟机后，最好使用这个命令。

例 20.27. 如何显示虚拟机的块设备错误

以下示例显示了 guest1 虚拟机的块设备错误：

```
# virsh domblkerror guest1
```

20.12.8. 显示块设备大小

virsh domblkinfo 域 命令可列出虚拟机中特定块设备的容量、分配和物理块大小。使用 virsh domblklist 命令列出所有块设备，然后通过在域名后面指定 virsh domblklist 输出中指定 Target 或 Source name 来选择要显示特定块设备。

例 20.28. 如何显示块设备大小

在本例中，您要列出 rhel7 虚拟机上的块设备，然后显示每个设备的块大小。

```
# virsh domblklist rhel7
Target     Source
```

```
----------------------------------------------
vda      /home/vm-images/rhel7-os
vdb      /home/vm-images/rhel7-data

# virsh domblkinfo rhel7 vda
Capacity:       10737418240
Allocation:     8211980288
Physical:       10737418240

# virsh domblkinfo rhel7 /home/vm-images/rhel7-data
Capacity:       104857600
Allocation:     104857600
Physical:       104857600
```

20.12.9. 显示与客户机虚拟机关联的块设备

virsh domblklist 域 [--inactive] [--details] 命令显示与指定 **guest** 虚拟机关联的所有块设备表。

如果指定了 **--inactive**，则结果将显示要在下次引导时使用的设备，并且不显示当前运行的客户机虚拟机正在使用的设备。如果指定了 **--details**，则该磁盘类型和设备值将包含在表中。此表中显示的信息可与需要提供块设备的其他命令一起使用，如 **virsh domblkinfo** 和 **virsh snapshot-create**。在为 **virsh snapshot-create** 命令生成 **xmlfile** 上下文信息时，还可使用磁盘 **Target** 或 **Source** 上下文。

例 20.29. 如何显示与虚拟机关联的块设备

以下示例显示了与 **rhel7** 虚拟机关联的块设备的详细信息。

```
# virsh domblklist rhel7 --details
Type       Device     Target     Source
------------------------------------------------
file       disk       vda        /home/vm-images/rhel7-os
file       disk       vdb        /home/vm-images/rhel7-data
```

20.12.10. 显示与客户机虚拟机关联的虚拟接口

virsh domblklist domain 命令显示与指定域关联的所有虚拟接口的表。**virsh domiflist** 命令需要虚拟机的名称（或 域），并且可以选择使用 **--inactive** 参数。后者检索不活动而不是正在运行的配置，通过默认设置检索。如果指定了 **--inactive**，则结果显示要在下次引导时使用的设备，不显示当前被运行客户机使用的设备。需要虚拟接口的 **MAC** 地址（如 **detach-interface**、**domif-setlink**、**domif-getlink**、**domifstat**、**dmifstat** ）和 **domiftune**）接受此命令显示的输出。

例 20.30. 如何显示与客户机虚拟机关联的虚拟接口

以下示例显示了与 rhel7 虚拟机关联的虚拟接口，然后显示 vnet0 设备的网络接口统计信息。

```
# virsh domiflist rhel7
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet0      network    default    virtio      52:54:00:01:1d:d0

# virsh domifstat rhel7 vnet0
vnet0 rx_bytes 55308
vnet0 rx_packets 969
vnet0 rx_errs 0
vnet0 rx_drop 0
vnet0 tx_bytes 14341
vnet0 tx_packets 148
vnet0 tx_errs 0
vnet0 tx_drop 0
```

20.13. 使用快照

20.13.1. 通过复制数据来缩短回填链

本节演示了如何使用 virsh blockcommit 域 <path> [<bandwidth>] [<base>] [--shallow] [<top>] [--active] [--delete] [--wait] [---verbose] [--timeout < number>] [--pivot] [--keep-overlay] [--async] [--keep-relative] 命令缩短后备链。命令有许多选项，这些选项列在帮助菜单或 man page 中。

virsh blockcommit 命令会将数据从一个链的一部分复制到后备文件中，以便您可以接收链的其余部分以绕过提交的部分。例如，假设这是当前状态：

```
base ← snap1 ← snap2 ← active.
```

使用 virsh blockcommit 将 snap2 的内容移动到 snap1，您可以从链中删除 snap2，从而加快备份速度。

过程 20.1. 如何缩短备份链

- 输入以下命令，将 guest1 替换为您的客户机虚拟机和 disk1 的名称，并将其替换为您的磁盘名称。

```
# virsh blockcommit guest1 disk1 --base snap1 --top snap2 --wait --verbose
```

snap2 的内容移动到 snap1 中，结果如下：

基本 containerruntime snap1 ImagePullBackOff 活跃.Snap2 不再有效，可以删除

警告

virsh blockcommit 将破坏依赖于 --base 参数的任何文件（除依赖于 --top 参数的文件外，作为这些文件现在指向基础）。要防止这种情况，请不要将更改提交至多个客户端共享的文件中。--verbose 选项允许在屏幕上显示进度。

20.13.2. 通过隔离镜像来缩短反向链

virsh blockpull 可在以下应用程序中使用：

1. *通过使用来自其后备镜像链的数据填充镜像，以将其扁平化。使镜像文件自我包含，使其不再依赖于后备镜像，如下所示：*

   - *前：base.img containerruntime 活跃*
   - *之后：guest 和 Active 不再使用 base.img，Active 包含所有数据。*

2. *备份镜像链的扁平化部分。这可用于将快照扁平化到顶级镜像，如下所示：*

   - *前：基本 containerruntime sn1 containerruntimesn2 containerruntime active*
   - *之后：base.img containerruntime 活跃。请注意，当前活动 现在包含来自 sn1 和 sn2 的所有数据，并且 guest sn1 和 sn2 均没有使用 sn2。*

3. 将磁盘映像移到主机上的新文件系统中。这允许在客户端运行时移动镜像文件，如下所示：

   - before（原始镜像文件）：/fs1/base.vm.img
   - 之后： /fs2/active.vm.qcow2 现在是新文件系统，不再使用 /fs1/base.vm.img。

4. 通过后复制存储迁移，在实时迁移中非常有用。磁盘镜像会在实时迁移完成后从源主机复制到目标主机。

   在这里，会发生：前：/source-host/base.vm.img After:/destination-host/active.vm.qcow2./source-host/base.vm.img 不再被使用。

过程 20.2. 如何通过扁平化数据来缩短后备链

1. 在运行 virsh blockpull 之前创建快照可能会有帮助。为此，请使用 virsh snapshot-create-as 命令。在以下示例中，将 guest1 替换为您的客户端虚拟机的名称，并将 snap1 替换为您的快照的名称。

   ```
   # virsh snapshot-create-as guest1 snap1 --disk-only
   ```

2. 如果链看起来像是这样：基本 REPOSITORY snap1 snap1 containerruntime 活动，请输入以下命令，将 guest1 替换为您的客户端虚拟机和 path1 的源路径（例如/home/username/VirtualMachines/*）。

   ```
   # virsh blockpull guest1 path1
   ```

   此命令将 snap1 的备份文件从 snap2 提取到活跃状态，从而使数据从 snap2 拉取到活动中，从而使数据成为活动的基础。

3. virsh 块拉取 完成后，在链中创建额外镜像的快照的 libvirt 跟踪将非常有用。使用这个命令删除过期的快照的跟踪，将 guest1 替换为您的 guest 虚拟机的名称，snap1 替换为快照的名称。

   ```
   # virsh snapshot-delete guest1 snap1 --metadata
   ```

可以按照如下所示执行 virsh blockpull 的其他应用程序：

例 20.31. 如何扁平化一个镜像并使用其后备镜像链中的数据进行填充

以下示例在 guest guest1 上扁平化 vda 虚拟磁盘，并使用其后备镜像链中的数据填充镜像，等待填充操作完成。

```
# virsh blockpull guest1 vda --wait
```

例 20.32. 如何扁平化后备镜像链的一部分

以下示例基于 /path/to/base.img 磁盘镜像在 guest1 上扁平化 vda 虚拟磁盘。

```
# virsh blockpull guest1 vda /path/to/base.img --base --wait
```

例 20.33. 如何将磁盘镜像移动到主机上的新文件系统中

要将磁盘映像移动到主机上的新文件系统中，请运行以下命令：在每个命令中，将 guest1 替换为您的客户机虚拟机和 disk1 的名称，替换为您的虚拟磁盘的名称。更改 XML 文件名和快照位置和路径：

```
# virsh snapshot-create guest1 --xmlfile /path/to/snap1.xml --disk-only
```

```
# virsh blockpull guest1 disk1 --wait
```

例 20.34. 如何在后复制存储迁移中使用实时迁移

要在 post-copy 存储迁移中使用实时迁移，请输入以下命令：

在目标上，输入以下命令将后备文件替换为主机上的后备文件的名称和位置。

```
# qemu-img create -f qcow2 -o backing_file=/source-host/vm.img /destination-
host/vm.qcow2
```

在源上输入以下命令，将 guest1 替换为您的客户端虚拟机的名称：

```
# virsh migrate guest1
```

在目的地中，输入以下命令，将 guest1 替换为您的客户端虚拟机的名称，并将 disk1 替换为您的虚拟磁盘的名称：

```
# virsh blockpull guest1 disk1 --wait
```

20.13.3. 更改客户机虚拟机块设备的大小

virsh blockresize 命令可以用于在客户机虚拟机运行时调整客户机虚拟机块设备的大小，使用块设备的绝对路径，其还对应于唯一目标名称(<target dev="name"/>)或源文件(<source file="name"/>)。这可以应用于附加到客户机虚拟机的一个磁盘设备（您可以使用命令 virsh domblklist 来显示表，其中显示了与给定客户机虚拟机关联的所有块设备的信息）。

注意

实时镜像大小始终调整映像大小，但客户机无法立即获取。使用最新的 guest 内核时，将自动更新 virtio-blk 设备的大小（旧内核需要重新启动客户端）。使用 SCSI 设备时，需要通过命令手动触发客户机中的重新扫描，即 echo > /sys/class/scsi_device/0:0:0:0/device/rescan。另外，使用 IDE 时，需要在获取新大小前重新启动 guest。

例 20.35. 如何重新定义客户机虚拟机块设备大小

以下示例将 guest1 虚拟机的块设备重新调整为 90 字节：

```
# virsh blockresize guest1 90 B
```

20.14. 显示到图形显示的连接的 URI

运行 virsh domdisplay 命令将输出 URI，然后使用此 URI 通过 VNC、SPICE 或 RDP 连接到 guest 虚拟机的图形显示。可以使用可选的 --type 指定图形显示类型。如果使用 --include-password 参数，则 URI 中将包含 SPICE 通道密码。

例 20.36. 如何显示 SPICE 的 URI

以下示例显示了 SPICE 的 URI，这是虚拟机 guest1 使用的图形显示：

```
# virsh domdisplay --type spice guest1
spice://192.0.2.1:5900
```

有关连接 URI 的更多信息，请参阅 libvirt 上游页面。

20.15. 显示 VNC 显示的 IP 地址和端口号

virsh vncdisplay 命令返回指定 guest 虚拟机的 VNC 桌面的 IP 地址和端口号。如果 guest 不可用，则会显示退出代码 1。

请注意，为了使这个命令正常工作，必须将 VNC 指定为 guest XML 文件的 devices 元素中的图形类型。有关详情请参考 第 23.17.11 节 “图形帧缓冲”。

例 20.37. 如何显示 VNC 的 IP 地址和端口号

以下示例显示了 guest1 虚拟机的 VNC 显示端口号：

```
# virsh vncdisplay guest1
127.0.0.1:0
```

20.16. 不使用丢弃块

virsh domfstrim domain [--minimum bytes] [--mountpoint mountPoint] 命令可在指定运行客户机虚拟机中的所有挂载的文件系统上调用 fstrim 程序。这个丢弃块不会被文件系统使用。如果使用 --minimum 参数，必须指定字节数。该数量将作为连续可用范围的长度发送到客户机内核。值小于这个数量可能会被忽略。增加这个值将利用不碎片化可用空间的文件系统创建竞争。请注意，不是此情况下的所有块都将被丢弃。默认最小值为零，表示丢弃每个可用块。如果您将这个值增加到大于零，则 fstrim 操作将更加快速地用于带有错误碎片可用空间的文件系统，尽管并非所有块都会被丢弃。如果用户只想修剪一个特定的挂载点，则应使用 --mountpoint 参数并应指定挂载点。

例 20.38. 如何丢弃未使用块

以下示例修剪在名为 guest1 的客户机虚拟机中运行的文件系统：

```
# virsh domfstrim guest1 --minimum 0
```

20.17. 客户端虚拟机检索命令

20.17.1. 显示主机物理机器名称

virsh domhostname domain 命令显示虚拟机监控程序可发布它的指定 **guest** 虚拟机的物理主机名。

例 20.39. 如何显示主机物理机器名称

以下示例显示 **guest1** 虚拟机的主机物理机器名称（如果虚拟机监控程序可用）：

```
# virsh domhostname guest1
```

20.17.2. 显示有关虚拟机的一般信息

virsh dominfo 域 命令显示有关指定 **guest** 虚拟机的基本信息。此命令也可以与选项 **[- domain] guestname** 一起使用。

例 20.40. 如何显示有关客户机虚拟机的一般信息

以下示例显示了关于名为 **guest1** 的客户机虚拟机的一般信息：

```
# virsh dominfo guest1
Id:             8
Name:           guest1
UUID:           90e0d63e-d5c1-4735-91f6-20a32ca22c40
OS Type:        hvm
State:          running
CPU(s):         1
CPU time:       271.9s
Max memory:     1048576 KiB
Used memory:    1048576 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c422,c469 (enforcing)
```

20.17.3. 显示虚拟机的 ID 号

虽然 virsh list 包括其输出中的 ID，但 virsh domid domain>|<ID 会显示客户机虚拟机的 ID，只要它正在运行。每次运行虚拟机时 ID 将更改。如果 guest 虚拟机已关闭，则计算机名称将显示为一系列破折号('-----')。此命令也可以与 [- domain guestname] 选项搭配使用。

> **例 20.41. 如何显示虚拟机的 ID 号**
>
> 要运行此命令并接收任何可用的输出，虚拟机应正在运行。以下示例生成 guest1 虚拟机的 ID 号：
>
> ```
> # virsh domid guest1
> 8
> ```

20.17.4. 在客户机虚拟机上中止运行任务

virsh domjobabort 域 命令在指定的 guest 虚拟机上中止当前正在运行的作业。此命令也可以与 [-domain guestname] 选项搭配使用。

> **例 20.42. 如何在客户机虚拟机上中止正在运行的作业**
>
> 在本例中，在要中止的 guest1 虚拟机上有一个作业正在运行。运行命令时，将 guest1 更改为虚拟机的名称：
>
> ```
> # virsh domjobabort guest1
> ```

20.17.5. 显示关于在客户机虚拟机中运行的作业的信息

virsh domjobinfo domain 命令显示有关在指定客户机虚拟机上运行的作业的信息，包括迁移统计信息。此命令也可以与 [--domain guestname] 选项一起使用，或使用 --completed 选项返回最近完成的作业统计信息。

> **例 20.43. 如何显示统计反馈**
>
> 以下示例列出了有关 guest1 虚拟机的统计信息：
>
> ```
> # virsh domjobinfo guest1
> Job type:         Unbounded
> Time elapsed:     1603        ms
> ```

```
Data processed:   47.004 MiB
Data remaining:   658.633 MiB
Data total:       1.125 GiB
Memory processed: 47.004 MiB
Memory remaining: 658.633 MiB
Memory total:     1.125 GiB
Constant pages:   114382
Normal pages:     12005
Normal data:      46.895 MiB
Expected downtime: 0           ms
Compression cache: 64.000 MiB
Compressed data:  0.000 B
Compressed pages: 0
Compression cache misses: 12005
Compression overflows: 0
```

20.17.6. 显示客户机虚拟机的名称

virsh domname domainID 命令显示其名称 guest 虚拟机名称（指定其 ID 或 UUID）。虽然 virsh list --all 命令也会显示 guest 虚拟机的名称，但此命令仅会列出 guest 的名称。

例 20.44. 如何显示客户机虚拟机的名称

以下示例显示了带有域 ID 8 的客户机虚拟机的名称：

```
# virsh domname 8
guest1
```

20.17.7. 显示虚拟机的状态

virsh domstate domain 命令显示给定 guest 虚拟机的状态。使用 --reason 参数也会显示显示状态的原因。此命令还可与 [--domain guestname] 选项以及 --reason 选项搭配使用，其中显示状态的原因。如果命令显示错误，您应当运行命令 virsh domblkerror。详情请查看 第 20.12.7 节 “在块设备中显示错误”。

例 20.45. 如何显示客户机虚拟机的当前状态

以下示例显示了 guest1 虚拟机的当前状态：

```
# virsh domstate guest1
running
```

20.17.8. 显示到虚拟机的连接状态

virsh domcontrol 域 显示用来控制指定客户机虚拟机的接口状态。对于不是 OK 或 Error 状态，它还将打印自控制接口进入显示状态以来经过的秒数。

例 20.46. 如何显示客户机虚拟机的接口状态

以下示例显示了 guest1 虚拟机接口的当前状态。

```
# virsh domcontrol guest1
ok
```

20.18. 将 QEMU 参数转换为 DOMAIN XML

virsh domxml-from-native 命令提供了一种将现有的 QEMU 参数集合转换为可由 libvirt 使用的域 XML 配置文件的方法。请注意，这个命令仅用于转换之前从命令行启动的现有 QEMU 虚拟机，以便通过 libvirt 管理它们。因此，这里描述的方法不应用于从头开始创建新 guest。必须使用 virsh、virt-install 或 virt-manager 创建新 guest。更多信息可在 libvirt 上游网站 中找到。

过程 20.3. 如何将 QEMU 客户机转换为 libvirt

1. 以参数文件（文件类型为 *.args）开头，本例中为 demo.args ：

```
$ cat demo.args
LC_ALL=C
PATH=/bin
HOME=/home/test
USER=test
LOGNAME=test /usr/bin/qemu -S -M pc -m 214 -smp 1 -nographic -monitor pty -no-acpi -
boot c -hda /dev/HostVG/QEMUGuest1 -net none -serial none -parallel none -usb
```

2. 要将此文件转换为域 XML 文件，以便 guest 可以由 libvirt 管理，请输入以下命令。记得将 qemu-guest1 替换为您的虚拟客户机虚拟机的名称，并将 demo.args 替换为 QEMU args 的文件名。

```
# virsh domxml-from-native qemu-guest1 demo.args
```

这个命令将 demo.args 文件转换为以下域 XML 文件：

图 20.1. 客户端虚拟机新配置文件

```
<domain type='qemu'>
  <uuid>00000000-0000-0000-0000-000000000000</uuid>
  <memory>219136</memory>
  <currentMemory>219136</currentMemory>
  <vcpu>1</vcpu>
  <os>
    <type arch='i686' machine='pc'>hvm</type>
    <boot dev='hd'/>
  </os>
  <clock offset='utc'/>
  <on_poweroff>destroy</on_poweroff>
  <on_reboot>restart</on_reboot>
  <on_crash>destroy</on_crash>
  <devices>
    <emulator>/usr/bin/qemu</emulator>
    <disk type='block' device='disk'>
      <source dev='/dev/HostVG/QEMUGuest1'/>
      <target dev='hda' bus='ide'/>
    </disk>
  </devices>
</domain>
```

20.19. 使用 VIRSH DUMP创建 GUEST 虚拟机内核的转储文件

对客户机虚拟机进行故障排除的方法之一（除了 kdump 和 pvpanic外），使用 virsh dump domain corefilepath [--bypass-cache] {--live | --crash | --reset} [--verbose] [--memory-only] [--format=format] 命令。这会创建一个转储文件，其中包含客户端虚拟机的核心，以便它可以分析，例如 crash 实用程序。

特别是，运行 virsh dump 命令将客户机虚拟机内核转储到您提供的核心文件路径指定的文件中。请注意，一些虚拟机监控程序可能会对这个操作有限制，可能需要用户手动确保对 corefilepath 参数指定的文件和路径的适当权限。该命令支持 SR-IOV 设备以及其他 passthrough 设备。以下参数会被支持，并具有以下效果：

- --bypass-cache - 保存的文件不会绕过主机的文件系统缓存。它对文件的内容没有影响。请注意，选择这个选项可能会减慢转储操作的速度。
- --live 将保存文件，因为客户机虚拟机继续运行，且不会暂停或停止客户机虚拟机。
- --crash 将客户机虚拟机置于崩溃状态，而不是在保存转储文件时将其保留为已暂停状态。客户机虚拟机将列出为"中断"，原因为"垃圾"。

- --reset - 成功保存转储文件时，将重置 guest 虚拟机。

- --verbose 显示转储过程的进度

- --memory-only - 使用此选项运行转储文件将创建一个转储文件，该文件的内容将仅包含客户机虚拟机的内存和 CPU 通用寄存器文件。在运行完整转储时，应使用这个选项。当无法实时迁移客户机虚拟机时（由于 passthrough PCI 设备），可能会发生这种情况。

  您可以使用 --format=格式 选项保存只内存转储。可用的格式如下：

  - ELF - 默认未压缩格式
  - kdump-zlib - kdump 压缩格式 zlib 压缩
  - kdump-lzo - kdump 压缩格式使用 LZO 压缩
  - kdump-snappy - kdump 压缩格式使用 Snappy 压缩

重要

crash 工具不再支持 virsh dump 命令的默认内核转储文件格式。如果您使用崩溃 来分析由 virsh dump 创建的内核转储文件，则必须使用 --memory-only 选项。

另外，在创建内核转储文件时，还必须使用 --memory-only 选项以附加到红帽支持问题单。

请注意，可以使用 virsh domjobinfo 命令监控整个过程，并可使用 virsh domjobabort 命令取消。

例 20.47. 如何使用 virsh 创建转储文件

以下示例创建 guest1 虚拟机的转储文件，将其保存到 core/file/path.file 文件中，然后重置 guest。使用这个命令的最常见情况是，如果您的客户机虚拟机没有正确行为，则使用这个命令的最常

见情况是：

```
# virsh dump guest1 core/file/path.file --memory-only --reset
```

20.20. 创建虚拟机 XML 转储（配置文件）

virsh dumpxml 命令将返回 guest 虚拟机的 XML 配置文件，然后您可以根据需要使用、保存或更改它们。

然后，可以使用 XML 文件(guest.xml)来重新创建客户机虚拟机（请参阅 第 20.22 节 “编辑虚拟机 XML 配置设置”您可以编辑此 XML 配置文件来配置附加设备或部署额外的客户机虚拟机。

例 20.48. 如何检索客户机虚拟机的 XML 文件

以下示例检索 guest1 虚拟机的 XML 配置，将其写入 guest1.xml 文件，然后验证进程是否已成功完成。

```
# virsh dumpxml guest1 > guest1.xml
# cat guest1.xml
<domain type='kvm'>
  <name>guest1-rhel6-64</name>
  <uuid>b8d7388a-bbf2-db3a-e962-b97ca6e514bd</uuid>
  <memory>2097152</memory>
  <currentMemory>2097152</currentMemory>
  <vcpu>2</vcpu>
  <os>
    <type arch='x86_64' machine='rhel6.2.0'>hvm</type>
    <boot dev='hd'/>
  </os>
[...]
```

20.21. 从配置文件创建客户机虚拟机

可从 XML 配置文件创建客户机虚拟机。您可以从之前创建的客户机虚拟机复制现有的 XML，或者使用 virsh dumpxml 命令。

例 20.49. 如何从 XML 文件创建客户机虚拟机

以下示例从现有 guest1.xml 配置文件创建新虚拟机。开始之前，您需要先使用此文件。您可以使用 virsh dumpxml 命令检索该文件。具体步骤请查看 例 20.48 “如何检索客户机虚拟机的 XML 文

件”。

```
# virsh create guest1.xml
```

20.22. 编辑虚拟机 XML 配置设置

通过 virsh edit 命令，用户可以编辑指定 guest 的域 XML 配置文件。运行此命令会在文本编辑器中打开 XML 文件，由 $EDITOR shell 参数指定（默认设置为 vi ）。

例 20.50. 如何编辑客户机虚拟机的 XML 配置设置

以下示例在默认文本编辑器中打开与 guest1 虚拟机关联的 XML 配置文件：

```
# virsh edit guest1
```

20.23. 在 KVM 客户机虚拟机中添加多功能 PCI 设备

在 KVM 客户机虚拟机中添加多功能 PCI 设备：

1. 运行 virsh edit guestname 命令，编辑 guest 虚拟机的 XML 配置文件。

2. 在 <address> 元素中，添加 multifunction='on' 属性。这可让将其他功能用于特定的多功能 PCI 设备。

```
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none'/>
<source file='/var/lib/libvirt/images/rhel62-1.img'/>
<target dev='vda' bus='virtio'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'
multifunction='on'/>
</disk>
```

对于有两个功能的 PCI 设备，修改 XML 配置文件使其包含第二个设备，其插槽号与第一个设备和不同的功能号，如 function='0x1'。例如：

```
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none'/>
```

```
<source file='/var/lib/libvirt/images/rhel62-1.img'/>
<target dev='vda' bus='virtio'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'
multifunction='on'/>
</disk>
<disk type='file' device='disk'>
<driver name='qemu' type='raw' cache='none'/>
<source file='/var/lib/libvirt/images/rhel62-2.img'/>
<target dev='vdb' bus='virtio'/>
<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x1'/>
</disk>
```

3. *运行 **lspci** 命令。**KVM** 客户机虚拟机的输出显示了 **virtio** 块设备：*

```
$ lspci

00:05.0 SCSI storage controller: Red Hat, Inc Virtio block device
00:05.1 SCSI storage controller: Red Hat, Inc Virtio block device
```

注意

***SeaBIOS** 应用以实际模式运行，以便与 **BIOS** 接口兼容。这将限制可用的内存量。因此，**SeanBIOS** 只能处理有限数量的磁盘。目前，支持的磁盘数量是：*

- ***virtio-scsi — 64***
- ***virtio-blk - 4***
- ***一个hci/sata - 24（4 个控制器连接了所有 6 端口）***
- ***USB-storage - 4***

作为临时解决方案，当将大量磁盘附加到虚拟机时，请确定您的系统磁盘有一个小的 **pci** 插槽号，因此 **SeaBIOS** 会首先在扫描 **pci** 总线时看到它。另外，也建议使用 **virtio-scsi** 设备而不是 **virtio-blk**，因为每个磁盘内存开销比较小。

20.24. 显示指定客户机虚拟机的 CPU 统计

***virsh cpu-stats 域 --total start count** 命令提供指定客户机虚拟机的 CPU 统计信息。默认情况下，它*

显示所有 CPU 的统计信息，以及总计。--total 选项仅显示总统计信息。count 选项将仅显示 计算 CPU 的统计信息。

例 20.51. 如何生成客户机虚拟机的 CPU 统计

以下示例为名为 guest1 的客户机虚拟机生成 CPU 统计信息。

```
# virsh cpu-stats guest1

CPU0:
 cpu_time         242.054322158 seconds
 vcpu_time        110.969228362 seconds
CPU1:
 cpu_time         170.450478364 seconds
 vcpu_time        106.889510980 seconds
CPU2:
 cpu_time         332.899774780 seconds
 vcpu_time        192.059921774 seconds
CPU3:
 cpu_time         163.451025019 seconds
 vcpu_time        88.008556137 seconds
Total:
 cpu_time         908.855600321 seconds
 user_time        22.110000000 seconds
 system_time      35.830000000 seconds
```

20.25. 获取虚拟客户机控制台的截屏

virsh screenshot guestname [imagefilepath] 命令截取当前虚拟客户机控制台的屏幕截图，并将其存储在文件中。如果没有提供文件路径，则会将屏幕截图保存到当前目录中。如果虚拟机监控程序支持多个 guest 虚拟机显示，请使用 **--screen screenID** 选项指定要捕获的屏幕。

例 20.52. 如何获取客户端机器控制台的截屏

以下示例获取 guest1 计算机的屏幕截图，并将其保存为 **/home/username/pics/guest1-screen.png** ：

```
# virsh screenshot guest1 /home/username/pics/guest1-screen.ppm
Screenshot saved to /home/username/pics/guest1-screen.ppm, with type of image/x-portable-pixmap
```

20.26. 将密钥组合发送到指定的虚拟客户机虚拟机

通过 virsh send-key domain --codeset -- 保持time keycode 命令，您可以将序列作为密钥代码发送到特定的 guest 虚拟机。每个 键码 可以是数字值，也可以是以下对应代码集的符号链接名称。

如果指定了 -- keeptime，则会为每个按键数以毫秒为单位进行。借助 --codeset，您可以指定代码集，默认为 Linux，但允许以下选项：

- Linux - 选择这个选项可使符号链接名称与对应的 Linux 键恒定宏名称匹配，而数字值则是由 Linux 通用输入事件子系统提供的。
- XT - 这将发送 XT 键盘控制器定义的值。不提供符号链接名称
- atset1 - 数字值是指 AT 键盘控制器 set1（兼容 XT 兼容集）定义的。与 atset1 的扩展密钥码可能与 XT codeset 中的扩展密钥码不同。不提供符号名。
- atset2 - 数字值由 AT 键盘控制器定义，设置 2。不提供符号名。
- atset3 - 数字值由 AT 键盘控制器定义，设置 3（兼容PS/2）。不提供符号名。
- os_x - 数字值由 OS-X 键盘输入子系统定义。符号链接名称与相应的 OS-X 密钥恒定宏名匹配。
- xt_kbd - 数字值由 Linux KBD 设备定义。这些是原始 XT 代码集的一个变体，但通常使用不同的编码器。不提供符号名。
- win32 - 数字值由 Win32 键盘输入子系统定义。符号链接名称与对应的 Win32 密钥恒定宏名匹配。
- USB - 用于键盘输入的 USB HID 规范定义数字值。不提供符号名。
- rfb - 用于发送原始密钥码的 RFB 扩展定义的值。这些是 XT 代码集的一个变体，但扩展密钥码的低位是第二个位，而不是第一个字节的高位。不提供符号名。

例 20.53. 如何将按键组合发送到客户机虚拟机

以下示例将 Linux 编码中的 Left Ctrl、LeftAlt 和 Delete 发送到 guest1 虚拟机，并将它们容纳 1 秒。这些密钥都同时发送，可能由客户机以随机顺序接收：

```
# virsh send-key guest1 --codeset Linux --holdtime 1000 KEY_LEFTCTRL KEY_LEFTALT
KEY_DELETE
```

注意

如果指定了多个密钥码，它们都同时发送到客户机虚拟机，因此可能以随机顺序接收。如果需要不同的密钥码，则必须多次运行 virsh send-key 命令，以便获得要发送序列的顺序。

20.27. 主机机器管理

这部分包含管理主机系统（命令作为 节点 ）所需的命令。

20.27.1. 显示主机信息

virsh nodeinfo 命令显示有关主机的基本信息，包括模型号、CPU 的数量、CPU 类型和物理内存大小。输出对应于 virNodeInfo 结构。具体来说，"CPU 插槽"字段显示每个 NUMA 单元的 CPU 插槽数。

例 20.54. 如何显示主机机器的信息

以下示例检索有关您的主机的信息：

```
$ virsh nodeinfo
CPU model:           x86_64
CPU(s):              4
CPU frequency:       1199 MHz
CPU socket(s):       1
Core(s) per socket:  2
Thread(s) per core:  2
NUMA cell(s):        1
Memory size:         3715908 KiB
```

20.27.2. 设置 NUMA 参数

virsh numatune 命令可以设置或检索指定客户机虚拟机的 NUMA 参数。在客户机虚拟机的配置 XML 文件内，这些参数嵌套在 <numatune> 元素中。如果不使用标志，则仅显示当前设置。numatune

domain 命令需要指定的 guest 虚拟机名称，并可以采用以下参数：

- --mode - 模式可以设置为 strict、interleave 或 preferred。运行域在实时模式时无法更改其模式，除非 guest 虚拟机在 严格 模式下启动。
- --nodeset 包含供主机物理计算机用于运行客户机虚拟机的 NUMA 节点列表。列表包含节点，它们各自用逗号分隔，使用短划线 - 用于节点范围，使用 ^ 排除某个节点。
- 每个实例只能使用以下三个标志之一
  - --config 将在持久客户机虚拟机下次引导时生效
  - --live 将设置正在运行的虚拟客户机的调度程序信息。
  - --current 将影响客户虚拟机的当前状态。

例 20.55. 如何为客户机虚拟机设置 NUMA 参数

以下示例为正在运行的 guest1 虚拟机将 NUMA 模式设置为 strict 用于节点 0、2 和 3：

```
# virsh numatune guest1 --mode strict --nodeset 0,2-3 --live
```

运行此命令会将 guest1 的运行配置更改为其 XML 文件中的以下配置。

```
<numatune>
    <memory mode='strict' nodeset='0,2-3'/>
</numatune>
```

20.27.3. 在 NUMA Cell 中显示 Free Memory 的挂载

virsh freecell 命令在指定 NUMA 单元中的机器上显示可用内存量。这个命令可以根据指定的选项，在机器上提供三个不同的内存显示之一。 指定单元格。

例 20.56. 如何显示虚拟机和 NUMA 单元的内存属性

以下命令显示所有单元的可用内存总量：

```
# virsh freecell
Total: 684096 KiB
```

要在单个单元中也显示可用内存数量，请使用 **--all** 选项：

```
# virsh freecell --all
    0:     804676 KiB
--------------------
Total:     804676 KiB
```

要显示特定单元中的各个内存量，请使用 **--cellno** 选项：

```
# virsh freecell --cellno 0
0: 772496 KiB
```

20.27.4. 显示 CPU 列表

virsh nodecpumap 命令显示主机可以使用的 CPU 数量，它还列出了当前在线的数量。

例 20.57. 如何显示可用于主机的 CPU 数量

以下示例显示了主机可用的 CPU 数量：

```
# virsh nodecpumap
  CPUs present: 4
  CPUs online: 1
  CPU map: y
```

20.27.5. 显示 CPU 统计信息

virsh nodecpustats [cpu_number] [--percent] 命令显示主机的 CPU 负载状态的统计信息。如果指定了 CPU，则统计信息仅针对指定的 CPU。如果指定了 百分比 选项，命令会显示通过一(1)秒记录的每个 CPU 统计的比例。

例 20.58. 如何显示 CPU 用量的统计信息

以下示例返回有关主机 CPU 负载的一般统计信息：

```
# virsh nodecpustats
user:                1056442260000000
system:               401675280000000
idle:                7549613380000000
iowait:                94593570000000
```

本例以百分比显示 CPU 编号 2 的统计信息：

```
# virsh nodecpustats 2 --percent
usage:          2.0%
user:           1.0%
system:         1.0%
idle:          98.0%
iowait:         0.0%
```

20.27.6. 管理设备

20.27.6.1. 使用 virsh 附加和更新设备

有关附加存储设备的详情请参考 第 13.3.6 节 “在客户机中添加存储设备”。

过程 20.4. 热插拔 USB 设备供客户机虚拟机使用

USB 设备可以附加到通过热插拔运行的虚拟机，或者在客户机关闭时连接。要在客户端中使用的设备必须附加到主机机器中。

1. 运行以下命令找到您要附加的 USB 设备：

```
# lsusb -v

idVendor           0x17ef Lenovo
idProduct          0x480f Integrated Webcam [R5U877]
```

2. 创建一个 XML 文件并为其提供逻辑名称（例如usb_device.xml ）。复制供应商和产品 ID 号（十六进制数字）与搜索中显示的信息完全相同。在 XML 文件中添加此信息，如 图 20.2 “USB 设备 XML 片断” 所示。记住该文件的名称，因为您需要在下一步中。

图 20.2. USB 设备 XML 片断

```
<hostdev mode='subsystem' type='usb' managed='yes'>
  <source>
    <vendor id='0x17ef'/>
    <product id='0x480f'/>
  </source>
</hostdev>
```

3. *通过运行以下命令来连接设备：运行该命令时，将 guest1 替换为虚拟机的名称，并将 usb_device.xml 替换为包含您上一步创建的厂商和产品 ID 的 XML 文件的名称。要使更改在下次重启时生效，请使用 --config 参数。要使更改对当前 guest 虚拟机生效，请使用 --current 参数。如需了解更多参数，请参阅 virsh man page。*

```
# virsh attach-device guest1 --file usb_device.xml --config
```

例 20.59. 如何从客户机虚拟机热拔设备

以下示例从 guest1 虚拟机分离使用 usb_device1.xml 文件 配置的 USB 设备：

```
# virsh detach-device guest1 --file usb_device.xml
```

20.27.6.2. 附加接口设备

virsh attach-interface 域类型 source [<target>] [<mac>] [<script>] [<model>] [<inbound>] [<inbound>] [< 出站>] [--config] [--live] [--current] 命令可使用以下参数：

- *--type - 允许您设置接口类型*
- *--source - 允许您设置网络接口源*
- *--live - 从正在运行的客户机虚拟机配置设置中获取值*
- *--config - 在下次引导时生效*
- *--current - 根据当前的配置设置获取其值*

- --target - 表示客户机虚拟机中的目标设备。
- -- MAC - 使用这个选项指定网络接口的 MAC 地址
- --script - 使用这个选项指定处理网桥的脚本文件的路径，而不是默认文件。
- --model - 使用这个选项指定型号类型。
- --inbound - 控制接口的入站带宽。可接受的值 的平均 是、峰值 和 burst。
- --outbound - 控制接口的出站带宽。可接受的值 的平均 是、峰值 和 burst。

注意

平均和峰值的值以每秒 KB 为单位表示，而在高峰速度内以千字节为单位，如网络 XML 上游文档 所述。

类型可以是 指定物理网络设备 的网络，或者 网桥 以指示设备的桥接。source 是设备的来源。要删除附加的设备，请使用 virsh detach-device 命令。

例 20.60. 如何将设备附加到客户机虚拟机

以下示例将 networkw 网络设备附加到 guest1 虚拟机。接口模型将作为 virtio 提供给客户端：

```
# virsh attach-interface guest1 networkw --model virtio
```

20.27.6.3. 更改 CDROM 的介质

virsh change-media 命令将 CDROM 的介质更改为另一个源或格式。该命令采用以下参数：也可在 man page 中找到有关这些参数的更多示例和说明。

- --path - 包含完全限定路径或磁盘设备目标的字符串
- --source - 包含介质源的字符串
- --eject - Ejects the media
- --insert - Inserts the media
- --update - 更新介质
- --current - 可以是 --live 和 --config，这取决于虚拟机监控程序驱动程序的实现
- --live - Alters running guest 虚拟机的实时配置
- --config - Alters persistent 配置，在下次引导时观察到的操作
- --force - 强制介质更改

20.27.7. 设置和显示节点内存参数

virsh node-memory-tune [shm-pages-to-scan] [shm-sleep-milisecs] [shm-merge-across-nodes] 命令显示并允许您设置节点内存参数。可使用这个命令设置以下参数：

- --shm-pages-to-scan - 在内核相同页面合并(KSM)服务进入休眠前将页面数量设置为扫描。
- --shm-sleep-milisecs - 设置 KSM 将在下一次扫描前休眠的 miliseconds 数
- --shm-merge-across-nodes - 指定是否可以合并来自不同 NUMA 节点的页面

例 20.61. 如何在 NUMA 节点之间合并内存页面

以下示例合并所有 NUMA 节点中的所有内存页面：

```
# virsh node-memory-tune --shm-merge-across-nodes 1
```

20.27.8. 列出主机上的设备

virsh nodedev-list --cap --tree 命令列出 libvirt 服务已知的主机上可用的所有设备。**--cap** 可以按能力类型过滤列表，每个设备由逗号隔开，并且不能与 **--tree** 一起使用。使用参数 **--tree**，将输出置于树结构中。

例 20.62. 如何显示主机上可用的设备

以下示例以树形格式列出主机上可用的设备。请注意，该列表已被截断：

```
# virsh nodedev-list --tree
computer
  |
  +- net_lo_00_00_00_00_00_00
  +- net_macvtap0_52_54_00_12_fe_50
  +- net_tun0
  +- net_virbr0_nic_52_54_00_03_7d_cb
  +- pci_0000_00_00_0
  +- pci_0000_00_02_0
  +- pci_0000_00_16_0
  +- pci_0000_00_19_0
  |   |
  |   +- net_eth0_f0_de_f1_3a_35_4f
 [...]
```

这个示例列出了主机上可用的 **SCSI** 设备：

```
# virsh nodedev-list --cap scsi
scsi_0_0_0_0
```

20.27.9. 在主机机器中创建设备

通过 **virsh nodedev-create file** 命令，您可以在主机物理机器上创建一个设备，然后将其分配给客户机虚拟机。虽然 **libvirt** 会自动检测哪些主机节点可用，但这个命令允许您注册 **libvirt** 没有检测到的硬

件。指定的文件应包含主机设备的顶级 <设备> 描述的 XML 描述。有关此类文件的示例，请参阅 例 20.65 “如何检索设备的 XML 文件”。

例 20.63. 如何从 XML 文件创建设备

在本例中，您已为您的 PCI 设备创建了一个 XML 文件，并将其保存为 scsi_host2.xml。以下命令允许您将这个设备附加到您的客户端：

```
# virsh nodedev-create scsi_host2.xml
```

20.27.10. 删除设备

virsh nodedev-destroy 命令可从主机中删除该设备。请注意，virsh 节点设备驱动程序不支持持久配置，因此重新引导主机计算机可使设备再次可用。

另请注意，不同的分配预期设备绑定到不同的后端驱动程序（vfio、kvm）。通过使用 --driver 参数，您可以指定预期的后端驱动程序。

例 20.64. 如何从主机物理机器中删除设备

以下示例从主机机器中删除名为 scsi_host2 的 SCSI 设备：

```
# virsh nodedev-destroy scsi_host2
```

20.27.11. 收集设备配置设置

virsh nodedev-dumpxml 设备 会输出指定主机设备的 XML 表示，包括设备名称等信息，设备连接的总线、供应商、产品 ID、功能以及 libvirt 使用的任何信息。参数 设备可以是设备名称 或 WWN 对（仅适用于 WWNN）、WWPN 格式（仅限HBA）。

例 20.65. 如何检索设备的 XML 文件

以下示例检索为 scsi_host2 标识的 SCSI 设备的 XML 文件。使用 virsh nodedev-list 命令获取的名称：

```
# virsh nodedev-dumpxml scsi_host2 <device> <name>scsi_host2</name>
<parent>scsi_host1</parent> <capability type='scsi_host'> <capability type='fc_host'>
<wwnn>2001001b32a9da5b</wwnn> <wwpn>2101001b32a9da5b</wwpn> </capability>
```

```
</capability> </device>
```

20.27.12. 为设备触发重置

virsh nodedev-reset device 命令触发针对指定的设备重置的设备。在客户机虚拟机通过或主机物理机器间传输节点设备之前，运行命令非常有用。libvirt 将根据需要自动执行该操作，但这个命令会在需要时明确重置。

例 20.66. 如何重置客户机虚拟机上的设备

以下示例在名为 scsi_host2 的客户机虚拟机上重置设备：

```
# virsh nodedev-reset scsi_host2
```

20.28. 检索客户机虚拟机信息

20.28.1. 获取虚拟机的域 ID

virsh domid 命令返回 guest 虚拟机的 ID。请注意，每次客户机启动或重启时此更改。此命令需要虚拟机的名称或虚拟机的 UUID。

例 20.67. 如何检索客户机虚拟机的域 ID

以下示例检索名为 guest1 的虚拟客户机的域 ID：

```
# virsh domid guest1
8
```

注意，domid 返回 - 对于处于关闭状态的客户端虚拟机。要确认虚拟机已关闭，您可以运行 virsh list --all 命令。

20.28.2. 获取客户机虚拟机的域名

virsh domname 命令返回给定 ID 或 UUID 的 guest 虚拟机的名称。请注意，每次客户端启动时 ID 会改变。

例 20.68. 如何检索虚拟机的 ID

以下示例检索 ID 为 8 的客户机虚拟机的名称：

```
# virsh domname 8
guest1
```

20.28.3. 获取客户机虚拟机的 UUID

virsh domuuid 命令返回给定客户机虚拟机或 ID 的通用基础镜像标识符。

例 20.69. 如何显示客户机虚拟机的 UUID

以下示例检索名为 **guest1** 的客户机虚拟机的 UUID：

```
# virsh domuuid guest1
r5b2-mySQL01 4a4c59a7-ee3f-c781-96e4-288f2862f011
```

20.28.4. 显示客户机虚拟机信息

virsh dominfo 命令显示有关该虚拟客户机虚拟机的名称、ID 或 UUID 的信息。请注意，每次虚拟机启动时 **ID** 会改变。

例 20.70. 如何显示客户机虚拟机常规详情

以下示例显示了关于名为 **guest1** 的客户机虚拟机的一般详情：

```
# virsh dominfo guest1
Id:             8
Name:           guest1
UUID:           90e0d63e-d5c1-4735-91f6-20a32ca22c48
OS Type:        hvm
State:          running
CPU(s):         1
CPU time:       32.6s
Max memory:     1048576 KiB
Used memory:    1048576 KiB
Persistent:     yes
Autostart:      disable
Managed save:   no
```

```
Security model: selinux
Security DOI:   0
Security label: system_u:system_r:svirt_t:s0:c552,c818 (enforcing)
```

20.29. 存储池命令

使用 libvirt，您可以管理各种存储解决方案，包括文件、原始分区和域特定格式，用于提供作为虚拟机中设备可见的存储卷。如需更多信息，请参阅 libvirt 上游页面。用于管理存储池的许多命令与用于客户机虚拟机的命令类似。

20.29.1. 搜索存储池 XML

virsh find-storage-pool-sources type 命令显示 XML 说明可找到的源的所有存储池。类型包括：**netfs**、**disk**、**dir**、**fs**、**iscsi**、**logical** 和 **gluster**。请注意，所有类型都对应于存储后端驱动程序，还有更多可用的类型（详情请参阅 **man page**）。您还可以使用 **--srcSpec** 选项提供模板源 XML 文件来进一步限制池的查询。

例 20.71. 如何列出可用存储池的 XML 设置

以下示例输出系统中所有可用逻辑存储池的 XML 设置：

```
# virsh find-storage-pool-sources logical
<sources>
  <source>
    <device path='/dev/mapper/luks-7a6bfc59-e7ed-4666-a2ed-6dcbff287149'/>
    <name>RHEL_dhcp-2-157</name>
    <format type='lvm2'/>
  </source>
</sources>
```

20.29.2. 查找存储池

virsh find-storage-pool-sources-as type 命令查找潜在的存储池源（给定的特定类型）。类型包括：**netfs**、**disk**、**dir**、**fs**、**iscsi**、**logical** 和 **gluster**。请注意，所有类型都对应于存储后端驱动程序，还有更多可用的类型（详情请参阅 **man page**）。该命令还采用可选参数 **主机**、**端口** 和 **启动器**。每个选项将指定查询的内容。

例 20.72. 如何找到潜在的存储池源

以下示例在指定主机中搜索基于磁盘的存储池。如果您不确定您的主机名，请首先运行命令 **virsh hostname** ：

```
# virsh find-storage-pool-sources-as disk --host myhost.example.com
```

20.29.3. 列出存储池信息

virsh pool-info pool 命令列出有关指定存储池对象的基本信息。此命令需要存储池的名称或 UUID。要检索此信息，请使用 **pool-list** 命令。

例 20.73. 如何检索存储池的信息

以下示例检索名为 **vdisk** 的存储池的信息：

```
# virsh pool-info vdisk

Name:           vdisk
UUID:
State:          running
Persistent:     yes
Autostart:      no
Capacity:       125 GB
Allocation:     0.00
Available:      125 GB
```

20.29.4. 列出可用存储池

virsh pool-list 命令列出 libvirt 已知的所有存储池对象。默认情况下，仅列出活跃的池；但是，使用 **--inactive** 参数仅列出非活动池，并使用 **--all** 参数列出所有存储池。此命令采用以下可选参数，该参数过滤搜索结果：

- **--inactive** - 列出不活跃的存储池
- **--all** - 列出活跃和不活跃的存储池
- **--persistent** - 列出持久性存储池
- **--transient** - 列出临时存储池

- --autostart - 列出启用了自动启动的存储池
- --no-autostart - 列出禁用自动启动的存储池
- --type type - 列出仅属于指定类型的池
- --details - 列出存储池的扩展详情

除了上述参数外，还有几种过滤标记可用于过滤列表的内容。-- persistent 将列表限制为 持久池，-- transient 将列表限制为临时池，-- autostart 将列表限制为自动启动池，最后 --no-autostart 将列表限制为禁用自动启动的存储池。

对于需要 --type 的所有存储池命令，池类型必须以逗号分隔。有效的池类型包括：dir、fs、netfs、逻辑、磁盘、iscsi、scsi、mpath、rbd、sheepdog 和 gluster。

details 选项指示 virsh 显示池持久性和容量相关信息。

注意

当此命令与旧服务器一起使用时，它被强制使用一系列带有固有竞争条件的 API 调用，其中池可能无法列出，或者在列表被收集时更改其状态。但是，较新的服务器没有这个问题。

例 20.74. 如何列出所有存储池

本例列出了活跃和不活跃的存储池：

```
# virsh pool-list --all
Name                 State      Autostart
-----------------------------------------
default              active     yes
vdisk                active     no
```

20.29.5. 刷新存储池列表

virsh pool-refresh pool 命令刷新存储池中包含的存储卷列表。

例 20.75. 如何刷新存储池中的存储卷列表

以下示例刷新名为 **vdisk** 的存储卷的列表：

```
# virsh pool-refresh vdisk

Pool vdisk refreshed
```

20.29.6. 创建、定义和启动存储池

20.29.6.1. 构建存储池

virsh pool-build pool 命令使用命令中指定的名称构建存储池。可选参数 **--overwrite** 和 **--no-overwrite** 只能用于 **FS** 存储池，或使用基于磁盘或逻辑类型的存储池。请注意，如果没有提供 **[--overwrite] 或 [--no-overwrite]**，且使用的池为 **FS**，则会假定类型是基于 目录的实际情况。除了池名称外，也可以使用存储池 **UUID**。

如果指定了 **--no-overwrite**，它会探测确定目标设备上是否已存在文件系统，如果目标设备中已存在，返回错误，或者使用 **mkfs** 来格式化目标设备（如果不存在）。如果指定了 **--overwrite**，则将执行 **mkfs** 命令，并覆盖目标设备中任何现有的数据。

例 20.76. 如何构建存储池

以下示例创建一个名为 **vdisk** 的基于磁盘的存储池：

```
# virsh pool-build vdisk

Pool vdisk built
```

20.29.6.2. 从 XML 文件定义存储池

virsh pool-define file 命令会创建，但不从 **XML** 文件 启动存储池对象。

例 20.77. 如何从 XML 文件中定义存储池

此示例假定您已使用存储池的设置创建了 XML 文件。例如：

```
<pool type="dir">
 <name>vdisk</name>
 <target>
        <path>/var/lib/libvirt/images</path>
 </target>
</pool>
```

然后，以下命令从 XML 文件（本例中为 vdisk.xml ）构建目录类型存储池：

```
# virsh pool-define vdisk.xml

Pool vdisk defined
```

要确认定义了存储池，请运行 virsh pool-list --all 命令，如 例 20.74 “如何列出所有存储池” 所示。但是，当运行 命令时，其状态将显示为 inactive，因为池还没有启动。有关启动存储池的说明，请参阅 例 20.81 “如何启动存储池”。

20.29.6.3. 创建存储池

virsh pool-create file 命令从关联的 XML 文件创建并启动存储池。

例 20.78. 如何从 XML 文件创建存储池

在这个示例中，假定您已使用存储池的设置创建了 XML 文件。例如：

```
<pool type="dir">
 <name>vdisk</name>
 <target>
        <path>/var/lib/libvirt/images</path>
 </target>
</pool>
```

以下示例根据 XML 文件（本例中为 vdisk.xml ）构建目录类型存储池：

```
# virsh pool-create vdisk.xml

Pool vdisk created
```

要确认创建了存储池，请运行 virsh pool-list --all 命令，如 例 20.74 “如何列出所有存储池” 所示。但是，当运行 命令时，其状态将显示为 inactive，因为池还没有启动。有关启动存储池的说明，请参阅 例 20.81 “如何启动存储池”。

20.29.6.4. 创建存储池

virsh pool-create-as name 命令从指定的原始参数创建和启动池对象名称。这个命令使用以下选项：

- --print-xml - 显示 XML 文件的内容，但不定义或创建存储池
- --type 类型定义 存储池类型。有关您可以使用的类型，请参阅 第 20.29.4 节 “列出可用存储池”。
- --source-host hostname - 用于底层存储的源主机物理机器
- --source-path 路径 - 底层存储的位置
- --source-dev 路径 - 底层存储的设备
- --source-name name - 源底层存储的名称
- --source-format 格式 - 源底层存储的格式
- --target path - 底层存储的目标

例 20.79. 如何创建和启动存储池

以下示例在 /mnt 目录中创建和启动一个名为 vdisk 的存储池：

```
# virsh pool-create-as --name vdisk --type dir --target /mnt

Pool vdisk created
```

20.29.6.5. 定义存储池

virsh pool-define-as <name> 命令创建，但没有启动，来自给定的原始参数的池对象名称。这个命令接受以下选项：

- --print-xml - 显示 XML 文件的内容，但不定义或创建存储池
- --type 类型定义 存储池类型。有关您可以使用的类型，请参阅 第 20.29.4 节 “列出可用存储池”。
- --source-host hostname - 为底层存储提供主机物理机器
- --source-path path - 底层存储的位置
- --source-dev devicename - 底层存储的设备
- --source-name sourcename - 源底层存储的名称
- --source-format 格式 - 源底层存储的格式
- --target targetname - 底层存储的目标

如果指定了 --print-xml，则将打印池对象的 XML，而不创建或定义池。否则，池需要构建指定的类型。对于需要 类型 的所有存储池命令，池类型必须以逗号分隔。有效的池类型包括：
dir、fs、netfs、逻辑、磁盘、iscsi、scsi、mpath、rbd、sheepdog 和 gluster。

> 例 20.80. 如何定义存储池
>
> 以下示例定义了一个名为 vdisk 的存储池，但不启动它。这个命令运行后，使用 virsh pool-start 命令激活存储池：

```
# virsh pool-define-as --name vdisk --type dir --target /mnt

Pool vdisk defined
```

20.29.6.6. 启动存储池

virsh pool-start pool 命令启动指定的存储池，这之前已定义但不活跃。此命令也可以将 UUID 用于存储池以及池的名称。

例 20.81. 如何启动存储池

以下示例启动您在 **例 20.78 “如何从 XML 文件创建存储池”** 中构建的 **vdisk** 存储池：

```
# virsh pool-start vdisk

Pool vdisk started
```

要验证池是否已运行 **virsh pool-list --all** 命令，并确认其状态是否活跃，如 **例 20.74 “如何列出所有存储池”** 所示。

20.29.6.7. 自动启动存储池

virsh pool-autostart pool 命令可让存储池在引导时自动启动。此命令需要池名称或 UUID。要禁用 **pool-autostart** 命令，请使用 命令中的 **--disable** 参数。

例 20.82. 如何自动启动存储池

以下示例自动启动您在 **例 20.78 “如何从 XML 文件创建存储池”** 中创建的 **vdisk** 存储池：

```
# virsh pool-autostart vdisk

Pool vdisk autostarted
```

20.29.7. 停止和删除存储池

virsh pool-destroy pool 命令停止存储池。停止后，**libvirt** 不会管理池，但池中包含的原始数据不会改变，之后可使用 **pool-create** 命令进行恢复。

例 20.83. 如何停止存储池

以下示例停止您在 例 20.78 “如何从 XML 文件创建存储池” 中构建的 vdisk 存储池：

```
# virsh pool-destroy vdisk

Pool vdisk destroyed
```

virsh pool-delete pool 命令会破坏指定存储池使用的资源。请务必注意，此操作不可恢复且不可恢复。但是，在此命令后池结构仍然存在，准备好接受创建新的存储卷。

例 20.84. 如何删除存储池

以下示例删除您在 例 20.78 “如何从 XML 文件创建存储池” 中构建的 vdisk 存储池。

```
# virsh pool-delete vdisk

Pool vdisk deleted
```

virsh pool-undefine pool 命令取消定义不活跃池的配置。

例 20.85. 如何取消定义存储池

以下示例取消定义您在 例 20.78 “如何从 XML 文件创建存储池” 中构建的 vdisk 存储池。这使得您的存储池成为临时的。

```
# virsh pool-undefine vdisk

Pool vdisk undefined
```

20.29.8. 为池创建 XML 转储文件

virsh pool-dumpxml 池 命令返回有关指定存储池对象的 XML 信息。使用 选项 **--inactive** 转储在下次启动时将使用的配置，而不是当前池配置。

例 20.86. 如何检索存储池的配置设置

以下示例检索您在 例 20.78 “如何从 XML 文件创建存储池” 中构建的 vdisk 存储池的配置设置。运行该命令后，配置文件会在终端中打开：

```
# virsh pool-dumpxml vdisk
<pool type="dir">
 <name>vdisk</name>
 <target>
        <path>/var/lib/libvirt/images</path>
 </target>
</pool>
```

20.29.9. 编辑存储池的配置文件

pool-edit pool 命令可打开指定的存储池的 XML 配置文件进行编辑。

这个方法是唯一在应用前进行错误检查时应该用来编辑 XML 配置文件的方法。

例 20.87. 如何编辑存储池的配置设置

以下示例编辑您在 例 20.78 “如何从 XML 文件创建存储池” 中构建的 vdisk 存储池的配置设置。运行该命令后，配置文件会在默认编辑器中打开：

```
# virsh pool-edit vdisk
<pool type="dir">
 <name>vdisk</name>
 <target>
        <path>/var/lib/libvirt/images</path>
 </target>
</pool>
```

20.30. 存储卷命令

这部分论述了创建、删除和管理存储卷的命令。创建存储卷需要至少一个存储池。有关如何创建存储池的示例，请参阅 例 20.78 “如何从 XML 文件创建存储池” 以了解有关存储池的信息，请参阅 第 13.2 节 “使用存储池”。有关存储卷的详情，请参考 第 13.3 节 “使用存储卷”。

20.30.1. 创建存储卷

virsh vol-create-from 池 文件 **vol** 命令可创建一个卷，并将另一个卷用作输入。这个命令需要存储池名称或存储池 UUID，并且接受以下参数和选项：

- --pool 字符串 - 需要 - 包含存储池的名称或存储池的 UUID（将附加到存储卷）。这个存储池不必是与您要用来将这个新存储卷关联的存储卷关联的存储池。

- --file string - 必需 - 包含包含存储卷参数的 XML 文件的名称。

- --vol string - 必需 - 包含您要用来基础这个新存储卷的存储卷的名称。

- --inputpool string - 可选 - 允许您命名与您要用作新存储卷的输入关联的存储池。

- --prealloc-metadata - 可选 - 为新存储卷预先分配元数据（用于 qcow2 而不是完整分配）。

有关示例，请参阅 第 13.3.2 节 “创建存储卷”。

20.30.2. 从参数创建存储卷

virsh vol-create-as 池名称 capacity 命令可从一组参数创建卷。pool 参数包含要在其中创建卷的存储池的名称或 UUID。此命令采用以下所需参数和选项：

- [--pool] 字符串 - 必需 - 包含关联的存储池的名称。

- [--name] string - 必需 - 包含新存储卷的名称。

- [- capacity] 字符串 - 必需 - 包含存储卷的大小，以整数表示。除非指定，则默认为 字节。分别使用后缀 b、k、M、G、T 表示字节、kilobyte、Mabyte、Gigabyte 和 terabyte。

- --allocation string - 可选 - 包含初始分配大小，以整数表示。除非指定，则默认为 字节。

- --format string - 可选 - 包含文件格式类型。可接受的类型包括：raw、bochs、qcow、qcow2、qed、host_device 和 vmdk。但是，这些只适用于基于文件的存储池。默认情况下，使用的 qcow 版本为版本 3。如果要更改版本，请参阅 第 23.19.2 节 “设置目标元素”。

- **--backing-vol 字符串 - 可选 - 包含后备卷。如果您要生成快照，则将使用此快照。**

- **--backing-vol-format string - 可选 - 包含后备卷的格式。如果您要生成快照，则将使用此快照。**

- **--prealloc-metadata - 可选 - 允许您预先分配元数据（用于 qcow2 而不是全分配）。**

例 20.88. 如何从一组参数创建存储卷

以下示例创建一个名为 vol-new 的 100MB 存储卷。它包含您在 例 20.78 “如何从 XML 文件创建存储池” 中创建的 vdisk存储池：

```
# virsh vol-create-as vdisk vol-new 100M


vol vol-new created
```

20.30.3. 从 XML 文件创建存储卷

virsh vol-create 池 file 命令从 XML 文件创建新存储卷，其中包含存储卷参数。

例 20.89. 如何从现有的 XML 文件创建存储卷

以下示例将基于 vol-new.xml 文件创建一个存储卷，如下所示：

```
<volume>
 <name>vol-new</name>
 <allocation>0</allocation>
 <capacity unit="M">100</capacity>
 <target>
  <path>/var/lib/libvirt/images/vol-new</path>
  <permissions>
   <owner>107</owner>
   <group>107</group>
            <mode>0744</mode>
            <label>virt_image_t</label>
          </permissions>
        </target>
</volume>
```

存储卷与存储池 vdisk 关联。到镜像的路径为 /var/lib/libvirt/images/vol-new ：

```
# virsh vol-create vdisk vol-new.xml

vol vol-new created
```

20.30.4. 克隆存储卷

virsh vol-clone vol-name new-vol-name 命令克隆现有的存储卷。虽然可以使用 virsh vol-create-from 命令，但不建议克隆存储卷。该命令接受 --pool 字符串 选项，它允许您指定与新存储卷关联的存储池。vol 参数是源存储卷的名称或密钥或者路径，name 参数则引用新存储卷的名称。有关详情请参考 第 13.3.2.1 节 “使用 virsh 创建存储卷”。

例 20.90. 如何克隆存储卷

以下示例将名为 vol-clone 的存储卷克隆到 名为 vol-clone 的新卷：

```
# virsh vol-clone vol-new vol-clone

vol vol-clone cloned from vol-new
```

20.31. 删除存储卷

virsh vol-delete vol pool 命令删除给定卷。该命令需要卷所在存储池的名称或 UUID，以及存储卷的名称。如果卷名是要删除的卷的密钥或路径，也可以使用要删除的卷。

例 20.91. 如何删除存储卷

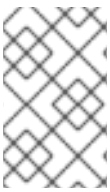
以下示例删除了一个名为 new-vol 的存储卷，其中包含存储池 vdisk ：

```
# virsh vol-delete new-vol vdisk

vol new-vol deleted
```

20.32. 删除存储卷的内容

virsh vol-wipe vol 池 命令可擦除卷，以确保之前卷中的数据无法在以后阅读。该命令需要一个 --pool 池，它是卷所在存储池的名称或 UUID，以及 池，即要擦除的卷名称或路径的名称。请注意，可以使用参数 --algorithm 并使用以下支持的算法之一选择不同的 wiping 算法而不是重写卷：

- *零 - 1-pass all zeroes*
- *nnsa - 4-pass NNSA 策略 Letter NAP-14.1-C(XVI-8)，用于清理可移动且不可删除的硬盘： random x2, 0x00, verify.*
- *DoD - 4-pass DoD 5220.22-M 部分 8-306 部分清理可移动且不可删除的磁盘： random, 0x00, 0xff, verify.*
- *BSI - 信息技术领域德国安全中心推荐 9-pass 方法(http://www.bsi.bund.de)：0XFF, 0xfe, 0xfd, 0xfb, 0xf7, 0xef, 0xdf, 0xbf, 0x7f.*
- *gutmann - Gutmann 白皮书中描述的规范 35-pass 序列。*
- *schneier - Bruce Schneier 描述的 7-pass 方法，"Applied Cryptography"(1996)：0x00, 0xff, 随机 x5.*
- *pfitzner7 - Roy Pfitzner 的 7-random-pass 方法：随机 x7*
- *pfitzner33 - Roy Pfitzner 的 33-random-pass 方法：随机 x33.*
- *Random - 1-pass pattern: random.s*



注意

算法的可用性可能会受主机上安装的"清理"二进制版本的限制。

例 20.92. 如何删除存储卷的内容（如何擦除存储卷）

以下示例擦除存储卷 new-vol 的内容，它关联有存储池 vdisk ：

```
# virsh vol-wipe new-vol vdisk

vol new-vol wiped
```

20.33. 将存储卷信息转储到 XML 文件

virsh vol-dumpxml vol 命令获取卷信息，创建包含内容内容的 **XML** 文件，并将其输出到 **stdout** 流中设置的设置。另外，您可以使用 **--pool** 选项提供关联的存储池的名称。

例 20.93. 如何转储存储卷的内容

以下示例将名为 **vol-new** 的存储卷的内容转储到 **XML** 文件中：

```
# virsh vol-dumpxml vol-new
```

20.34. 列出卷信息

virsh vol-info vol 命令列出了有关给定存储卷的基本信息。您必须提供存储卷名称、密钥或路径。该命令也接受 **--pool** 选项，您可以在其中指定与存储卷关联的存储池。您可以提供池名称，或者提供 **UUID**。

例 20.94. 如何查看有关存储卷的信息

以下示例检索有关名为 **vol-new** 的存储卷的信息。运行这个命令时，您应该将存储卷的名称改为您的存储卷的名称：

```
# virsh vol-info vol-new
```

virsh vol-list pool 命令列出与给定存储池关联的所有卷。此命令需要存储池的名称或 **UUID**。**--details** 选项指示 **virsh** 额外显示卷类型和容量可用信息。

例 20.95. 如何显示与存储卷关联的存储池

以下示例列出所有与存储池 **vdisk** 关联的存储卷：

```
# virsh vol-list vdisk
```

20.35. 检索存储卷信息

virsh vol-pool vol 命令返回给定存储卷的池名称或 UUID。默认情况下，返回存储池名称。如果使用 **--uuid** 选项，则返回池 UUID。命令需要存储卷的密钥或者路径来返回请求的信息。

例 20.96. 如何显示存储卷的名称或 UUID

以下示例检索在路径 **/var/lib/libvirt/images/vol-new** 中找到的存储卷的名称：

```
# virsh vol-pool /var/lib/libvirt/images/vol-new

vol-new
```

vol-path --pool pool-or-uuid vol-name-or-key 命令返回给定卷的路径。命令需要 **--pool pool-or-uuid**，这是卷所在存储池的名称或 UUID。它还需要 **vol-name-or-key**，这是请求路径的卷的名称或密钥。

vol-name vol-key-or-path 命令返回给定卷的名称，其中 **vol-key-or-path** 是要返回卷的密钥或路径。

vol-key --pool pool-or-uuid vol-name-or-path 命令返回给定卷的卷密钥，其中 **--pool pool-or-uuid** 是卷的名称或 UUID，而 **vol-name-or-path** 是卷的名称或路径返回卷的密钥。

20.36. 显示每个虚拟机信息

20.36.1. 显示客户机虚拟机

使用 **virsh** 显示活跃客户机虚拟机列表及其当前状态：

```
# virsh list
```

其它可用选项包括：

- **--all** - 列出所有 guest 虚拟机。例如：

```
# virsh list --all
 Id Name                 State
----------------------------------
  0 Domain-0             running
  1 Domain202            paused
  2 Domain010            shut off
  3 Domain9600           crashed
```

注意

如果在运行 **virsh list --all** 时不显示任何结果，则可能是因为您不会以 **root** 用户身份创建虚拟机。

virsh list --all 命令识别以下状态：

- **Running** - 状态为 CPU 上当前活跃的 guest 虚拟机。
- **idle** - idle 状态表示客户机虚拟机处于空闲状态，可能无法运行或可运行。当客户机虚拟机在 I/O（传统等待状态）上处于睡眠状态时会出现这种情况，因为它没有其他操作。
- **paused** - 当客户机虚拟机暂停时，它会消耗内存和其他资源，但不符合从虚拟机监控程序调度 CPU 资源的条件。在 **virt-manager** 或 **virsh suspend** 命令中使用 **paused** 按钮后，暂停的状态发生。
- **关闭** - 处于关闭状态，用于关闭过程中的客户机虚拟机。客户机虚拟机发送了一个关机信号，应在正常停止其操作过程中。这可能不会用于所有虚拟客户机操作系统；一些操作系统不响应这些信号。
- **关闭** - 关闭状态 表示客户机虚拟机没有运行。当 guest 虚拟机完全关闭或尚未启动时，这可能会导致。
- **crashed** - crashed 状态表示客户机虚拟机已经崩溃，且只能在 guest 虚拟机被配置为崩溃时发生。
- **pmsuspended** - 客户机电源管理已暂停。

- **--inactive - 列出已定义但当前未激活的客户机虚拟机。这包括 关闭 并 崩溃 的机器。**

- **--managed-save - 启用了受管保存状态的客户机将列为 已保存的。请注意，要使用这个选项过滤 guest，还需要使用 --all 或 --inactive 选项。**

- **--name - 命令列出了客户机的名称，而不是默认的表格式。这个选项与 --uuid 选项相互排斥，该选项只打印客户端 UUID 列表，使用 --table 选项，它决定了应当使用表风格输出。**

- **-- title - 同时列出 guest 标题字段，其中通常包含 guest 的简短描述。这个选项必须与默认的(--table)输出格式一起使用。例如：**

```
$ virsh list --title

Id    Name                           State    Title
----------------------------------------------------------------------
0     Domain-0                       running  Mailserver1
2     rhelvm                         paused
```

- **--persistent - 仅包括持久的 guest 包含在列表中。使用 --transient 参数列出临时 guest。**

- **--with-managed-save - 配置了受管保存的客户机列表。要列出没有虚拟机的虚拟机，请使用 -- without-managed-save 选项。**

- **--state-running - 仅列出正在运行的 guest.同样，使用 --state-paused 用于暂停的 guest，--state-shutoff 用于关闭的 guest，--state-other 将所有状态列为回退。**

- **--autostart - 只列出自动启动 guest。要列出禁用了这个功能的客户端，请使用 --no-autostart 参数。**

- **--with-snapshot - 列出可以列出快照映像的客户机。要过滤没有快照的 guest，请使用 --without-snapshot 选项。**

20.36.2. 显示虚拟 CPU 信息

使用 virsh 显示客户机虚拟机中的虚拟 CPU 信息：

```
# virsh vcpuinfo {domain-id, domain-name or domain-uuid}
```

virsh vcpuinfo 输出示例：

```
# virsh vcpuinfo guest1
VCPU:           0
CPU:            2
State:          running
CPU time:       7152.4s
CPU Affinity:   yyyy

VCPU:           1
CPU:            2
State:          running
CPU time:       10889.1s
CPU Affinity:   yyyy
```

20.36.3. 将 vCPU 固定到主机物理机器的 CPU

virsh vcpupin 命令将虚拟 CPU 分配给物理 CPU。

```
# virsh vcpupin guest1
VCPU: CPU Affinity
----------------------------------
  0: 0-3
  1: 0-3
```

vcpupin 命令可使用以下参数：

- **--vCPU 需要 vcpu 号**
- **[--cpulist] 字符串 列出要设置的主机物理机器的 CPU 编号，或省略用于查询的选项**
- **--config 在下次引导时影响**
- **--live 会影响正在运行的客户机虚拟机**
- **--current 对当前客户机虚拟机状态的影响**

20.36.4. 显示有关给定域的虚拟 CPU 计数的信息

virsh vcpucount 命令需要一个 域名 或域 ID

```
# virsh vcpucount guest1
maximum      config         2
maximum      live           2
current      config         2
current      live           2
```

vcpucount 可以采用以下参数：

- **--maximum get maximum on vcpus**
- **--active get number 当前 active vcpus**
- **--live 从运行的客户机虚拟机获取值**
- **--config get value 用于下一次引导时**
- **--current 根据当前客户机虚拟机状态获取值**
- **返回的 --guest 数量是从客户机的视角获得**

20.36.5. 配置虚拟 CPU 关联性

使用物理 CPU 配置虚拟 CPU 的关联性：

```
# virsh vcpupin domain-id vcpu cpulist
```

domain-id 参数是 guest 虚拟机的 ID 号或名称。

vcpu 参数表示分配给客户机虚拟机的虚拟 CPU 数量。必须提供 vcpu 参数。

cpulist 参数是一个以逗号分开的物理 CPU 标识符号列表。cpulist 参数决定 VCPU 可以运行的物理 CPU。

其他参数（如 --config 会对下次引导造成影响），-- live 则会影响正在运行的 guest 虚拟机，--current 会影响当前的客户机虚拟机状态。

20.36.6. 配置虚拟 CPU 数

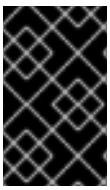
使用此命令更改客户机虚拟机中活跃的虚拟 CPU 数量。默认情况下，这个命令适用于活跃的 guest 虚拟机。要在下次启动虚拟客户机时更改将使用的不活跃设置，请使用 --config 标志。要使用 virsh 修改分配给客户机虚拟机的 CPU 数量：

```
# virsh setvcpus {domain-name, domain-id or domain-uuid} count [[--config] [--live] | [--current]] [--maximum] [--guest]
```

例如：

```
# virsh setvcpus guestVM1 2 --live
```

将 vCPU 数量设置为两个 guestVM1，这将在 guestVM1 运行时执行此操作。



重要

Red Hat Enterprise Linux 7 不支持热拔 vCPU。

计数值可能会受主机、虚拟机监控程序或来自客户机虚拟机原始描述的限制。

如果指定了 --config 标志，则更改将更改为 guest 虚拟机的存储 XML 配置，并且仅在启动 guest 时生效。

如果指定了 --live，guest 虚拟机必须处于活动状态，且更改会立即发生。这个选项将允许热插 vCPU。如果虚拟机监控程序支持，可以将 --config 和 --live 标志一起指定。

如果指定了 --current，该标志会影响当前的客户机虚拟机状态。

如果没有指定标记，则假定 --live 标志。如果 guest 虚拟机未激活，该命令将会失败。另外，如果没有指定标志，它适用于虚拟机监控程序，是否也假定 --config 标志。这决定了 XML 配置是否已调整，以永久保留更改。

--maximum 标志控制下次虚拟机引导时可热插的最大虚拟 CPU 数量。因此，它只能与 --config 标志一起使用，不能与 --live 标志一起使用。

请注意，数 不能超过分配给客户机虚拟机的 CPU 数量。

如果指定了 --guest，该标志会修改当前客户机虚拟机的 CPU 状态。

20.36.7. 配置内存分配

使用 virsh 修改客户机虚拟机的内存分配：

```
# virsh setmem {domain-id or domain-name} count
```

例如：

```
# virsh setmem vr-rhel6u1-x86_64-kvm --kilobytes 1025000
```

您必须以 KB 为单位指定计数。新计数值不能超过您为客户机虚拟机指定的数量。低于 64 MB 的值不太可能用于大多数客户机虚拟机操作系统。较高最大内存值不会影响活跃的客户机虚拟机。如果新值小于可用内存，它将缩小可能会导致客户机虚拟机崩溃。

这个命令有以下选项

- domain - 由域名、id 或 uuid 指定
- size - 确定新内存大小，作为缩放整数。默认单位是 KiB，但可以指定不同的单位：

  有效内存单元包括：

- b 或 bytes （以字节为单位）
- KB 代表 KB（10^3 或 1,000 字节块）
- k 或 KiB 用于 kibibytes（2^{10} 或 1024 字节）
- MB 表示 MB（10^6 或 1,000,000 字节块）
- M 或 MiB 代表兆字节（^{220} 或 1,048,576 字节）
- GB 表示千兆字节（10^9 或 1,000,000 字节块）
- g 或 GiB 用于千兆字节（2^{30} 或块 1,073,741,824 字节）
- TB 代表 TB（10^{12} 或 1,000,000,000 字节块）
- T 或 TiB 代表 tebibytes（2^{40} 或块 1,099,511,627,776 字节）

请注意，所有值都将按 libvirt 舍入到最接近的基千字节，并且也可进一步取整为管理程序支持的粒度。有些虚拟机监控程序还可强制实施最少 4000KiB（或 4000 x 2^{10} 或 4,096,000 字节）。这个值的单位由可选属性 内存单元 决定，它默认为 kibibytes(KiB)，以测量给出的值乘以 2^{10} 或块 1024 字节。

- --config - 命令在下次引导时生效
- --live - 命令控制正在运行的客户机虚拟机的内存
- --current - 命令控制当前客户机虚拟机的内存

20.36.8. 更改域的内存分配

virsh setmaxmem 域 大小 --config --live --current 命令允许为客户机虚拟机设置最大内存分配，如下所示：

```
# virsh setmaxmem guest1 1024 --current
```

可为最大内存指定的大小是一个缩放整数，默认情况下以 kibibytes 表示，除非提供了支持的后缀。此命令可使用以下参数：

- --config - 在下次引导时影响
- --live - 控制正在运行的虚拟客户机虚拟机的内存，提供虚拟机监控程序支持此操作，因为不是所有虚拟机监控程序都允许对最大内存限值进行实时更改。
- --current - 控制当前客户端虚拟机的内存

20.36.9. 显示客户机虚拟机块设备信息

使用 virsh domblkstat 命令显示正在运行的客户机虚拟机的块设备统计信息。使用 --human 以更用户友好的方式显示统计数据。

```
# virsh domblkstat GuestName block-device
```

20.36.10. 显示客户机虚拟机网络设备信息

使用 virsh domifstat 命令显示正在运行的客户机虚拟机的网络接口统计信息。

```
# virsh domifstat GuestName interface-device
```

20.37. 管理虚拟网络

这部分论述了使用 virsh 命令管理虚拟网络。列出虚拟网络：

```
# virsh net-list
```

这个命令会生成类似如下的输出：

```
# virsh net-list
Name                 State      Autostart
-----------------------------------------
default              active     yes
vnet1                active     yes
vnet2                active     yes
```

查看特定虚拟网络的网络信息：

```
# virsh net-dumpxml NetworkName
```

以 XML 格式显示有关指定虚拟网络的信息：

```
# virsh net-dumpxml vnet1
<network>
  <name>vnet1</name>
  <uuid>98361b46-1581-acb7-1643-85a412626e70</uuid>
  <forward dev='eth0'/>
  <bridge name='vnet0' stp='on' forwardDelay='0' />
  <ip address='192.168.100.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.100.128' end='192.168.100.254' />
    </dhcp>
  </ip>
</network>
```

其他用来管理虚拟网络的 virsh 命令包括：

- virsh net-autostart network-name :将 network-name 标记为在 libvirt 守护进程启动时自动启动。--disable 选项取消标记 network-name。
- virsh net-create XMLfile :使用来自现有文件的 XML 定义，启动新的（转换）网络。
- virsh net-define XMLfile :使用现有文件中的 XML 定义来定义新网络，而不启动它。
- virsh net-destroy network-name :销毁指定为 network-name 的网络。

- virsh net-name networkUUID :将指定的 网络UUID 转换为网络名称。
- virsh net-uuid network-name :将指定的 network-name 转换为网络 UUID。
- virsh net-start nameOfInactiveNetwork :启动不活动的网络。
- virsh net-undefine nameOfInactiveNetwork :删除网络的不活跃 XML 定义。这对网络状态没有影响。如果在执行此命令时域正在运行，则网络将继续运行。但是，网络会变得临时而不是持久。

libvirt 能够定义可由域使用并链接到实际网络设备的虚拟网络。有关此功能的详细信息，请参阅 libvirt 上游网站 的文档。虚拟网络的许多命令与用于域的命令类似，但命名虚拟网络的方式就是其名称或 UUID。

20.37.1. 自动启动虚拟网络

virsh net-autostart 命令将在客户端虚拟机引导时自动启动虚拟网络。

```
# virsh net-autostart network [--disable]
```

该命令接受 --disable 选项，该选项禁用 autostart 命令。

20.37.2. 从 XML 文件创建虚拟网络

virsh net-create 命令从 XML 文件创建虚拟网络。要获得 libvirt 使用的 XML 镜像格式的描述，请参阅 libvirt 上游网站。这个命令文件是 XML 文件的路径。要从 XML 文件创建虚拟网络，请运行：

```
# virsh net-create file
```

20.37.3. 从 XML 文件定义虚拟网络

virsh net-define 命令定义 XML 文件的虚拟网络，只定义网络，而不是实例化。

```
# virsh net-define file
```

20.37.4. 停止虚拟网络

virsh net-destroy 命令销毁（停止）由其名称或 **UUID** 指定的给定虚拟网络。这会立即生效。要停止指定的 网络网络。

```
# virsh net-destroy network
```

20.37.5. 创建转储文件

virsh net-dumpxml 命令会以 **XML** 转储的形式输出到指定虚拟网络的 **stdout**。如果指定了 **--inactive**，则物理功能不会扩展到其关联的虚拟功能中。

```
# virsh net-dumpxml network [--inactive]
```

20.37.6. 编辑虚拟网络的 XML 配置文件

以下命令编辑网络的 **XML** 配置文件：

```
# virsh net-edit network
```

用于编辑 **XML** 文件的编辑器可由 **$VISUAL** 或 **$EDITOR** 环境变量提供，默认为 **vi**。

20.37.7. 获取有关虚拟网络的信息

virsh net-info 返回 网络对象 的基本信息。

```
# virsh net-info network
```

20.37.8. 列出虚拟网络的信息

virsh net-list 命令返回活动网络的列表。如果指定了 **--all**，这也将包括定义但不活跃的网络。如果只指定 **--inactive**，则将列出非活动。您可能还想通过 **--persistent** 过滤返回的网络来列出永久网络，使用 **--transient** 列出临时临时网络，**--autostart** 列出启用自动启动的临时网络，使用 **--no-autostart** 列出禁用自动启动的网络。

备注：与旧的服务器通信时，这个命令强制使用一系列带有固有竞争条件的 **API** 调用，其中池可能无法列出，或者在列表被收集时更改的时间超过一次。较新的服务器没有这个问题。

要列出虚拟网络，请运行：

```
# virsh net-list [--inactive | --all] [--persistent] [<--transient>] [--autostart] [<--no-autostart>]
```

20.37.9. 将网络 UUID 转换为网络名称

virsh net-name 命令会将网络 UUID 转换为网络名称。

```
# virsh net-name network-UUID
```

20.37.10. 将网络名称转换为网络 UUID

virsh net-uuid 命令可将网络名称转换为网络 UUID。

```
# virsh net-uuid network-name
```

20.37.11. 启动之前定义的 Inactive 网络

virsh net-start 命令启动（之前定义的）非活动网络。

```
# virsh net-start network
```

20.37.12. 取消定义 Inactive Network 的配置

virsh net-undefine 命令取消定义不活跃网络的配置。

```
# virsh net-undefine network
```

20.37.13. 更新现有网络定义文件

```
# virsh net-update network directive section XML [--parent-index index] [[--live] [--config] | [--current]]
```

virsh net-update 命令通过向部分发出 以下指令 之一来更新现有网络定义的指定部分：

- **add-first**

- *add-last 或 add （这些是同义）*
- *删除*
- *修改*

部分 可以是以下之一：

- *bridge*
- *domain*
- *ip*
- *ip-dhcp-host*
- *ip-dhcp-range*
- *forward*
- *转发接口*
- *forward-pf*
- *portgroup*

- *dns-host*
- *dns-txt*
- *dns-srv*

每个部分通过 XML 元素层次结构的串联命名，这导致了更改的元素。例如，ip-dhcp-host 更改 <一个主机> 元素，它包含在网络的 <ip> 元素中的 <dhcp> 元素中。

XML 是被更改类型的完整 XML 元素的文本（例如：<host mac="00:11:22:33:44:55' ip='1.2.3.4'/>），或者包含完整 XML 元素的文件的名称。通过查看提供的文本的第一个字符 - 如果第一个字符是 <，则它是 XML 文本，如果第一个字符不是 >，则这是包含要使用的 xml 文本的文件名称。parent -index 选项用于指定所请求元素的多个父元素（基于 0）。

例如，dhcp <主机> 元素可以是网络中的任意多个 <ip> 元素之一；如果不提供父索引，则会选择最适当的 <ip> 元素（通常是一个已具有 <dhcp> 元素），但如果给出了 --parent-index，则特定的 <ip> 实例将会获得修改。如果指定了 --live，会影响正在运行的网络。如果指定了 --config，则会影响持久网络的下一次启动。如果指定了 --current，会影响当前网络状态。可以给定 --live 和 --config 标志，但 --current 是独占的。不指定任何标志与指定 --current 相同。

20.37.14. 使用 virsh 迁移客户机虚拟机

有关使用 virsh 迁移的信息，请参考使用 virsh 的 Live KVM 迁移部分。 第 15.5 节 “使用 virsh 进行实时 KVM 迁移”

20.37.15. 为客户机虚拟机设置静态 IP 地址

如果客户机虚拟机被配置为从 DHCP 获取其 IP 地址，但您仍需要它具有可预测的静态 IP 地址，则可使用以下步骤修改 libvirt 使用的 DHCP 服务器配置。这个步骤需要您知道客户机接口的 MAC 地址，以便进行这个更改。因此，您需要在客户机创建之后执行操作，或在创建客户机之前决定 guest 的 MAC 地址，然后在创建客户机虚拟机时手动设置相同的地址。

另外，您应该注意，这个过程只适用于连接到 libvirt 虚拟网络的客户机接口，其转发模式为 "nat"、"route"，或者根本没有转发模式。如果网络配置了 forward mode="bridge" 或 "hostdev"，则此过程将无法正常工作。在这样的情形中，DCHP 服务器位于网络中的其他位置，因此不受 libvirt 的控

制。在这种情况下，需要在远程 DHCP 服务器上进行静态 IP 条目。要做到这一点，查看服务器提供的文档。

过程 20.5. 设置静态 IP 地址

此过程在主机物理机器上执行。

1. 检查客户机 XML 配置文件

通过运行 virsh domiflist guest1 命令显示客户机的网络配置设置。替换您的虚拟机的名称，用 guest1 代替。此时会显示一个表。查看 Source 列。这是您的网络的名称。在本例中，网络名为 default。这个名称将用于剩余的步骤和 MAC 地址。

```
# virsh domiflist guest1
Interface  Type       Source     Model       MAC
-------------------------------------------------------
vnet4      network    default    virtio      52:54:00:48:27:1D
```

2. 验证 DHCP 范围

您所设置的 IP 地址必须在为网络指定的 dhcp 范围内。另外，它还不得与网络上的任何其他现有静态 IP 地址冲突。要检查可用地址以及已使用的地址范围，请在主机中使用以下命令：

```
# virsh net-dumpxml default | egrep 'range|host\ mac'

<range start='198.51.100.2' end='198.51.100.254'/>
<host mac='52:54:00:48:27:1C:1D' ip='198.51.100.2'/>
```

您看到的输出将与示例不同，您可能会看到更多行和多个主机 mac 行。每个 guest 静态 IP 地址都将有一行。

3. 设置静态 IP 地址

在主机计算机上运行以下命令，将 default 替换为网络的名称。

```
# virsh net-update default add ip-dhcp-host '<host mac="52:54:00:48:27:1D"
ip="198.51.100.3"/>' --live --config
```

live 选项允许此更改立即发生，-- config 选项则可永久保留更改。在您使用有效的 IP 和 MAC 地址时，此命令还将适用于尚未创建的客户机虚拟机。MAC 地址应该是有效的单播 MAC 地址（由 分隔的6 位十六进制对，第一个数字对为偶数）；当 libvirt 创建新的随机 MAC 地址时，它会将 52:54:00 用于前三个数字对，建议遵照此惯例。

4. 重启接口（可选）

如果 guest 虚拟机当前正在运行，则需要强制客户端虚拟机重新请求 DHCP 地址。如果 guest 未运行，您下次启动时新的 IP 地址将实施。要重启接口，在主机机器中输入以下命令：

```
# virsh domif-setlink guest1 52:54:00:48:27:1D down
# sleep 10
# virsh domif-setlink guest1 52:54:00:48:27:1D up
```

该命令使客户机虚拟机的操作系统认为以太网电缆没有被插入，然后在十秒后重新插入。sleep 命令非常重要，因为许多 DHCP 客户端允许简短断开电缆，而无需重新请求 IP 地址。十秒足够长，以便 DHCP 客户端忘记旧的 IP 地址，并在执行命令后请求一个新地址。如果因某种原因命令失败，则必须从客户机操作系统的管理界面重置 guest 的接口。

20.38. INTERFACE 命令

以下命令可操作主机接口，因此不应从客户机虚拟机中运行。这些命令应该从主机物理计算机上的终端运行。

警告

只有在机器禁用 NetworkManager 服务并使用 network 服务时，才会支持本节中的命令。

通常，这些主机接口可由客户机虚拟机 <接口> 元素（如系统创建桥接接口）中的名称使用，但没有任何主机接口与所有特定虚拟机配置 XML 关联。主机接口的许多命令与用于客户机虚拟机的命令类似，命名接口的方式就是其名称或其 MAC 地址。但是，当使用 iface 参数的 MAC 地址时，仅当该地址是唯一的时（如果接口和桥接共享相同的 MAC 地址，这通常是这种情况，那么使用该 MAC 地址会导致因为不确定性而造成错误，且您必须转为使用名称）。

20.38.1. 通过 XML 文件定义和启动主机物理接口

virsh iface-define 文件 命令从 XML 文件定义主机接口。这个命令只定义接口，也不会启动该接口。

```
# virsh iface-define iface.xml
```

要启动已经定义的接口，请运行 iface-start 接口，其中 interface 是接口名称。

20.38.2. 为主机接口编辑 XML 配置文件

命令 virsh iface-edit 界面 编辑主机接口的 XML 配置文件。这是编辑 XML 配置文件 的唯一 推荐方法。（有关这些文件的更多信息，请参阅 第 23 章 操作域 XML。）

20.38.3. 列出主机接口

virsh iface-list 显示活动主机接口列表。如果指定了 --all，此列表还包含已定义但处于非活动状态的接口。如果只指定 --inactive，则将列出不活动接口。

20.38.4. 将 MAC 地址转换为接口名称

virsh iface-name 接口 命令将主机接口 MAC 地址转换为接口名称，提供的 MAC 地址在主机的接口中是唯一的。此命令需要 接口，即接口的 MAC 地址。

virsh iface-mac interface 命令将主机的接口名称转换为 MAC 地址（在这种情况下，接口 ）是接口名称。

20.38.5. 停止和取消定义特定主机物理接口

virsh iface-destroy 接口 命令销毁（停止）给定主机接口，这与在主机上运行 virsh if-down 类似。此命令将禁用该接口，使其生效。

要取消定义接口，请使用 virsh iface-undefine interface 命令以及接口名称。

20.38.6. 显示主机配置文件

virsh iface-dumpxml 接口 --inactive 命令显示主机接口信息，作为 XML 转储到 stdout。如果指定了 --inactive 参数，则输出会反映在下次启动时将使用的接口的持久性状态。

20.38.7. 创建网桥设备

virsh iface-bridge 命令创建名为 bridge 的桥接设备，将现有网络设备 接口 附加到新网桥，这将立即开始工作，并启用了 STP，延迟为 0。

```
# virsh iface-bridge interface bridge
```

请注意，可以使用 --no-stp 选项、-- no-start 选项以及延迟的秒数来更改这些设置。接口的 IP 地址配置将移到新网桥设备中。有关关闭桥接的详情，请参考 第 20.38.8 节 “关闭桥接设备”

20.38.8. 关闭桥接设备

virsh iface-unbridge bridge --no-start 命令关闭名为 bridge 的指定网桥设备，将其底层接口释放回正常使用情况，并将所有 IP 地址配置从网桥设备移到基础设备。除非使用 --no-start 参数，否则将重启底层接口，但通常不建议重新启动。有关创建桥接的命令，请参阅 第 20.38.7 节 “创建网桥设备”。

20.38.9. 操作接口快照

virsh iface-begin 命令创建了当前主机接口设置的快照，稍后可以提交（使用 virsh iface-commit）或恢复(virsh iface-rollback)。这适用于定义和启动新主机接口时失败的情况，以及发生系统错误配置的情况。如果快照已存在，则此命令将失败，直到之前的快照已提交或恢复。如果创建快照及其最终提交或回滚期间，如果对 libvirt API 之外的主机接口进行任何外部更改，则未定义的行为将会导致。

使用 virsh iface-commit 命令声明自上次 virsh iface-begin 开始所做的所有更改工作，然后删除回滚点。如果没有使用 virsh iface-begin 启动接口快照，则此命令将失败。

使用 virsh iface-rollback 将所有主机接口设置恢复到最后一次执行 virsh iface-begin 命令的状态。如果之前没有执行 virsh iface-begin 命令，则 virsh iface-rollback 将失败。请注意，如果在运行 virsh iface-commit 之前重新启动主机物理计算机，将执行自动回滚，该自动回滚会将主机的配置恢复到执行 virsh iface-begin 的状态。如果网络配置不当更改导致主机无法撤销更改，则这很有用，但主机是节能，或者强制重新引导。

例 20.97. 使用快照的示例

定义和启动新主机接口。

```
# virsh iface-begin
# virsh iface-define eth4-if.xml
# virsh if-start eth4
```

如果某种失败并且网络停止运行，则回滚更改。

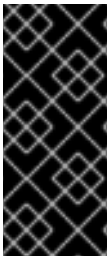
```
# virsh iface-rollback
```

如果一切正常，请提交更改。

```
# virsh iface-commit
```

20.39. 管理快照

后续部分描述可执行的操作以操作客户机虚拟机快照。快照 取客户机虚拟机的磁盘、内存和设备状态（在指定时间点），并保存它供以后使用。快照有多种，从保存操作系统镜像的"彻底"副本，以便在可能成为破坏性操作之前保存虚拟机的状态。快照通过唯一的名称来标识。有关用于表示快照属性的 XML 格式文档，请参阅 libvirt 上游网站。



重要

Red Hat Enterprise Linux 7 仅支持在客户机虚拟机暂停或关闭时创建快照。在 Red Hat Virtualization™ 中提供了运行客户机的快照（也称为 实时快照）。如需详细信息，请致电您的服务代表。

20.39.1. 创建快照

virsh snapshot-create 命令使用客户机虚拟机 XML 文件中指定的属性（如 <name> 和 <description> 元素以及 <disks>）为客户机虚拟机创建一个快照。要创建快照运行：

```
# virsh snapshot-create domain XML file [--redefine [--current] [--no-metadata] [--halt] [--disk-only] [--reuse-external] [--quiesce] [--atomic]
```

客户机虚拟机名称、id 或 uid 可用作客户机虚拟机要求。XML 要求是一个字符串，它必须至少包含名称、描述 和磁盘元素。

剩余的可选参数如下：

- --disk-only - 客户机虚拟机的内存状态不包含在快照中。
- 如果完全省略 XML 文件字符串，libvirt 将为所有字段选择一个值。新快照将变为最新状态，如 snapshot-current 所列出。此外，快照将仅包含磁盘状态，而不是包含 guest 虚拟机状态的常规系统检查点。磁盘快照比全系统检查点快，但恢复到磁盘快照可能需要 fsck 或日志重

播，因为在电源 cord 拉特拉特拉特时，磁盘状态就如同磁盘状态。请注意，混合 --halt 和 --disk-only 会丢失任何时间没有冲刷到磁盘的数据。

- --halt - 在创建快照后，使客户机虚拟机保持不活动状态。混合 --halt 和 --disk-only 会丢失任何在时间和内存状态时没有刷新到磁盘的数据。

- --redefine 指定：如果 virsh snapshot-dumpxml 生成的所有 XML 元素都有效；它可以用于将快照层次结构从一台计算机迁移到另一台计算机，为重新创建层次结构，以重新建立一个 临时的 guest 虚拟机，该虚拟机之后使用相同的名称和 UUID 重新创建，或者在快照元数据中对快照元数据（如主机特定部分）进行改变。提供此标志时，xmlfile 参数是必需的，除非也指定了 --current 标志，否则 guest 虚拟机的当前快照不会被更改。

- --no-metadata 创建快照，但所有元数据都会立即丢弃（即，libvirt 不会将快照视为当前），除非稍后使用 --redefine 来教授 libvirt 元数据。

- --reuse-external，如果使用和快照 XML 请求一个带有现有文件目标的外部快照，目标必须存在，并且会被重复使用；否则，快照将被拒绝，以避免丢失现有文件的内容。

- --quiesce libvirt 将尝试使用客户机代理冻结并取消冻结客户机虚拟机的已挂载文件系统。但是，如果客户机虚拟机没有客户机代理，则快照创建将失败。快照可以包含虚拟机的内存状态。快照必须是 external。

- -- Atomic 可使 libvirt 保证快照可以成功执行，或者没有更改失败。请注意，不是所有虚拟机监控程序都支持这种管理程序。如果没有指定此标志，那么某些虚拟机监控程序在部分操作后可能会失败，且必须使用 virsh dumpxml 来查看是否发生了任何部分更改。

快照元数据的存在将阻止尝试取消定义持久的客户机虚拟机。但是，对于 临时 客户机虚拟机，当客户机虚拟机退出运行时快照元数据会被静默丢失（无论是命令，如 销毁 还是内部客户端操作）。

20.39.2. 为当前客户机虚拟机创建快照

virsh snapshot-create-as 命令使用域 XML 文件中指定的属性（如 名称和描述 元素）为 guest 虚拟机创建一个快照。如果这些值不包含在 XML 字符串中，则 libvirt 将选择一个值。要创建快照运行：

```
# snapshot-create-as domain {[--print-xml] | [--no-metadata] [--halt] [--reuse-external]} [name]
[description] [--disk-only [--quiesce]] [--atomic] [[--memspec memspec]] [--diskspec] diskspec]
```

剩余的可选参数如下：

- --print-xml为 snapshot-create 创建适当的 XML 作为输出，而不是实际创建快照。

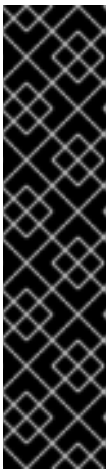
- --halt 在创建快照后，保持客户机虚拟机处于非活动状态。

- --disk-only 会创建一个不包含客户机虚拟机状态的快照。

- --memspec 可用于控制检查点是否为内部还是外部。标志为必需，后跟 [file=]name[,snapshot=type] 形式的 memspec，其中 type 可以是 none、internal 或 external。要在 file=name 中包含字面逗号，请使用第二个逗号进行转义。

- --diskspec 选项可以用来控制 --disk-only 和外部检查点如何创建外部文件。根据域 XML 中的 <disk> 元素数量，这个选项可以多次出现。每个 <diskspec> 都以磁盘[,snapshot=type][,driver=type][,file=name] 的形式。如果特定磁盘省略 --diskspec，则使用虚拟机配置中的默认行为。要在磁盘或者 file=name 中包含字面逗号，请使用第二逗号进行转义。文字 --diskspec 必须在每个 diskspec 之前，除非同时存在三个 域、name 和 description。例如，vda、snapshot=external、file=/path/to 的 diskspec 会生成以下 XML：

```
<disk name='vda' snapshot='external'>
  <source file='/path/to,new'/>
</disk>
```



重要

红帽建议使用外部快照，因为当由其他虚拟化工具处理时，它们更灵活且更可靠。要创建外部快照，请使用 virsh-create-as 命令和 --diskspec vda,snapshot=external 选项

如果没有使用这个选项，virsh 会创建内部快照，但由于缺乏稳定性和优化功能，我们不建议使用它。如需更多信息，请参阅 第 A.13 节 “使用 libvirt 创建外部快照的临时解决方案”。

- 指定 --reuse-external，域 XML 或 diskspec 选项请求一个带有现有文件目标的外部快照，

然后目标必须存在，并且会被重复使用；否则，快照将拒绝，以避免丢失现有文件的内容。

- 指定了 --quiesce，libvirt 将尝试使用客户机代理来冻结和取消暂停客户机虚拟机的已挂载文件系统。但是，如果域没有客户机代理，则快照创建将失败。目前，这需要仅传递 --disk-only。

- --no-metadata 会创建快照数据，但任何元数据都会立即丢弃（即，libvirt 不会将快照视为当前），除非稍后使用 snapshot-create 来教授 libvirt 元数据。这个标志与 --print-xml不兼容

- -- Atomic 将导致 libvirt 保证快照可以成功执行，或者没有更改失败。请注意，不是所有虚拟机监控程序都支持这种管理程序。如果没有指定此标志，那么某些虚拟机监控程序在部分操作后可能会失败，且必须使用 virsh dumpxml 来查看是否发生了任何部分更改。

警告

创建在 64 位 ARM 平台 主机上运行的 KVM 客户机快照目前无法正常工作。请注意，红帽不支持 64 位 ARM 上的 KVM。

20.39.3. 当前使用中的快照

virsh snapshot-current 命令用于查询当前正在使用的快照。

```
# virsh snapshot-current domain {[--name] | [--security-info] | [snapshotname]}
```

如果没有使用 snapshotname，则 guest 虚拟机的当前快照的快照 XML（如果有）将以输出的形式显示。如果指定了 --name，只需以输出的形式发送当前快照名称而不是完整 XML。如果提供 --security-info 的安全敏感信息将包含在 XML 中。使用 snapshotname，生成请求使现有的名为 snapshot 成为当前快照，而无需将其恢复到客户机虚拟机。

20.39.4. snapshot-edit

这个命令用于编辑当前使用的快照：

```
# virsh snapshot-edit domain [snapshotname] [--current] {[--rename] [--clone]}
```

如果同时指定了 snapshotname 和 --current，它会强制编辑的快照成为当前快照。如果省略 snapshotname，则必须提供 --current，以便编辑当前快照。

这等同于以下命令顺序，但它还包含一些错误检查：

```
# virsh snapshot-dumpxml dom name > snapshot.xml
# vi snapshot.xml [note - this can be any editor]
# virsh snapshot-create dom snapshot.xml --redefine [--current]
```

如果指定了 --rename，则快照被重命名。如果指定了 --clone，然后更改快照名称将生成快照元数据的克隆。如果未指定，则编辑不会更改快照名称。请注意，必须小心更改快照名称，因为单个 qcow2 文件中内部快照的内容只能从原始快照名称访问。

20.39.5. snapshot-info

snapshot-info 域 命令显示有关快照的信息。要使用，请运行：

```
# snapshot-info domain {snapshot | --current}
```

输出有关指定 快照 的基本信息，或使用 --current 的当前快照。

20.39.6. snapshot-list

列出给定客户机虚拟机的所有可用快照，默认显示快照名称、创建时间和客户机虚拟机状态列。要使用，请运行：

```
# virsh snapshot-list domain [{--parent | --roots | --tree}] [{[--from] snapshot | --current} [--
descendants]] [--metadata] [--no-metadata] [--leaves] [--no-leaves] [--inactive] [--active] [--disk-only]
[--internal] [--external]
```

可选参数如下：

- --parent 在 output 表中添加一个列，其中包含每个快照的父名称。此选项不能与 --roots 或 --tree 一起使用。
- --roots 将列表过滤为仅显示没有父项的快照。此选项不能与 --parent 或 --tree 一起使用。

- --tree 以树形格式显示输出，仅列出快照名称。此选项不能用于 --root 或 -- parent。

- --from 将列表过滤为给定快照的子快照，或者如果提供了 --current，则会导致列表从当前快照开始。在隔离或者与 --parent 一起使用时，该列表仅限于直接子对象，除非也存在 --descendants。与 --tree 一起使用时，使用 --descendants 是简化的。这个选项与 --roots 不兼容。请注意，列表中不包含 --from 或 --current 的起点，除非也存在 --tree 选项。

- 指定 --leaves，列表将过滤为仅包含任何子项的快照。同样，如果指定了 --no-leaves，则列表将过滤为仅包含带有子项的快照。（请注意，忽略这两个选项都不会过滤，而提供这两个选项将产生相同的列表或错误，具体取决于服务器是否识别了标志。）过滤选项与 --tree 不兼容。

- 指定了 --metadata，列表将过滤为仅包含 libvirt 元数据的快照，因此可防止不保护持久客户机虚拟机，或者在销毁 临时 客户机虚拟机时丢失。同样，如果指定了 --no-metadata，则列表将过滤为仅包含存在的快照，而无需提供 libvirt 元数据。

- 指定 --inactive，列表将过滤为在客户端虚拟机关闭时拍摄的快照。如果指定了 --active，则列表将过滤为在客户机虚拟机运行时生成的快照，其中快照包含内存状态，以恢复到该运行状态。如果指定了 --disk-only，则列表将过滤为在客户机虚拟机运行时生成的快照，但快照只包括磁盘状态。

- 指定 --internal，列表将过滤为使用现有磁盘镜像的内部存储的快照。如果指定了 --external，列表将过滤为使用外部文件进行磁盘镜像或内存状态的快照。

20.39.7. snapshot-dumpxml

virsh snapshot-dumpxml 域 快照命令输出名为 snapshot 的虚拟客户机快照的快照 XML。要使用，请运行：

```
# virsh snapshot-dumpxml domain snapshot [--security-info]
```

security-info 选项将包括安全敏感信息。使用 virsh snapshot-current 来轻松访问当前快照的 XML。

20.39.8. snapshot-parent

输出父快照的名称（如果有、指定快照）或当前带有 --current 的快照的名称。要使用，请运行：

```
# virsh snapshot-parent domain {snapshot | --current}
```

20.39.9. snapshot-revert

将给定域恢复为快照指定的 快照，或使用 --current 恢复到当前快照。

警告

请注意，这是一个破坏性操作；自上次快照执行之后域中的任何更改都将丢失。另请注意，snapshot-revert 完成之后域的状态将在执行原始快照时是域的状态。

要恢复快照，请运行：

```
# virsh snapshot-revert domain {snapshot | --current} [{--running | --paused}] [--force]
```

通常，恢复到快照会使域处于创建快照时的状态，但没有 guest 虚拟机状态的磁盘快照会使域处于非活动状态。传递 --running 或 --paused 选项将执行额外的状态更改（如引导不活跃域，或暂停正在运行的域）。因为临时域无法处于不活跃状态，因此在恢复到临时域的磁盘快照时，需要使用这些标志之一。

快照恢复涉及额外的风险，需要使用 --force 进行。一个快照是快照中缺少恢复配置的完整域信息；因为 libvirt 无法证明当前配置与快照时使用的配置匹配，请提供 --force s，因为快照与当前配置兼容（如果不是，则域可能无法运行）。另一种情况是将正在运行的域恢复到一个活动状态，其中必须创建新的管理程序，而不是重复使用现有的虚拟机监控程序，因为它意味着破坏任何现有 VNC 或 Spice 连接等缺点；这种条件发生，使用明显不兼容的配置，以及结合 --start 或 --pause 标志的活跃快照。

20.39.10. snapshot-delete

virsh snapshot-delete 域 命令会删除指定域的快照。为此，请运行：

```
# virsh snapshot-delete domain {snapshot | --current} [--metadata] [{--children | --children-only}]
```

此命令会删除名为 snapshot 的域 的快照，或使用 --current 当前的快照。如果此快照有子快照，则此快照中的更改将合并到子快照。如果使用 --children 选项，则会删除此快照和此快照的任何子项。如果使用 --children-only，它将删除此快照的任何子项，但保留此快照不变。这两个标志是相互排斥的。

使用 --metadata 会删除由 libvirt 维护的快照元数据，同时让快照内容完全完好以供外部工具访问；否则，删除快照也会从该时间点删除其数据内容。

20.40. 虚拟机 CPU 型号配置

20.40.1. 简介

每个虚拟机监控程序都拥有自己的策略，用于了解默认情况下将针对其 CPU 看到的客户机虚拟机。些虚拟机监控程序决定将哪些 CPU 主机物理机器功能可用于客户机虚拟机，而 QEMU/KVM 则为客户机虚拟机提供一个通用模型，名为 qemu32 或 qemu64。这些虚拟机监控程序执行更高级的过滤，将所有物理 CPU 分类到少数组中，并为向客户机虚拟机显示的每个组有一个基准 CPU 模型。这种行为使客户机虚拟机在主机物理机器之间安全迁移，只要它们都有可分类到同一组的物理 CPU。libvirt 通常并不强制实施策略本身，而是提供高层定义他们自己的策略的机制。了解如何获取 CPU 模型信息并定义合适的客户机虚拟机 CPU 模型，可确保在主机物理机器之间成功迁移客户机虚拟机。请注意，管理程序只能模拟了解在虚拟机监控程序发行后创建的功能以及可能无法模拟的功能。

20.40.2. 了解主机物理机器 CPU 型号

virsh capabilities 命令显示 XML 文档，用于描述管理程序连接和主机物理计算机的功能。显示的 XML 模式已扩展，以提供主机物理机器 CPU 模型的信息。描述 CPU 模型的一个挑战是，每个架构都有不同的方法来公开其功能。QEMU/KVM 和 libvirt 使用一种组合了 CPU 型号名称字符串的方案，以及一组指定标志。

数据库不实际列出了所有已知的 CPU 型号，因此 libvirt 具有少量基准 CPU 模型名称列表。它选择共享最大 CPUID 位数与实际主机物理机器 CPU 的 CPUID 位，然后将剩余位列为已命名功能。请注意，libvirt 不会显示基准 CPU 包含哪些功能。这看起来像一个缺陷一样，但在本节中会说明这一点，实际上并不需要知道此信息。

20.40.3. 确定 VFIO IOMMU 设备支持

使用 virsh domcapabilities 命令确定对 VFIO 的支持。请参见以下示例输出：

图 20.3. 确定 VFIO 支持

```
# virsh domcapabilities

[...output truncated...]

<enum name='pciBackend'>
        <value>default</value>
        <value>vfio</value>

[...output truncated...]
```

20.40.4. 确定兼容的 CPU 型号来保证主机物理机器池

现在，可以找出单一主机物理计算机具有的 CPU 功能，下一步是决定哪些 CPU 功能最好向客户机虚拟机公开。如果已知客户机虚拟机绝不需要迁移到另一台主机物理计算机，则主机物理机器 CPU 模型可以直接通过未修改进行传递。虚拟化数据中心可能具有一组配置，它可以保证所有服务器都拥有 100% 的相同 CPU。同样，可以通过未修改来直接传递主机物理机器 CPU 模型。然而，较常见的情况是主机物理机器之间的 CPU 变化。在这种混合 CPU 环境中，必须确定最低的 denominator CPU。这并不完全直接，因此 libvirt 为此任务提供了一个 API。如果 libvirt 是 XML 文档列表，每个文档都描述主机物理计算机的 CPU 模型，libvirt 在内部将其转换为 CPUID 掩码，计算它们的交集，并将 CPUID 掩码结果转换为 XML CPU 描述。

以下是执行 virsh 功能时 libvirt 作为基本工作站功能 的一个示例：

图 20.4. 拉取主机物理机器的 CPU 模型信息

```
<capabilities>
  <host>
    <cpu>
      <arch>i686</arch>
      <model>pentium3</model>
      <topology sockets='1' cores='2' threads='1'/>
      <feature name='lahf_lm'/>
      <feature name='lm'/>
      <feature name='xtpr'/>
      <feature name='cx16'/>
      <feature name='ssse3'/>
      <feature name='tm2'/>
      <feature name='est'/>
      <feature name='vmx'/>
      <feature name='ds_cpl'/>
      <feature name='monitor'/>
      <feature name='pni'/>
      <feature name='pbe'/>
      <feature name='tm'/>
      <feature name='ht'/>
      <feature name='ss'/>
      <feature name='sse2'/>
      <feature name='acpi'/>
      <feature name='ds'/>
      <feature name='clflush'/>
      <feature name='apic'/>
    </cpu>
  </host>
</capabilities>
```

现在，与不同的服务器进行比较，使用相同的 virsh capabilities 命令：

图 20.5. 从随机服务器生成 CPU 描述

```
<capabilities>
  <host>
    <cpu>
      <arch>x86_64</arch>
      <model>phenom</model>
      <topology sockets='2' cores='4' threads='1'/>
      <feature name='osvw'/>
      <feature name='3dnowprefetch'/>
      <feature name='misalignsse'/>
      <feature name='sse4a'/>
      <feature name='abm'/>
      <feature name='cr8legacy'/>
      <feature name='extapic'/>
      <feature name='cmp_legacy'/>
      <feature name='lahf_lm'/>
      <feature name='rdtscp'/>
      <feature name='pdpe1gb'/>
      <feature name='popcnt'/>
      <feature name='cx16'/>
      <feature name='ht'/>
      <feature name='vme'/>
    </cpu>
    ...snip...
```

要查看此 CPU 描述是否与以前的工作站 CPU 描述兼容，请使用 virsh cpu-compare 命令。

减少的内容存储在名为 virsh-caps-workstation-cpu-only.xml 的文件中，可以在此文件上执行 virsh cpu-compare 命令：

```
# virsh cpu-compare virsh-caps-workstation-cpu-only.xml
Host physical machine CPU is a superset of CPU described in virsh-caps-workstation-cpu-only.xml
```

如本输出中所示，libvirt 正确报告 CPU 严格兼容。这是因为客户端 CPU 中缺少几个功能。为了能够在客户端和服务器之间迁移，需要打开 XML 文件并注释掉一些功能。要确定需要删除哪些功能，请在包含两台计算机的 CPU 信息运行 virsh cpu - baseline 命令。运行 # virsh cpu-baseline two-cpus.xml 将生成：

图 20.6. 复合 CPU 基准

```
<cpu match='exact'>
  <model>pentium3</model>
  <feature policy='require' name='lahf_lm'/>
  <feature policy='require' name='lm'/>
  <feature policy='require' name='cx16'/>
  <feature policy='require' name='monitor'/>
  <feature policy='require' name='pni'/>
  <feature policy='require' name='ht'/>
  <feature policy='require' name='sse2'/>
  <feature policy='require' name='clflush'/>
  <feature policy='require' name='apic'/>
</cpu>
```

此复合文件显示哪些元素是通用的。不常见的一切都应被注释掉。

20.41. 配置客户机虚拟机 CPU 型号

对于简单的默认值，客户机虚拟机 CPU 配置接受与主机物理机功能 XML 公开相同的基本 XML 表示。换句话说， **virsh cpu-baseline** *命令的 XML 可以直接复制到 域 元素下顶级的客户机虚拟机 XML 中。 在前面的 XML 代码段中，在客户机虚拟机 XML 中描述 CPU 时有几个额外的属性。这大多可以忽略，但是对于好奇，这都是关于它们的作用的快速描述。顶级* **<cpu>** *元素具有与 可能 值匹配的属性：*

- **match='minimum'** *- 主机物理机器 CPU 必须至少有客户机虚拟机 XML 中描述的 CPU 功能。如果主机物理计算机的附加功能除客户机虚拟机配置之外，这些功能也会暴露于客户机虚拟机。*

- **match='exact'** *- 主机物理机器 CPU 必须至少有客户机虚拟机 XML 中描述的 CPU 功能。如果主机物理计算机的附加功能在客户机虚拟机配置之外，将从客户机虚拟机中屏蔽这些功能。*

- **match='strict'** *- 主机物理机器 CPU 必须与客户机虚拟机 XML 中描述的 CPU 功能完全相同。*

下一个改进是， **<feature>** *元素可以具有可能的值的额外 'policy' 属性：*

- **policy='force'** *- 即使主机物理机器没有，也会向客户机虚拟机公开该功能。这通常仅在软件模拟的情况下有用。*

注意

即使使用 强制 策略，管理程序可能无法模拟特定功能。

- policy='require' - 向客户机虚拟机公开功能，并在主机物理机没有它时失败。这是合理的默认值。
- policy='optional' - 如果支持该功能，则会向客户机虚拟机公开该功能。
- policy='disable' - 如果主机物理机器有这个功能，则从客户机虚拟机中隐藏它。
- policy='forbid' - 如果主机物理计算机具有此功能，则会失败并拒绝启动 guest 虚拟机。

"forbid"策略适用于特定情况，当错误运行的应用程序在 CPUID 掩码中没有时也尝试使用功能，并且您希望防止使用该功能的主机物理机器上意外运行客户机虚拟机。'optional' 策略在迁移方面具有特殊行为。最初启动 guest 虚拟机时，标志为可选，但当客户机虚拟机实时迁移时，此策略将转变为 'require'，因为您不能在迁移过程中消失。

20.42. 管理客户机虚拟机的资源

virsh 允许根据每个虚拟机对资源的分组和分配。这由 libvirt 守护进程管理，后者代表客户机虚拟机创建 cgroups 并管理它们。系统管理员唯一保留的内容是查询或针对指定的客户机虚拟机设置可调项。libvirt 服务使用以下 cgroups 来调整和监控虚拟机：

- 内存 - 内存控制器允许在 RAM 和交换使用量上设置限制，并查询组中所有进程的累积使用
- Cpuset - CPU 集控制器将一个组中的进程绑定到一组 CPU，并控制在 CPU 间的迁移。
- cpuacct - CPU accountingt 控制器提供有关一组进程的 CPU 用量的信息。
- cpu - CPU 调度程序控制器控制组中的进程的优先级。这和授予 nice 级别的权限类似。

- devices - 设备控制器授予对字符和块设备的访问控制列表。
- freezer - freezer 控制器暂停并恢复执行组中的进程。这和整个组的 SIGSTOP 类似。
- net_cls - 网络类控制器通过将进程与 tc 网络类关联来管理网络利用率。

cgroup 由 libvirt 中的 systemd 设置。以下 virsh tuning 命令影响了 cgroup 配置的方式：

- schedinfo - 中描述的 第 20.43 节 “设置计划参数”
- blkdeviotune - 中描述的 第 20.44 节 “磁盘 I/O 轮转”
- blkiotune - 中描述的 第 20.45 节 “显示或设置块 I/O 参数”
- domiftune - 中描述的 第 20.12.5 节 “设置网络接口带宽参数”
- memtune - 描述 第 20.46 节 “配置内存调整”

如需有关 cgroups 的更多信息，请参阅 Red Hat Enterprise Linux 7 资源管理指南。

20.43. 设置计划参数

virsh schedinfo 命令修改主机上虚拟机进程的主机调度参数。应使用以下命令格式：

```
# virsh schedinfo domain --set --current --config --live
```

每个参数解释如下：

- 域 - 客户机虚拟机域

- --set - 这里放置的字符串是要被调用的控制器或操作。字符串使用 parameter=value 格式。如果需要，还应添加其他参数或值。

- --current - 与 --set 一起使用时，会将指定的 设置 字符串用作当前调度程序信息。在使用 的情况下不使用 时，会显示当前的调度程序信息。

- --config - 与 --set 一起使用时，将在下次重启时使用指定的 设置 字符串。在不使用 的情况下使用 时，将显示保存在 配置文件中的调度程序信息。

- --live - 与 --set 一起使用时，将使用当前运行的客户机虚拟机上指定的 设置 字符串。如果使用时，不使用 时，将显示运行中虚拟机当前使用的配置设置

调度程序可以使用以下参数来设置： cpu_shares、 vcpu_period 和 vcpu_quota。这些参数应用到 vCPU 线程。

下面显示了参数如何映射到 cgroup 字段名称：

- cpu_shares:cpu.shares

- vcpu_period:cpu.cfs_period_us

- vcpu_quota:cpu.cfs_quota_us

例 20.98. schedinfo 显示

此示例显示 shell 客户机虚拟机的调度信息

```
# virsh schedinfo shell
Scheduler      : posix
cpu_shares     : 1024
vcpu_period    : 100000
vcpu_quota     : -1
```

例 20.99. schedinfo 集合

在本例中，cpu_shares 被改为 2046。这将影响当前状态，而不是配置文件。

```
# virsh schedinfo --set cpu_shares=2046 shell
Scheduler      : posix
cpu_shares     : 2046
vcpu_period    : 100000
vcpu_quota     : -1
```

libvirt 还支持 emulator_period 和 emulator_quota 参数，这些参数修改仿真程序进程的设置。

20.44. 磁盘 I/O 轮转

virsh blkdeviotune 命令可为指定的 guest 虚拟机设置磁盘 I/O 节流。这可以防止客户机虚拟机过度利用共享资源，从而影响其他客户机虚拟机的性能。应使用以下格式：

```
#virsh blkdeviotune domain <device> [[--config] [--live] | [--current]] [[total-bytes-sec] | [read-bytes-sec] [write-bytes-sec]] [[total-iops-sec] [read-iops-sec] [write-iops-sec]]
```

唯一必需的参数是客户机虚拟机的域名。要列出域名，请运行 virsh domblklist 命令。--config、--live 和 --current 参数的工作方式与 第 20.43 节 “设置计划参数” 相同。如果没有指定限制，它将查询当前的 I/O 限制设置。否则，使用以下标记更改限制：

- --total-bytes-sec - 指定每秒字节数为单位的总吞吐量限制。
- --read-bytes-sec - 指定读取吞吐量（以字节/秒为单位）。
- --write-bytes-sec - 指定写入吞吐量（以字节/秒为单位）。
- --total-iops-sec - 指定每秒总 I/O 操作限值。
- --read-iops-sec - 指定每秒读取 I/O 操作限值。
- --write-iops-sec - 指定每秒写入 I/O 操作限值。

如需更多信息，请参阅 virsh man page 的 blkdeviotune 部分。有关域 XML 示例，请参阅 图 23.27 “devices - 硬盘、软盘、CD-ROM 示例”。

20.45. 显示或设置块 I/O 参数

blkiotune 命令集或显示指定 guest 虚拟机的 I/O 参数。应使用以下格式：

```
# virsh blkiotune domain [--weight weight] [--device-weights device-weights] [---device-read-iops-sec -
device-read-iops-sec] [--device-write-iops-sec device-write-iops-sec] [--device-read-bytes-sec device-
read-bytes-sec] [--device-write-bytes-sec device-write-bytes-sec] [[--config] [--live] | [--current]]
```

有关这个命令的更多信息，请参阅 虚拟化调整和优化指南

20.46. 配置内存调整

Virtualization 调整和优化 指南中涵盖了 virsh memtune virtual_machine --parameter size 命令。

第 21 章 使用离线工具进行客户机虚拟机磁盘访问

21.1. 简介

Red Hat Enterprise Linux 7 提供多个 libguestfs 工具，可访问、编辑和创建客户机虚拟机磁盘或其他磁盘镜像。这些工具有多种用途，包括：

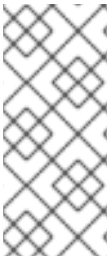
- 查看或下载位于客户机虚拟机磁盘中的文件。
- 在客户机虚拟机磁盘上编辑或上传文件。
- 读取或写入客户机虚拟机配置.
- 准备包含文件、目录、文件系统、分区、逻辑卷和其他选项的新磁盘镜像。
- 修复无法引导的客户机虚拟机，或者需要启动配置更改的客户机虚拟机。
- 监控客户机虚拟机的磁盘使用情况.
- 审计客户虚拟机合规性，例如组织安全标准。
- 通过克隆和修改模板来部署客户机虚拟机.
- 读取 CD 和 DVD ISO 映像以及软盘磁盘映像.

警告

您不得使用本章中列出的实用程序写入附加到虚拟机的客户机虚拟机或磁盘镜像，即使是在写入模式下打开这样的磁盘镜像。

这样做会导致客户机虚拟机的磁盘损坏。这些工具会尽量避免您这样做，但不能保证所有情况的安全。如果客户机虚拟机可能正在运行，则红帽强烈建议您不要使用该工具。

为提高安全性，某些实用程序可以用于只读模式（使用 --ro 选项），不会保存更改。



注意

libguestfs 及相关实用程序的文档的主要来源是 Linux man page。该 API 记录在 guestfs(3) 中，guestfish 记录在 guestfish(1) 中，虚拟化实用程序记录在自己的手册页（如 virt-df(1)）中。有关故障排除信息，请参阅 第 A.17 节 “libguestfs 故障排除”

21.1.1. 使用远程连接时要小心

Red Hat Enterprise Linux 7 中的一些虚拟化命令允许您指定一个远程 libvirt 连接。例如：

```
# virt-df -c  qemu://remote/system -d Guest
```

但是，Red Hat Enterprise Linux 7 中的 libguestfs 工具无法访问远程 libvirt 客户机的磁盘，使用远程 URL 的命令也无法按预期工作。

不过，从 Red Hat Enterprise Linux 7 开始，libguestfs 可以通过网络块设备(NBD)访问远程磁盘源。您可以使用 qemu-nbd 命令从远程机器导出磁盘镜像，并使用 nbd:// URL 访问它。您可能需要在防火墙中打开端口（端口 10809），如下所示：

在远程系统中： qemu-nbd -t disk.img

在本地系统中： virt-df -a nbd://remote

以下 libguestfs 命令会受到影响：

- *guestfish*
- *guestmount*
- *virt-alignment-scan*
- *virt-cat*
- *virt-copy-in*
- *virt-copy-out*
- *virt-df*
- *virt-edit*
- *virt-filesystems*
- *virt-inspector*
- *virt-ls*
- *virt-rescue*

- virt-sysprep
- virt-tar-in
- virt-tar-out
- virt-win-reg

21.2. 术语

本节介绍本章中全程所使用的术语。

- libguestfs（GUEST 文件系统 LIBrary）- 底层 C 库，提供打开磁盘镜像、读写文件的基本功能，等等。您可以将 C 程序直接写入这个 API。
- guestfish（GUEST 文件系统 Interactive SHell）是一个交互式的 shell，您可以通过命令行或从 shell 脚本使用。它公开 libguestfs API 的所有功能。
- 各种 virt 工具在 libguestfs 之上构建，它们提供了一种从命令行执行特定任务的方法。这些工具包括 virt-df、virt-rescue、virt-resize 和 virt-edit。
- augeas 是一个用于编辑 Linux 配置文件的库。虽然这与 libguestfs 独立，但 libguestfs 的大部分值都来自此工具的组合。
- guestmount 是 libguestfs 和 FUSE 之间的接口。它主要用于从主机物理机器上的磁盘镜像挂载文件系统。这个功能不是必需功能，但会很有用。

21.3. 安装

要安装 libguestfs、guestfish、libguestfs 工具和 guestmount，请输入以下命令：

```
# yum install libguestfs libguestfs-tools
```

要安装每个 libguestfs 相关软件包，包括语言绑定，请输入以下命令：

```
# yum install '*guestf*'
```

21.4. GUESTFISH SHELL

guestfish 是一个交互式 shell，您可以通过命令行或从 shell 脚本使用以访问 guest 虚拟机文件系统。libguestfs API 的所有功能均可从 shell 访问。

要开始查看或编辑虚拟机磁盘镜像，请输入以下命令，将路径替换为您预期的磁盘镜像：

```
$ guestfish --ro -a /path/to/disk/image
```

--ro 表示磁盘镜像以只读方式打开。这个模式总是安全，但不允许写入访问。仅在 确定 客户机虚拟机没有运行时省略这个选项，或者磁盘镜像没有附加到实时客户机虚拟机。无法使用 libguestfs 编辑实时客户机虚拟机，并尝试使磁盘不可逆损坏。

/path/to/disk/image 是到磁盘的路径。这可以是文件、主机物理机器逻辑卷（如 /dev/VG/LV）或 SAN LUN(/dev/sdf3)。

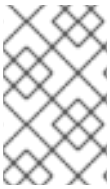
注意

libguestfs 和 guestfish 不需要 root 权限。当被访问的磁盘镜像需要 root 用户进行读取或写入时，您只需要以 root 用户身份运行它们。

当您以交互方式启动 guestfish 时，它将显示此提示：

```
$ guestfish --ro -a /path/to/disk/image

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

 Type: 'help' for help on commands
       'man' to read the manual
       'quit' to quit the shell

><fs>
```

在提示符处，键入 run 来启动库并附加磁盘镜像。第一次完成最多可能需要 30 秒。随后的开始将更快地完成。



注意

libguestfs 将使用硬件虚拟化加速（如 KVM（如果可用）来加快这个过程。

输入了 run 命令后，可以使用其他命令，作为以下部分所示。

21.4.1. 使用 guestfish 查看文件系统

这部分提供有关使用 guestfish 查看文件系统的信息。

21.4.1.1. 手动列出和查看

list-filesystems 命令将列出 libguestfs 找到的文件系统。这个输出显示了 Red Hat Enterprise Linux 4 磁盘镜像：

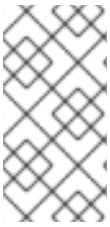
```
><fs> run
><fs> list-filesystems
/dev/vda1: ext3
/dev/VolGroup00/LogVol00: ext3
/dev/VolGroup00/LogVol01: swap
```

其他有用的命令包括 list-devices、list-partitions、lvs、pvs、vfs-type 和 file。您可以通过键入 help 命令来获取 更多信息和有关任何命令的帮助，如下例所示：

```
><fs> help vfs-type
 NAME
   vfs-type - get the Linux VFS type corresponding to a mounted device

 SYNOPSIS
    vfs-type mountable

 DESCRIPTION
   This command gets the filesystem type corresponding to the filesystem on
   "device".

   For most filesystems, the result is the name of the Linux VFS module
   which would be used to mount this filesystem if you mounted it without
   specifying the filesystem type. For example a string such as "ext3" or
   "ntfs".
```

要查看文件系统的实际内容，必须先挂载它。

您可以使用 ls、ll ll、cat、更多、下载和 tarout 等 guestfish 命令来查看和下载文件和目录。



注意

此 shell 中没有当前工作目录的概念。与一般的 shell 不同，您无法使用 cd 命令来更改目录。所有路径都必须在顶部以正斜杠（）字符开始完全合格。使用 Tab 键完成路径。

要从 guestfish shell 退出，请键入 exit 或按 Ctrl+d。

21.4.1.2. 通过 guestfish 检查

guestfish 本身可以检查映像并挂载文件系统，而不是手动列出和挂载文件系统。要做到这一点，在命令行中添加 -i 选项：

```
$ guestfish --ro -a /path/to/disk/image -i

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

 Type: 'help' for help on commands
       'man' to read the manual
       'quit' to quit the shell

 Operating system: Red Hat Enterprise Linux AS release 4 (Nahant Update 8)
 /dev/VolGroup00/LogVol00 mounted on /
 /dev/vda1 mounted on /boot

 ><fs> ll /
 total 210
 drwxr-xr-x. 24 root root  4096 Oct 28 09:09 .
 drwxr-xr-x  21 root root  4096 Nov 17 15:10 ..
 drwxr-xr-x.  2 root root  4096 Oct 27 22:37 bin
 drwxr-xr-x.  4 root root  1024 Oct 27 21:52 boot
 drwxr-xr-x.  4 root root  4096 Oct 27 21:21 dev
 drwxr-xr-x. 86 root root 12288 Oct 28 09:09 etc
 ...
```

由于 guestfish 需要启动 libguestfs 后端才能执行检查和挂载，因此在使用 -i 选项时不需要 run 命令。i 选项 适用于许多常见的 Linux 客户机虚拟机。

21.4.1.3. 按名称访问客户机虚拟机

当您指定给 libvirt 的名称时（换句话说，如 virsh list --all所示），可从命令行访问客户机虚拟机。使用 -d 选项按名称或不使用 -i 选项访问客户机虚拟机：

```
$ guestfish --ro -d GuestName -i
```

21.4.2. 使用 guestfish 添加文件

要添加含有 guestfish 的文件，您需要拥有完整的 URI。该文件可以是本地文件，也可以是位于网络块设备(NBD)或远程块设备(RBD)的文件。

这些示例中任何 URI 的格式应类似。对于本地文件，请使用 ///:

- guestfish -a disk.img
- guestfish -a file:///directory/disk.img
- guestfish -a nbd://example.com[:port]
- guestfish -a nbd://example.com[:端口]/exportname
- guestfish -a nbd://?socket=/socket
- guestfish -a nbd:///exportname?socket=/socket
- guestfish -a rbd:///pool/disk
- guestfish -a rbd://example.com[:端口]/pool/disk

21.4.3. 使用 guestfish 修改文件

要修改文件，请创建目录或对客户机虚拟机进行其他更改，首先在本节的开头指出警告：您的 guest 虚拟机必须关机。使用 guestfish 编辑或更改正在运行的磁盘将导致磁盘损坏。这部分提供了编辑 /boot/grub/grub.conf 文件的示例。当确定客户端虚拟机已关闭时，可以省略 --ro 标志，以便使用命令获得写访问，例如：

```
$ guestfish -d RHEL3 -i

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

 Type: 'help' for help on commands
       'man' to read the manual
       'quit' to quit the shell

 Operating system: Red Hat Enterprise Linux AS release 3 (Taroon Update 9)
 /dev/vda2 mounted on /
 /dev/vda1 mounted on /boot

><fs> edit /boot/grub/grub.conf
```

用于编辑文件的命令包括 编辑、vi 和 emacs。还有许多命令用于创建文件和目录，如 写入、mkdir、上传和 tar 中。

21.4.4. 使用 guestfish 的其他操作

您还可以格式化文件系统、创建分区、创建和调整 LVM 逻辑卷以及更多大小，包括 mkfs、part-add、lvresize、lvresize、lvcreate 和 pvcreate 等命令。

21.4.5. 使用 guestfish 进行 shell 脚本

熟悉了使用 guestfish 时，根据您的需要以互动方式使用 guestfish，编写带有它的 shell 脚本会很有用。以下是向客户机添加新的 MOTD（日期消息）的简单 shell 脚本：

```
#!/bin/bash -
 set -e
 guestname="$1"

 guestfish -d "$guestname" -i <<'EOF'
   write /etc/motd "Welcome to Acme Incorporated."
   chmod 0644 /etc/motd
 EOF
```

21.4.6. augeas 和 libguestfs 脚本

在编写脚本以操作 Linux 客户机虚拟机配置时，可将 libguestfs 与 Augeas 相结合。例如，以下脚本使用 Augeas 解析客户机虚拟机的键盘配置，并打印布局。请注意，这个示例仅适用于运行 Red Hat

Enterprise Linux 的客户机虚拟机：

```
#!/bin/bash -
 set -e
 guestname="$1"

 guestfish -d "$1" -i --ro <<'EOF'
   aug-init / 0
   aug-get /files/etc/sysconfig/keyboard/LAYOUT
 EOF
```

augeas 也可用于修改配置文件。您可以修改以上脚本以更改键盘布局：

```
#!/bin/bash -
 set -e
 guestname="$1"

 guestfish -d "$1" -i <<'EOF'
   aug-init / 0
   aug-set /files/etc/sysconfig/keyboard/LAYOUT '"gb"'
   aug-save
 EOF
```

请注意两个脚本之间的三个更改：

1. *第二个示例中删除了 --ro 选项，它能够写入客户机虚拟机。*
2. *aug-get 命令已更改为 aug-set 以修改值，而不是获取它。新值将是 "gb"（包括引号）。*
3. *此处使用 aug-save 命令，因此 Augeas 会将更改写出到磁盘。*

注意

有关 8 月的更多信息，请访问网站 http://augeas.net。

guestfish 可以比我们在此简介文档中的更多操作介绍更多。例如，从头开始创建磁盘镜像：

```
guestfish -N fs
```

或者从磁盘镜像复制整个目录：

```
><fs> copy-out /home /tmp/home
```

有关详细信息，请参见 man page guestfish(1)。

21.5. 其他命令

这部分论述了使用 guestfish 查看和编辑客户机虚拟机磁盘镜像更简单的工具。

- virt-cat 与 guestfish 下载命令类似。它下载并向客户机虚拟机显示单个文件。例如：

```
# virt-cat RHEL3 /etc/ntp.conf | grep ^server
server      127.127.1.0       # local clock
```

- virt- edit 与 guestfish 编辑命令类似。它可用于交互地编辑客户机虚拟机内的单个文件。例如，您可能需要在基于 Linux 的客户机虚拟机中编辑 grub.conf 文件，不会引导：

```
# virt-edit LinuxGuest /boot/grub/grub.conf
```

virt-edit 有一个模式，可用于对单个文件进行简单的非交互式更改。对于这个方法，使用 -e 选项。例如，以下命令将 Linux 客户机虚拟机中的 root 密码更改为没有密码：

```
# virt-edit LinuxGuest /etc/passwd -e 's/^root:.*?:/root::/'
```

- virt-ls 与 guestfish ls ll 和 find 命令类似。它用于列出目录或目录（递归）。例如，以下命令将递归列出 Linux 客户机虚拟机中的 /home 下的文件和目录：

```
# virt-ls -R LinuxGuest /home/ | less
```

21.6. VIRT-RESCUE:RESCUE SHELL

这部分提供有关救援 shell 的信息。

21.6.1. 简介

这部分论述了 virt-rescue，它类似于虚拟机的救援 CD。它将 guest 虚拟机引导进入救援 shell，以便能够执行维护以更正错误并且 guest 虚拟机可以修复。

virt-rescue 和 guestfish 之间有一些重叠。务必要区分不同的用途。virt-rescue 使用普通 Linux 文件系统工具进行交互式、临时更改。它尤其适用于修复已出现故障的 guest 虚拟机。virt-rescue 无法脚本化。

相比之下，guestfish 非常有用，可通过一组正式的命令(libguestfs API)进行结构化更改，尽管其也可以以交互方式使用。

21.6.2. 运行 virt-rescue

在 guest 虚拟机中使用 virt-rescue 之前，请确保 guest 虚拟机未在运行，否则将发生磁盘损坏。当您确定客户机虚拟机未上线时，请输入：

```
$ virt-rescue -d GuestName
```

（其中 GuestName 是 libvirt 已知的 guest 名称），或者：

```
$ virt-rescue -a /path/to/disk/image
```

（其中，路径可以是任意文件，任意逻辑卷、LUN 等）包含客户机虚拟机磁盘。

您会看到过去滚动的输出，因为 virt-rescue 引导 rescue 虚拟机。最后会显示：

```
Welcome to virt-rescue, the libguestfs rescue shell.

Note: The contents of / are the rescue appliance.
You have to mount the guest virtual machine's partitions under /sysroot
before you can examine them.

bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
><rescue>
```

此处的 shell 提示符是普通的 bash shell，提供了一定程度的普通 Red Hat Enterprise Linux 命令集。例如，您可以输入：

```
><rescue> fdisk -l /dev/vda
```

以上命令将列出磁盘分区。要挂载文件系统，建议您将其挂载到 /sysroot 下，它是救援计算机中的空目录，以便用户挂载您想要的任何内容。请注意，/ 下的文件 是救援虚拟机本身中的文件：

```
><rescue> mount /dev/vda1 /sysroot/
EXT4-fs (vda1): mounted filesystem with ordered data mode. Opts: (null)
><rescue> ls -l /sysroot/grub/
 total 324
 -rw-r--r--. 1 root root     63 Sep 16 18:14 device.map
 -rw-r--r--. 1 root root  13200 Sep 16 18:14 e2fs_stage1_5
 -rw-r--r--. 1 root root  12512 Sep 16 18:14 fat_stage1_5
 -rw-r--r--. 1 root root  11744 Sep 16 18:14 ffs_stage1_5
 -rw-------. 1 root root   1503 Oct 15 11:19 grub.conf
 [...]
```

完成对客户机虚拟机进行修复后，通过输入 exit 或 Ctrl+d 退出 shell。

virt-rescue 具有很多命令行选项。最常用的选项有：

- --ro:在 guest 虚拟机上以只读模式运行。不会保存任何更改。您可以使用它来试验 guest 虚拟机。当您从 shell 退出后，所有更改将被丢弃。
- --network:启用来自 rescue shell 的网络访问。例如，如果您需要将 RPM 或其他文件下载到客户机虚拟机中，可使用此示例。

21.7. VIRT-DF:监控磁盘使用情况

本节介绍监控磁盘使用情况的信息。

21.7.1. 简介

这部分论述了 virt-df，它显示来自磁盘镜像或客户机虚拟机的文件系统使用。它与 Linux df 命令相似，但针对虚拟机。

21.7.2. 运行 virt-df

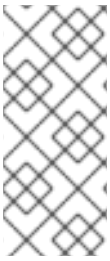
要显示在磁盘镜像中找到的所有文件系统的文件系统使用，请输入以下命令：

```
# virt-df -a /dev/vg_guests/RHEL7
 Filesystem                1K-blocks       Used  Available  Use%
 RHEL6:/dev/sda1              101086      10233      85634   11%
 RHEL6:/dev/VolGroup00/LogVol00 7127864    2272744    4493036   32%
```

（其中 /dev/vg_guests/RHEL7 是 Red Hat Enterprise Linux 7 虚拟机磁盘镜像。在这种情况下，路径是此磁盘镜像所在的主机物理机器逻辑卷。）

您还可以自行使用 virt-df 来列出关于 libvirt 已知的所有客户机虚拟机的信息。virt-df 命令识别了一些与标准 df 相同的选项，如 -h （可读）和 -i （显示索引节点而不是块）。

```
# virt-df -h -d domname
 Filesystem                    Size       Used  Available  Use%
 F14x64:/dev/sda1              484.2M      66.3M     392.9M   14%
 F14x64:/dev/vg_f14x64/lv_root    7.4G       3.0G       4.4G   41%
 RHEL6brewx64:/dev/sda1        484.2M      52.6M     406.6M   11%
 RHEL6brewx64:/dev/vg_rhel6brewx64/lv_root
                              13.3G       3.4G       9.2G   26%
```



注意

您可以安全地在 live guest 虚拟机上使用 virt-df，因为它只需要只读访问。但是，您不应预期数字与在 guest 虚拟机内运行的 df 命令中的值完全一致。这是因为，磁盘上与实时客户机虚拟机的状态稍有同步的内容。然而，它应该足够适合用于分析和监控目的。

virt-df 旨在让您将统计信息集成到监控工具、数据库等中。这样，系统管理员可以生成有关磁盘使用情况趋势的报告；如果客户机虚拟机即将耗尽磁盘空间，则发出警报。为此，您应使用 --csv 选项生成机器可读的 Comma-Separated-Values(CSV)输出。CSV 输出可由大多数数据库、电子表格软件和各种其他工具和编程语言读取。原始 CSV 类似如下：

```
# virt-df --csv -d RHEL6Guest
 Virtual Machine,Filesystem,1K-blocks,Used,Available,Use%
 RHEL6brewx64,/dev/sda1,102396,24712,77684,24.1%
 RHEL6brewx64,/dev/sda2,20866940,7786652,13080288,37.3%
```

21.8. VIRT-RESIZE：重新定义虚拟机大小

这部分提供有关重新定义离线虚拟客户机大小的信息。

21.8.1. 简介

这部分论述了 virt-resize、扩展或缩小客户机虚拟机的工具。它仅适用于离线的 guest 虚拟机（下线）。它的工作原理是复制客户机虚拟机镜像并让原始磁盘映像保持不变。这是理想是因为您可以使用原始镜像作为备份，但需要两倍的磁盘空间。

21.8.2. 扩展磁盘镜像

本节演示了扩展磁盘镜像的简单情况：

1. 找到要重新定义的磁盘镜像大小。您可以将 virsh dumpxml GuestName 用于 libvirt 客户机虚拟机。

2. 决定您要如何扩展 guest 虚拟机。在客户机虚拟机磁盘中运行 virt-df -h 和 virt-filesystems，如下所示：

```
# virt-df -h -a /dev/vg_guests/RHEL6
Filesystem                  Size       Used  Available  Use%
RHEL6:/dev/sda1               98.7M      10.0M      83.6M   11%
RHEL6:/dev/VolGroup00/LogVol00  6.8G       2.2G       4.3G   32%

# virt-filesystems -a disk.img --all --long -h
/dev/sda1 ext3 101.9M
/dev/sda2 pv 7.9G
```

以下示例演示了如何进行：

- 将第一个（引导）分区的大小从大约 100MB 增加到 500MB。
- 将总磁盘大小从 8GB 增加到 16GB。
- 扩展第二个分区以填满剩余空间。
- 展开 /dev/VolGroup00/LogVol00，以在第二个分区中填写新空间。

1. 确保已关闭 guest 虚拟机。

2. 将原始磁盘重命名为备份。您这样做的方式取决于原始磁盘的主机物理机器存储环境。如果

存储为文件，请使用 mv 命令。对于逻辑卷（在这个示例中所示），请使用 lvrename:

```
# lvrename /dev/vg_guests/RHEL6 /dev/vg_guests/RHEL6.backup
```

3. 创建新磁盘。本例中的要求可扩展到 16GB 的总磁盘大小。因为这里使用逻辑卷，所以使用以下命令：

```
# lvcreate -L 16G -n RHEL6 /dev/vg_guests
Logical volume "RHEL6" created
```

4. 此命令表达了第 2 步的要求：

```
# virt-resize \
    /dev/vg_guests/RHEL6.backup /dev/vg_guests/RHEL6 \
    --resize /dev/sda1=500M \
    --expand /dev/sda2 \
    --LV-expand /dev/VolGroup00/LogVol00
```

前两个参数是输入磁盘和输出磁盘。-- resize /dev/sda1=500M 重新定义第一个分区最多 500MB。- expand /dev/sda2 扩展第二个分区以填满所有剩余空间。-LV-expand /dev/VolGroup00/LogVol00 扩大客户端虚拟机逻辑卷到第二个分区中。

virt-resize 描述了它在输出中执行的操作：

```
Summary of changes:
  /dev/sda1: partition will be resized from 101.9M to 500.0M
  /dev/sda1: content will be expanded using the 'resize2fs' method
  /dev/sda2: partition will be resized from 7.9G to 15.5G
  /dev/sda2: content will be expanded using the 'pvresize' method
  /dev/VolGroup00/LogVol00: LV will be expanded to maximum size
  /dev/VolGroup00/LogVol00: content will be expanded using the 'resize2fs' method
  Copying /dev/sda1 ...
  [############################################################]
  Copying /dev/sda2 ...
  [############################################################]
  Expanding /dev/sda1 using the 'resize2fs' method
  Expanding /dev/sda2 using the 'pvresize' method
  Expanding /dev/VolGroup00/LogVol00 using the 'resize2fs' method
```

5. 尝试启动虚拟机。如果它正常工作（在测试后对其进行彻底测试），您可以删除备份磁盘。如果失败，请关闭虚拟机，删除新磁盘，然后将备份磁盘重新重命名为其原始名称。

6. 使用 virt-df 或 virt-filesystems 显示新大小：

```
# virt-df -h -a /dev/vg_pin/RHEL6
  Filesystem                  Size       Used  Available  Use%
  RHEL6:/dev/sda1             484.4M      10.8M     448.6M    3%
  RHEL6:/dev/VolGroup00/LogVol00 14.3G       2.2G      11.4G   16%
```

请注意，在某些情况下调整客户机虚拟机的大小可能会出现问题。如果 virt-resize 失败，则可以通过一些提示来查看并尝试在 virt-resize(1)man page 中。对于某些较旧的 Red Hat Enterprise Linux 客户机虚拟机，您可能需要特别注意有关 GRUB 的提示。

21.9. VIRT-INSPECTOR:检查客户机虚拟机

本节介绍检查客户机虚拟机。

21.9.1. 简介

virt-inspector 是一个检查磁盘镜像的工具，用来查找它所包含的操作系统。

21.9.2. 安装

要安装 virt-inspector 和文档，请输入以下命令：

```
# yum install libguestfs-tools
```

文档（包括 XML 输出和 Relax-NG 模式）将安装在 /usr/share/doc/libguestfs-devel-*/ 中，其中 * 被 libguestfs 版本号取代。

21.9.3. 运行 virt-inspector

您可以针对任何磁盘镜像或 libvirt 客户机虚拟机运行 virt-inspector，如下例所示：

```
$ virt-inspector -a disk.img > report.xml
```

或者如下所示：

```
$ virt-inspector -d GuestName > report.xml
```

结果将是一个 XML 报告(report.xml)。XML 文件的主要组件是一个顶层 **<operatingsytems>** 元素，它通常包含一个 **<operatingsystem>** 元素，如下所示：

```
<operatingsystems>
  <operatingsystem>

    <!-- the type of operating system and Linux distribution -->
    <name>linux</name>
    <distro>rhel</distro>
    <!-- the name, version and architecture -->
    <product_name>Red Hat Enterprise Linux Server release 6.4 </product_name>
    <major_version>6</major_version>
    <minor_version>4</minor_version>
    <package_format>rpm</package_format>
    <package_management>yum</package_management>
    <root>/dev/VolGroup/lv_root</root>
    <!-- how the filesystems would be mounted when live -->
    <mountpoints>
      <mountpoint dev="/dev/VolGroup/lv_root">/</mountpoint>
      <mountpoint dev="/dev/sda1">/boot</mountpoint>
      <mountpoint dev="/dev/VolGroup/lv_swap">swap</mountpoint>
    </mountpoints>

   < !-- filesystems-->
     <filesystem dev="/dev/VolGroup/lv_root">
       <label></label>
       <uuid>b24d9161-5613-4ab8-8649-f27a8a8068d3</uuid>
       <type>ext4</type>
       <content>linux-root</content>
       <spec>/dev/mapper/VolGroup-lv_root</spec>
     </filesystem>
     <filesystem dev="/dev/VolGroup/lv_swap">
       <type>swap</type>
       <spec>/dev/mapper/VolGroup-lv_swap</spec>
     </filesystem>
    <!-- packages installed -->
    <applications>
      <application>
        <name>firefox</name>
        <version>3.5.5</version>
        <release>1.fc12</release>
      </application>
    </applications>

  </operatingsystem>
</operatingsystems>
```

最能处理这些报告是使用 W3C 标准 XPath 查询来完成的。Red Hat Enterprise Linux 7 附带 xpath 命令行程序，可用于简单的实例。但是，为了长期和高级使用，您应该考虑使用 XPath 库和您首选的编

程语言。

例如，您可以使用以下 XPath 查询列出所有文件系统设备：

```
$ virt-inspector GuestName | xpath //filesystem/@dev
 Found 3 nodes:
 -- NODE --
 dev="/dev/sda1"
 -- NODE --
 dev="/dev/vg_f12x64/lv_root"
 -- NODE --
 dev="/dev/vg_f12x64/lv_swap"
```

或者使用以下命令列出所有安装应用程序的名称：

```
$ virt-inspector GuestName | xpath //application/name
 [...long list...]
```

21.10. 使用从编程语言中使用 API

libguestfs API 可直接使用 Red Hat Enterprise Linux 7 中的以下语言：c、C++、Perl、Python、Java、Ruby 和 OCaml。

- *要安装 C 和 C++ 绑定，请输入以下命令：*

  ```
  # yum install libguestfs-devel
  ```

- *安装 Perl 绑定：*

  ```
  # yum install 'perl(Sys::Guestfs)'
  ```

- *安装 Python 绑定：*

  ```
  # yum install python-libguestfs
  ```

- *安装 Java 绑定：*

```
# yum install libguestfs-java libguestfs-java-devel libguestfs-javadoc
```

- 安装 Ruby 绑定：

```
# yum install ruby-libguestfs
```

- 安装 OCaml 绑定：

```
# yum install ocaml-libguestfs ocaml-libguestfs-devel
```

每个语言的绑定本质上是相同的，但具有小的语法更改。一个 C 语句：

```
guestfs_launch (g);
```

在 Perl 中会出现以下内容：

```
$g->launch ()
```

或者类似 OCaml 中的以下内容：

```
g#launch ()
```

本节仅详述了 C 中的 API。

在 C 和 C++ 绑定中，您必须手动检查错误。在其他绑定中，错误会被转换为异常；以下示例中显示的其他错误检查不是其他语言所必需的，但您可能希望添加代码来进行异常。有关 libguestfs API 架构的一些兴趣点，请参见以下列表：

- libguestfs API 是同步的。每个调用块直到完成为止。如果要异步调用，您必须创建一个线程。

- libguestfs API 不是线程安全：每个句柄只能从一个线程使用；或者，如果您希望在线程间共享句柄，您应该实施自己的 mutex，以确保两个线程无法同时在一个句柄上执行命令。

-

您不应该在同一磁盘镜像上打开多个句柄。如果所有句柄都是只读的，但仍然不推荐这样做。

- 如果其他任何可能使用该磁盘镜像（例如，实时虚拟机）时，您不应该添加用于写入的磁盘镜像。这样做会导致磁盘崩溃。

- 在当前使用（例如，实时虚拟机）的磁盘镜像上打开只读处理。但是，结果可能会不预计或不一致，特别是当您读取它时，磁盘镜像被大量写入。

21.10.1. 使用 C 程序与 API 交互

您的 C 程序应该首先包括 <guestfs.h> 标头文件并创建句柄：

```
#include <stdio.h>
#include <stdlib.h>
#include <guestfs.h>

int
main (int argc, char *argv[])
{
  guestfs_h *g;

  g = guestfs_create ();
  if (g == NULL) {
    perror ("failed to create libguestfs handle");
    exit (EXIT_FAILURE);
   }

   /* ... */

   guestfs_close (g);

   exit (EXIT_SUCCESS);
 }
```

将该程序保存到文件中(test.c)。编译该程序并使用以下两个命令运行它：

```
gcc -Wall test.c -o test -lguestfs
./test
```

在这个阶段，它不应该打印任何输出。本节的其余部分展示了如何扩展此程序来创建新的磁盘映像、对其进行分区、将其格式化为 ext4 文件系统，并在文件系统中创建一些文件。磁盘镜像名为 disk.img，并在当前目录中创建。

该程序的概要是：

- *创建句柄。*
- *将磁盘添加到句柄中。*
- *启动 **libguestfs** 后端。*
- *创建分区、文件系统和文件。*
- *关闭句柄并退出。*

以下是修改的程序：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <guestfs.h>

int
main (int argc, char *argv[])
{
  guestfs_h *g;
  size_t i;

  g = guestfs_create ();
  if (g == NULL) {
    perror ("failed to create libguestfs handle");
    exit (EXIT_FAILURE);
  }

  /* Create a raw-format sparse disk image, 512 MB in size. */
  int fd = open ("disk.img", O_CREAT|O_WRONLY|O_TRUNC|O_NOCTTY, 0666);
  if (fd == -1) {
    perror ("disk.img");
    exit (EXIT_FAILURE);
  }
  if (ftruncate (fd, 512 * 1024 * 1024) == -1) {
    perror ("disk.img: truncate");
    exit (EXIT_FAILURE);
```

```
}
if (close (fd) == -1) {
  perror ("disk.img: close");
  exit (EXIT_FAILURE);
}

/* Set the trace flag so that we can see each libguestfs call. */
guestfs_set_trace (g, 1);

/* Set the autosync flag so that the disk will be synchronized
 * automatically when the libguestfs handle is closed.
 */
guestfs_set_autosync (g, 1);

/* Add the disk image to libguestfs. */
if (guestfs_add_drive_opts (g, "disk.img",
      GUESTFS_ADD_DRIVE_OPTS_FORMAT, "raw", /* raw format */
      GUESTFS_ADD_DRIVE_OPTS_READONLY, 0,   /* for write */
      -1 /* this marks end of optional arguments */ )
    == -1)
  exit (EXIT_FAILURE);

/* Run the libguestfs back-end. */
if (guestfs_launch (g) == -1)
  exit (EXIT_FAILURE);

/* Get the list of devices. Because we only added one drive
 * above, we expect that this list should contain a single
 * element.
 */
char **devices = guestfs_list_devices (g);
if (devices == NULL)
  exit (EXIT_FAILURE);
if (devices[0] == NULL || devices[1] != NULL) {
  fprintf (stderr,
           "error: expected a single device from list-devices\n");
  exit (EXIT_FAILURE);
}

/* Partition the disk as one single MBR partition. */
if (guestfs_part_disk (g, devices[0], "mbr") == -1)
  exit (EXIT_FAILURE);

/* Get the list of partitions. We expect a single element, which
 * is the partition we have just created.
 */
char **partitions = guestfs_list_partitions (g);
if (partitions == NULL)
  exit (EXIT_FAILURE);
if (partitions[0] == NULL || partitions[1] != NULL) {
  fprintf (stderr,
           "error: expected a single partition from list-partitions\n");
  exit (EXIT_FAILURE);
}

/* Create an ext4 filesystem on the partition. */
```

```
  if (guestfs_mkfs (g, "ext4", partitions[0]) == -1)
    exit (EXIT_FAILURE);

  /* Now mount the filesystem so that we can add files. */
  if (guestfs_mount_options (g, "", partitions[0], "/") == -1)
    exit (EXIT_FAILURE);

  /* Create some files and directories. */
  if (guestfs_touch (g, "/empty") == -1)
    exit (EXIT_FAILURE);

  const char *message = "Hello, world\n";
  if (guestfs_write (g, "/hello", message, strlen (message)) == -1)
    exit (EXIT_FAILURE);

  if (guestfs_mkdir (g, "/foo") == -1)
    exit (EXIT_FAILURE);

  /* This uploads the local file /etc/resolv.conf into the disk image. */
  if (guestfs_upload (g, "/etc/resolv.conf", "/foo/resolv.conf") == -1)
    exit (EXIT_FAILURE);

  /* Because 'autosync' was set (above) we can just close the handle
   * and the disk contents will be synchronized. You can also do
   * this manually by calling guestfs_umount_all and guestfs_sync.
   */
  guestfs_close (g);

  /* Free up the lists. */
  for (i = 0; devices[i] != NULL; ++i)
    free (devices[i]);
  free (devices);
  for (i = 0; partitions[i] != NULL; ++i)
    free (partitions[i]);
  free (partitions);

  exit (EXIT_SUCCESS);
}
```

使用以下两个命令编译并运行该程序：

```
gcc -Wall test.c -o test -lguestfs
./test
```

如果程序成功完成，您应该保留一个名为 disk.img 的磁盘镜像，您可以使用 guestfish 检查它：

```
guestfish --ro -a disk.img -m /dev/sda1
><fs> ll /
><fs> cat /foo/resolv.conf
```

默认情况下，对于 C 和 C++ 绑定，libguestfs 将错误打印到 stderr。您可以通过设置错误处理程序来更改此行为。guestfs(3)手册页详细讨论。

21.11. VIRT-SYSPREP:重置虚拟机设置

virt-sysprep 命令行工具可用于重置或取消配置客户机虚拟机，以便从中克隆。这个过程涉及删除 SSH 主机密钥、删除持久网络 MAC 配置和删除用户帐户。virt-sysprep 还可以通过添加 SSH 密钥、用户或徽标来自定义虚拟机。根据需要，可以启用或禁用每个步骤。

要使用 virt-sysprep，客户端虚拟机必须离线，因此您必须在运行命令前关闭该虚拟机。请注意，virt-sysprep 修改客户机或磁盘镜像而无需复制它。如果要保留客户端虚拟机的现有内容，则必须首先复制或克隆磁盘。有关复制和克隆磁盘的更多信息，请参阅 libguestfs.org。

不建议以 root 身份使用 virt-sysprep，除非需要 root 才能访问磁盘镜像。然而，在这种情况下，最好将磁盘镜像的权限更改为可由运行 virt-sysprep 的非 root 用户写入。

要安装 virt-sysprep，请输入以下命令：

```
# yum install /usr/bin/virt-sysprep
```

以下命令选项可与 virt-sysprep 一起使用：

表 21.1. virt-sysprep 命令

命令	描述	示例
--help	显示有关特定命令或 **virt-sysprep** 命令的简短帮助条目。详情请查看 virt-sysprep man page。	**virt-sysprep --help**
-a [*file*] 或 --add [*file*]	添加 *指定的文件*，它应该是 guest 虚拟机的磁盘映像。磁盘镜像的格式被自动探测到。要覆盖此选项并强制使用特定格式，请使用 **--format** 选项。	**virt-sysprep --add /dev/vms/disk.img**
-a [*URI*] 或 --add [*URI*]	添加远程磁盘。URI 格式与 guestfish 兼容。如需更多信息，请参阅 第 21.4.2 节 “使用 guestfish 添加文件”。	**virt-sysprep -a rbd://example.com[:port]/pool/disk**

命令	描述	示例
-c [*URI*] 或 --connect [*URI*]	如果使用 libvirt，则连接到给定的 URI。如果省略，它会通过 KVM 管理程序进行连接。如果您直接指定客户机块设备(**virt-sysprep -a**)，那么根本不使用 libvirt。	**virt-sysprep -c qemu:///system**
-d [*guest*] 或 --domain [*guest*]	从指定的客户机虚拟机中添加所有磁盘。可以使用域 UUID 而不是域名。	**virt-sysprep --domain *90df2f3f-8857-5ba9-2714-7d95907b1c9e***
-n 或 --dry-run	对客户机虚拟机执行只读"dry run" sysprep 操作。这将运行 sysprep 操作，但会丢弃对磁盘末尾的任何更改。	**virt-sysprep -n**
--enable [*操作*]	启用指定的 *操作*。要列出可能的操作，请使用 --list 命令。	**virt-sysprep --enable ssh-hostkeys,udev-persistent-net**
--operation 或 --operations	选择执行了哪些 sysprep 操作。要禁用某个操作，请在操作名称前面使用 -。	**virt-sysprep --operations ssh-hotkeys,udev-persistent-net** 会启用这两个操作，而 **virt-sysprep --operations firewall-rules,-tmp-files** 会启用 firewall-rules 操作并禁用 tmp-files 操作。如需有效操作列表，请参阅 libguestfs.org。
--format [**raw**|**qcow2**|**auto**]	-a 选项的默认值是自动检测磁盘镜像的格式。使用此选项强制使用命令行中的 -a 选项的磁盘格式。使用 --format 自动检测自动探测到后续 -a 选项（请参阅上面的 -a 命令）。	**virt-sysprep --format raw -a *disk.img*** 会强制为 disk.img 强制原始格式(no auto-detection)，**virt-sysprep --format raw -a *disk.img* --format auto -a *another.img*** *会强制对* ***disk.img*** *强制（无自动探测）强制自动探测到* ***another.img*** *的自动探测。如果您有不受信任的原始格式客户机磁盘镜像，则应该使用这个选项指定磁盘格式。这可避免恶意客户端出现潜在的安全问题。*
--list-operations	列出 virt-sysprep 程序支持的操作。它们按照一行列出，包含一个或多个单空格分隔字段。输出中的第一个字段是操作名称，可以提供给 **--enable** 标志。如果默认禁用操作，或者为空白，则第二个字段是 * 字符。同一行中的其他字段包括操作的描述。	**virt-sysprep --list-operations**

命令	描述	示例
--mount-options	为客户机虚拟机中的每个挂载点设置挂载选项。使用以分号分隔的 mountpoint:options 对列表。您可能需要将该列表用引号放在此列表上，以保护它免受 shell 的影响。	**virt-sysprep --mount-options "/:notime"** 将使用 **notime** 操作挂载根目录。
-q 或 --quiet	防止打印日志消息。	**virt-sysprep -q**
-v 或 --verbose	为调试启用详细消息。	**virt-sysprep -v**
-v 或 --version	显示 virt-sysprep 版本号并退出。	**virt-sysprep -V**
--root-password	设置 root 密码。可用于显式指定新密码，或者使用所选文件的第一行中的字符串（更安全）。	**virt-sysprep --root-password password:***123456* **-a** *guest.img* 或 **virt-sysprep --root-password 文件：***SOURCE_FILE_PATH* **-a** *guest.img*

如需更多信息，请参阅 libguestfs 文档。

21.12. VIRT-CUSTOMIZE：自定义虚拟机设置

virt-customize 命令行工具可用于自定义虚拟机。例如，通过安装软件包并编辑配置文件：

要使用 virt-customize，客户端虚拟机必须离线，因此您必须在运行命令前关闭它。请注意，virt-customize 修改客户机或磁盘镜像而无需复制它。如果要保留客户端虚拟机的现有内容，则必须首先复制或克隆磁盘。有关复制和克隆磁盘的更多信息，请参阅 libguestfs.org。

警告

在实时虚拟机上使用 virt-customize，或其他磁盘编辑工具可导致磁盘崩溃。在使用 此命令之前，必须 关闭虚拟机。另外，不应同时编辑磁盘镜像。

建议您不要以 root 身份运行 virt-customize。

要安装 virt-customize，请运行以下命令之一：

```
# yum install /usr/bin/virt-customize
```

或

```
# yum install libguestfs-tools-c
```

以下命令选项可用于 virt-customize ：

表 21.2. virt-customize 选项

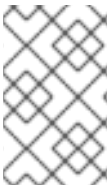
命令	描述	示例
--help	显示有关特定命令或 **virt-customize** 工具的简短帮助条目。如需了解更多帮助，请参阅 virt-customize man page。	**virt-customize --help**
-a [*file*] 或 --add [*file*]	添加 *指定的文件*，它应该是 guest 虚拟机的磁盘映像。磁盘镜像的格式被自动探测到。要覆盖此选项并强制使用特定格式，请使用 **--format** 选项。	**virt-customize --add /dev/vms/disk.img**
-a [*URI*] 或 --add [*URI*]	添加远程磁盘。URI 格式与 guestfish 兼容。如需更多信息，请参阅 第 21.4.2 节 “使用 guestfish 添加文件”。	**virt-customize -a rbd://example.com[:port]/pool/disk**
-c [*URI*] 或 --connect [*URI*]	如果使用 libvirt，则连接到给定的 URI。如果省略，它会通过 KVM 管理程序进行连接。如果您直接指定客户机块设备(**virt-customize -a**)，则根本不使用 libvirt。	**virt-customize -c qemu:///system**
-d [*guest*] 或 --domain [*guest*]	从指定的客户机虚拟机中添加所有磁盘。可以使用域 UUID 而不是域名。	**virt-customize --domain *90df2f3f-8857-5ba9-2714-7d95907b1c9e***

命令	描述	示例
-n 或 --dry-run	在客户机虚拟机上执行只读"dry run"自定义操作。这会运行自定义操作，但会丢弃对磁盘末尾的任何更改。	**virt-customize -n**
--format [**raw**|**qcow2**|**auto**]	-a 选项的默认值是自动检测磁盘镜像的格式。使用此选项强制使用命令行中的 -a 选项的磁盘格式。使用 **--format** 自动检测自动探测到后续 -a 选项（请参阅上面的 **-a** 命令）。	**virt-customize --format raw -a *disk.img*** 会为 disk.img 强制原始格式(no auto-detection)，但 **virt-customize --format raw -a *disk.img* --format auto -a *another.img*** *会强制为* **disk.img** *强制（无自动探测）强制自动探测* ***到另一个.img****。如果您有不受信任的原始格式客户机磁盘镜像，则应该使用这个选项指定磁盘格式。这可避免恶意客户端出现潜在的安全问题。*
-m [*MB*] 或 --memsize [*MB*]	更改分配给 **--run** 脚本的内存量。如果 **--run** 脚本或 **--install** 选项会导致内存问题，请增加内存分配。	**virt-customize --memsize 1024**
--network 或 --no-network	在安装过程中启用或禁用客户机的网络访问。默认为 enabled。使用 **--no-network** 禁用访问。此命令不会影响启动后对网络进行客户机访问。如需更多信息，请参阅 libguestfs 文档。	**virt-customize -a http://[user@]example.com[:port]/disk.img**
-q 或 --quiet	防止打印日志消息。	**virt-customize -q**
-smp [*N*]	启用可由 **--install** 脚本使用的 *N* 个虚拟 CPU。*N* 必须为 2 或更多。	**virt-customize -smp 4**
-v 或 --verbose	为调试启用详细消息。	**virt-customize --verbose**
-v 或 --version	显示 **virt-customize** 版本号并退出。	**virt-customize --V**
-x	启用 libguestfs API 调用追踪。	**virt-customize -x**

***virt-customize** 命令使用自定义选项来配置客户机的定制方式。以下提供了有关 **--selinux-relabel** 自定义选项的信息。*

***selinux -relabel** 自定义选项重新标记虚拟客户机中的文件，以便它们具有正确的 SELinux 标签。此*

选项尝试立即重新标记文件。如果失败，则镜像上激活 /.autorelabel。这会将重新标记操作调度到下一次镜像引导时。



注意

这个选项只应用于支持 SELinux 的 guest。

以下示例在客户机上安装 GIMP 和 Inkscape 软件包，并确保 SELinux 标签将在虚拟机下次启动时正确。

例 21.1. 使用 virt-customize 在客户端中安装软件包

```
virt-customize -a disk.img --install gimp,inkscape --selinux-relabel
```

如需更多信息，包括自定义选项，请参阅 libguestfs.org。

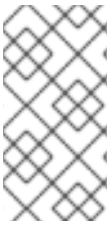
21.13. VIRT-DIFF:列出虚拟机文件间的差异

virt-diff 命令行工具可用于列出两个虚拟机磁盘映像中的文件差异。输出显示了虚拟机运行之后虚拟机磁盘映像的更改。命令也可用于显示覆盖的不同。

注意

您可以在实时虚拟机上安全使用 virt-diff，因为它只需要只读访问。

此工具在文件名、文件大小、校验和、扩展属性、文件内容以及运行的镜像和所选镜像之间找到了差异。



注意

virt-diff 命令不检查引导加载程序、分区或文件系统中未使用的空间，或者"hidden"扇区。因此，建议您不要将此用作安全或取证工具。

要安装 virt-diff，请运行以下命令之一：

```
# yum install /usr/bin/virt-diff
```

或

```
# yum install libguestfs-tools-c
```

要指定两个虚拟机，您必须在第一个 guest 中使用 -a 或 -d 选项，第二个客户机使用 -A 或 -D 选项。例如：

```
$ virt-diff -a old.img -A new.img
```

您还可以使用 libvirt 已知的名称。例如：

```
$ virt-diff -d oldguest -D newguest
```

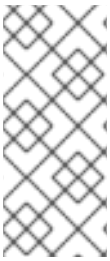
以下命令选项可与 virt-diff 一起使用：

表 21.3. virt-diff 选项

命令	描述	示例
--help	显示有关特定命令或有关 **virt-diff** 实用程序的简短帮助条目。详情请查看 virt-diff man page。	**virt-diff --help**
-a [*file*] 或 --add [*file*]	添加 *指定的文件*，它应该是第一个虚拟机的磁盘镜像。如果虚拟机有多个块设备，您必须为所有这些设备提供单独的 **-a** 选项。 磁盘镜像的格式被自动探测到。要覆盖此选项并强制使用特定格式，请使用 **--format** 选项。	**virt-customize --add /dev/vms/original.img -A /dev/vms/new.img**
-a [*URI*] 或 --add [*URI*]	添加远程磁盘。URI 格式与 guestfish 兼容。如需更多信息，请参阅 第 21.4.2 节 “使用 guestfish 添加文件”。	**virt-diff -a rbd://example.com[:port]/pool/newdisk -A rbd://example.com[:port]/pool/olddisk**
--all	与 **--extra-stats --times --uids --xattrs** 相同。	**virt-diff --all**

命令	描述	示例
--atime	默认情况下，**virt-diff** 忽略文件访问时间的变化，因为它们不太可能有趣。使用 **--atime** 选项显示访问时间差异。	**virt-diff --atime**
-a [*file*]	添加 *指定的文件* 或 *URI*，它应该是第二个虚拟机的磁盘镜像。	**virt-diff --add /dev/vms/original.img -A /dev/vms/new.img**
-c [*URI*] 或 --connect [*URI*]	如果使用 libvirt，则连接到给定的 URI。如果省略，则它连接到默认的 libvirt 管理程序。如果您直接指定 guest 块设备(**virt-diff -a**)，那么根本不使用 libvirt。	**virt-diff -c qemu:///system**
--csv	提供采用逗号分隔的值(CSV)格式的结果。此格式可以轻松导入到数据库和电子表格中。有关详情请参考 注意。	virt-diff --csv
-d [*guest*] 或 --domain [*guest*]	将指定客户机虚拟机中的所有磁盘添加为第一个客户机虚拟机。可以使用域 UUID 而不是域名。	$ **virt-diff --domain *90df2f3f-8857-5ba9-2714-7d95907b1c9e***
-d [*guest*]	将指定客户机虚拟机中的所有磁盘添加为第二个客户机虚拟机。可以使用域 UUID 而不是域名。	**virt-diff --D *90df2f3f-8857-5ba9-2714-7d95907b1cd4***
--extra-stats	显示额外的统计数据。	**virt-diff --extra-stats**
--format 或 --format= [**raw**|**qcow2**]	**-a**/**-A** 选项的默认值是自动检测磁盘镜像的格式。使用此选项强制使用命令行上的 **-a**/**-A** 选项的磁盘格式。使用 **--format** 自动检测自动探测到后续 -a 选项（请参阅上面的 **-a** 命令）。	**virt-diff --format raw -a *new.img* -A *old.img*** 为 new.img 和 old.img 强制执行原始格式(no auto-detection)，但 **virt-diff --format raw -a *new.img* --format auto -a *old.img*** 会强制使用 raw 格式（无自动探测）用于 **new.img**，并恢复为 **old.img** 的 auto-detectionion。如果您有不受信任的原始格式客户机磁盘镜像，则应该使用这个选项指定磁盘格式。这可避免恶意客户端出现潜在的安全问题。
-h 或 --human-readable	以可读格式显示文件大小。	**virt-diff -h**

命令	描述	示例
--time-days	将更改文件的时间字段显示前的天数（如果在将来使用的话）。 请注意，输出中的 **0** 表示在将来的 86,399 秒（23 小时、59 分钟和 59 秒）之间。	**virt-diff --time-days**
-v 或 --verbose	为调试启用详细消息。	**virt-diff --verbose**
-v 或 --version	显示 **virt-diff** 版本号并退出。	**virt-diff -V**
-x	启用 libguestfs API 调用追踪。	**virt-diff -x**



注意

以逗号分隔的值(CSV)格式可能难以解析。因此，建议对于 shell 脚本，您应该使用 csvtool 和其他语言，使用 CSV 处理库（如 Perl 或 Python 内置 csv 库的 Text::CSV）。另外，大多数电子表格和数据库可直接导入 CSV。

如需更多信息，包括附加选项，请参阅 libguestfs.org。

21.14. VIRT-SPARSIFY:回收 EMPTY 磁盘空间

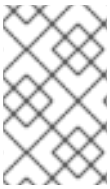
virt-sparsify 命令行工具可用于生成虚拟机磁盘（或任何磁盘镜像）稀疏。这也被称为精简配置。磁盘镜像上的可用磁盘空间转换为主机上可用空间。

virt-sparsify 命令可以处理大多数文件系统，如 ext2、ext3、ext4、btrfs、NTFS。它还可用于 LVM 物理卷。virt-sparsify 可以针对任何磁盘镜像操作，而不只是虚拟机磁盘镜像。

警告

使用 virt-sparsify on live 虚拟机或与其他磁盘编辑工具同时存在，可能会导致磁盘崩溃。在使用 此命令之前，必须 关闭虚拟机。另外，不应同时编辑磁盘镜像。

命令也可用于在某些磁盘格式间进行转换。例如，virt-sparsify 可以将原始磁盘镜像转换为精简置备的 qcow2 镜像。



注意

如果虚拟机有多个磁盘并使用卷管理，virt-sparsify 将正常工作，但它不会非常有效。

如果输入是 raw，则默认输出为 raw sparse。输出镜像的大小必须使用了解稀疏性的工具进行检查。

```
$ ls -lh test1.img
-rw-rw-r--. 1 rjones rjones 100M Aug  8 08:08 test1.img
$ du -sh test1.img
3.6M   test1.img
```

请注意，ls 命令将映像大小显示为 100M。但是，du 命令可以正确地显示到 3.6M 的镜像大小。

重要限制

以下是一些重要限制列表：

- 在使用 virt-sparsify 前必须关闭虚拟机。
- 在最糟糕的情况中，virt-sparsify 可能需要最多 2 个源磁盘镜像的虚拟大小。用于临时复制，另一个用于目标镜像。

  如果您使用 --in-place 选项，则不需要大量的临时空间。
- virt-sparsify 无法用于调整磁盘镜像大小。要重新定义磁盘镜像大小，请使用 virt-resize。有关 virt-resize 的详情请参考 第 21.8 节 “virt-resize：重新定义虚拟机大小”。
- virt-sparsify 无法使用加密磁盘，因为加密磁盘无法被解析。
- virt-sparsify 无法对分区之间的空间进行解析。此空间通常用于引导加载程序等关键项目，因此这不是实际未使用的空间。

- 在 复制 模式中，qcow2 内部快照不会复制到目的地镜像。

示例

要安装 virt-sparsify，请运行以下命令之一：

```
# yum install /usr/bin/virt-sparsify
```

或

```
# yum install libguestfs-tools-c
```

sparsify 一个磁盘：

```
# virt-sparsify /dev/sda1 /dev/device
```

将 /dev/sda1 的内容复制到 /dev/device，使输出稀疏。如果 /dev/device 已存在，它会被覆盖。已检测到 /dev/sda1 格式，并用作 /dev/device 的格式。

在格式间进行转换：

```
# virt-sparsify disk.raw --convert qcow2 disk.qcow2
```

尝试在源磁盘镜像中找到的每个文件系统上归零并报出可用空间。

要防止特定文件系统上的零覆盖可用空间：

```
# virt-sparsify --ignore /dev/device /dev/sda1 /dev/device
```

从磁盘镜像中的所有文件系统创建稳定的磁盘映像，而不会用零覆盖文件系统上的可用空间。

在不创建临时副本的情况下生成磁盘镜像稀疏：

```
# virt-sparsify --in-place disk.img
```

使指定的磁盘镜像稀疏，覆盖镜像文件。

virt-sparsify 选项

以下命令选项可用于 virt-sparsify:

表 21.4. virt-sparsify 选项

命令	描述	示例
--help	显示有关特定命令或 **virt-sparsify** 实用程序的简短帮助条目。详情请查看 virt-sparsify man page。	**virt-sparsify --help**
--check-tmpdir **ignore**|**继续**|**warn**|**失败**	估计，如果 *tmpdir* 有足够的空间来完成操作。如果没有足够的空间来完成操作，则指定选项将决定该行为。 • **忽略** ：该问题将被忽略，并且操作继续进行。 • **继续** ：报告错误，并且操作将继续。 • **warn**:报告错误并等待用户按 Enter 键。 • **失败** ：报告错误并中止操作。 这个选项不能与 **--in-place** 选项一起使用。	**virt-sparsify --check-tmpdir ignore /dev/sda1 /dev/device** **virt-sparsify --check-tmpdir continue /dev/sda1 /dev/device** **virt-sparsify --check-tmpdir warn /dev/sda1 /dev/device** **virt-sparsify --check-tmpdir 失败 /dev/sda1 /dev/device**
--compress	压缩输出文件。*这只有在* 输出格式是 qcow2 时有效。这个选项不能与 **--in-place** 选项一起使用。	**virt-sparsify --compress /dev/sda1 /dev/device**
--convert	使用指定格式创建稀疏镜像。如果没有指定格式，则使用输入格式。 支持以下输出格式并已知可以正常工作：raw、qcow、vdi 您可以使用 QEMU 模拟器支持的任何格式。 建议您使用 **--convert** 选项。这样，**virt-sparsify** 不需要猜测输入格式。 这个选项不能与 **--in-place** 选项一起使用。	**virt-sparsify --convert raw /dev/sda1 /dev/device** **virt-sparsify --convert qcow2 /dev/sda1 /dev/device** **virt-sparsify --convert *other_format indisk outdisk***

命令	描述	示例
--format	指定输入磁盘镜像的格式。如果没有指定，则会从镜像中检测到格式。在使用不受信任的原始格式客户机磁盘镜像时，请确保指定格式。	**virt-sparsify --format raw /dev/sda1 /dev/device** **virt-sparsify --format qcow2 /dev/sda1 /dev/device**
--ignore	忽略指定的文件系统或卷组。 当指定文件系统且未指定 **--in-place** 选项时，文件系统的可用空间不会为零。但是，现有零块被 sparsified。当指定 **--in-place** 选项时，该文件系统将完全忽略。 当指定卷组时，卷组将被忽略。应该在没有 **/dev/** 前缀的情况下使用卷组名称。例如： **--ignore vg_foo** **--ignore** 选项可以多次包含在命令中。	**virt-sparsify --ignore filesystem1 /dev/sda1 /dev/device** **virt-sparsify --ignore volume_group/dev/sda1 /dev/device**
--in-place	使镜像稀疏原位，而不是制作临时副本。虽然原位问题比复制问题更高效，但它不能像复制差异一样恢复太多的磁盘空间。使用丢弃（也称为 trim 或 unmap）支持进行原位升级。 要使用原位解析，请指定一个要进行替换的磁盘镜像。 当指定原位升级时，无法使用以下选项： • **--convert** 和 **--compress**，因为它们需要批量磁盘格式更改。 **--check-tmpdir**，因为不需要大量临时空间。	**virt-sparsify --in-place disk.img**
-x	启用 libguestfs API 调用追踪。	**virt-sparsify -x filesystem1 /dev/sda1 /dev/device**

如需更多信息，包括附加选项，请参阅 libguestfs.org。

第 22 章 用于客户机虚拟机管理的图形用户界面工具

除了 virt-manager 外，Red Hat Enterprise Linux 7 还提供以下工具，可让您访问 guest 虚拟机的控制台。

22.1. VIRT-VIEWER

virt-viewer 是一个最小的命令行实用程序，用于显示客户机虚拟机的图形控制台。控制台可使用 VNC 或 SPICE 协议访问。guest 可以通过其名称、ID 或 UUID 指代。如果客户机还没有运行，可以将查看器设置为在尝试连接到控制台前等待其启动。查看器可以连接到远程主机以获取控制台信息，然后使用同一网络传输连接到远程控制台。

与 virt-manager 相比，virt-viewer 提供了一组较小的功能，但资源需求要少。另外，与 virt-manager 不同，在大多数情形中，virt-viewer 不需要对 libvirt 的读写权限。因此，它可以被授权用户用来连接到和显示客户机，但不能配置它们。

要安装 virt-viewer，请运行：

```
# yum install virt-viewer
```

Syntax

基本 virt-viewer 命令行语法如下：

```
# virt-viewer [OPTIONS] {guest-name|id|uuid}
```

要查看可与 virt-viewer 一起使用的完整选项列表，请查看 virt-viewer man page。

连接到客户机虚拟机

如果不带任何选项使用，virt-viewer 将列出可以在本地系统的默认虚拟机监控程序上连接到的客户机。

连接到使用默认虚拟机监控程序的指定客户机虚拟机：

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

连接到使用 KVM-QEMU 管理程序的客户机虚拟机：

```
# virt-viewer --connect qemu:///system guest-name
```

使用 TLS 连接到远程控制台：

```
# virt-viewer --connect qemu://example.org/ guest-name
```

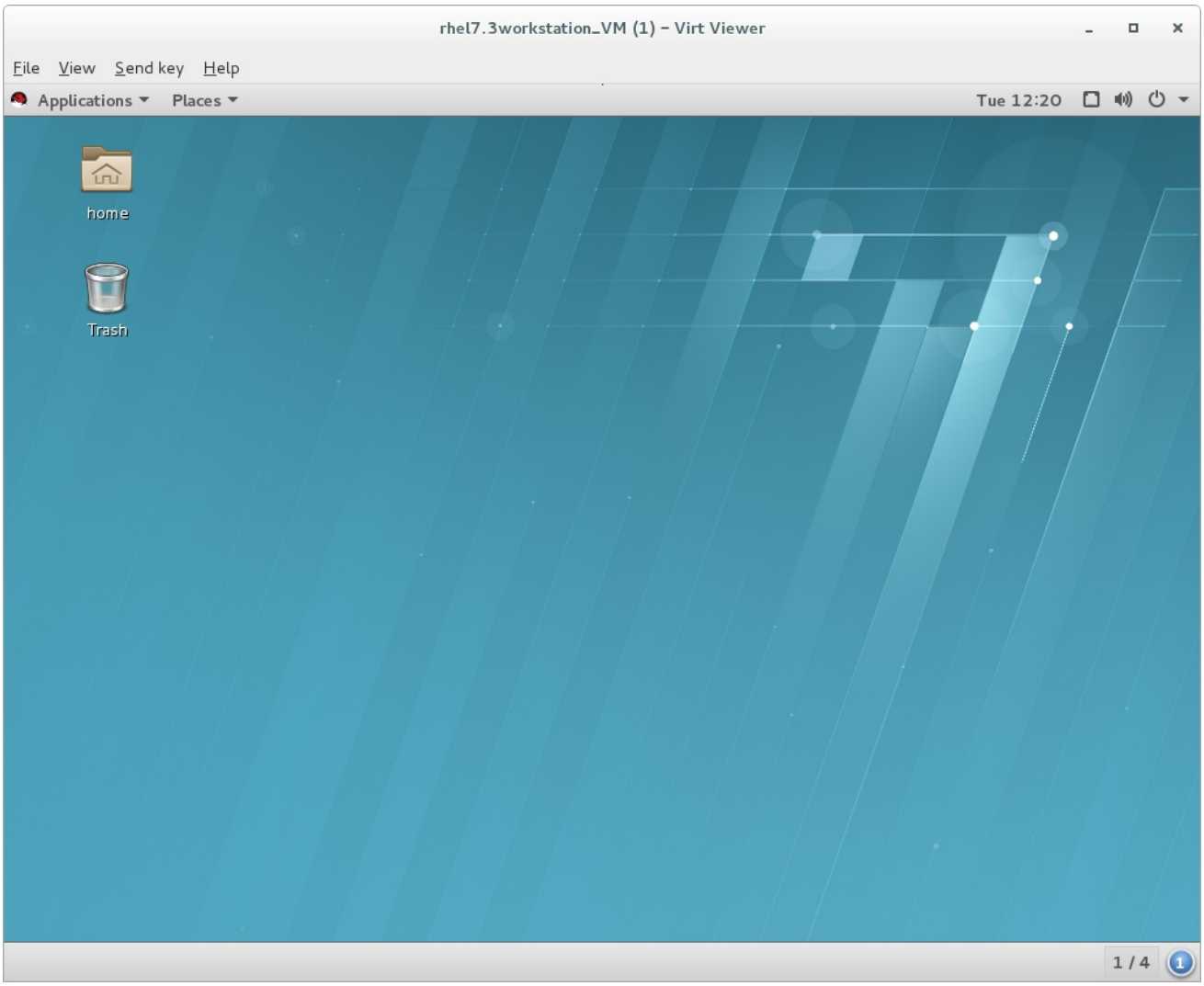
要使用 SSH 连接到远程主机上的控制台，请查找客户机配置，然后向控制台直接进行非隧道连接：

```
# virt-viewer --direct --connect qemu+ssh://root@example.org/ guest-name
```

Interface

默认情况下，virt-viewer 接口只提供与客户机交互的基本工具：

图 22.1. virt-viewer 接口示例



设置热密钥

*要为 **virt-viewer** 会话创建自定义键盘快捷键（也称为热键），请使用 **--hotkeys** 选项：*

```
# virt-viewer --hotkeys=action1=key-combination1[,action2=key-combination2] guest-name
```

可将以下操作分配给一个热密钥：

- *切换完整屏幕*
- ***release-cursor***
- ***SmartCard-insert***

- SmartCard-remove

key-name 组合热键不区分大小写。请注意，热密钥设置不会切换到未来的 virt-viewer 会话。

例 22.1. 设置 virt-viewer 热密钥

在连接到名为 testguest 的 KVM-QEMU 客户机时，添加热键更改为完整屏幕模式：

```
# virt-viewer --hotkeys=toggle-fullscreen=shift+f11 qemu:///system testguest
```

kiosk 模式

在 kiosk 模式中，virt-viewer 仅允许用户与连接的桌面交互，并且不提供与客户机设置或主机系统交互的任何选项，除非 guest 关闭。当管理员希望限制用户对指定 guest 的操作范围时，这很有用。

要使用 kiosk 模式，请使用 -k 或 --kiosk 选项连接到 guest。

例 22.2. 在 kiosk 模式中使用 virt-viewer

要在虚拟机关闭后以 kiosk 模式连接到 KVM-QEMU 虚拟机，请使用以下命令：

```
# virt-viewer --connect qemu:///system guest-name --kiosk --kiosk-quit on-disconnect
```

但请注意，只有 kiosk 模式，无法确保用户在关闭后与主机系统或客户机设置进行交互。这需要进一步的安全措施，如禁用主机上的窗口管理器。

22.2. REMOTE-VIEWER

remote-viewer 是支持 SPICE 和 VNC 的简单远程桌面显示客户端。它与 virt-viewer 分享大部分功能和局限性。

但是，与 virt-viewer 不同，remote-viewer 不需要 libvirt 连接到远程 guest 显示。因此，remote-viewer 可用于连接到远程主机上的虚拟机，它们不提供与 libvirt 交互或使用 SSH 连接的权限。

要安装 remote-viewer 工具，请运行：

```
# yum install virt-viewer
```

Syntax

基本的 remote-viewer 命令行语法如下：

```
# remote-viewer [OPTIONS] {guest-name|id|uuid}
```

要查看可用于 remote-viewer 的选项的完整列表，请参阅 remote-viewer man page。

连接到客户机虚拟机

如果不使用任何选项，remote-viewer 将列出可以在本地系统的默认 URI 上连接到的客户机。

要使用 remote-viewer 连接到特定 guest，请使用 VNC/SPICE URI。有关获取 URI 的详情，请参考 第 20.14 节 “显示到图形显示的连接的 URI”。

例 22.3. 使用 SPICE 连接到虚拟机显示

使用以下命令连接到名为"testguest"的机器上的 SPICE 服务器，它使用端口 5900 进行 SPICE 通信：

```
# remote-viewer spice://testguest:5900
```

例 22.4. 使用 VNC 连接到客户机显示

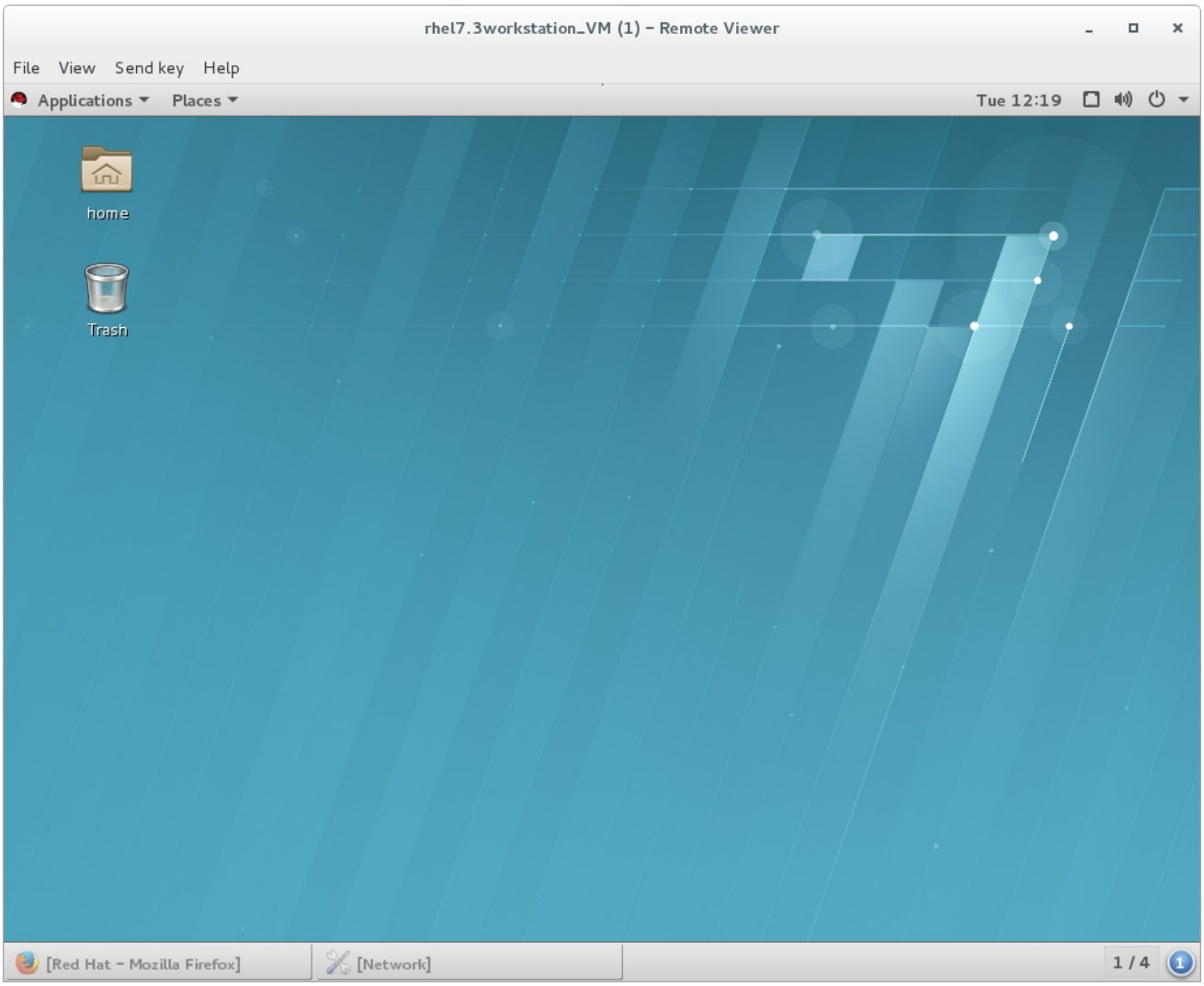
使用以下命令连接到名为 testguest2 的机器上的 VNC 服务器，它使用端口 5900 进行 VNC 通信：

```
# remote-viewer vnc://testguest2:5900
```

Interface

默认情况下，remote-viewer 接口只提供与客户机交互的基本工具：

图 22.2. remote-viewer 接口示例



22.3. GNOME BOXES

Box 是一个轻量级图形化的桌面虚拟化工具，可用于查看和访问虚拟机和远程系统。

与 virt-viewer 和 remote-viewer 不同，Bxes 允许查看客户机虚拟机，同时创建和配置它们，类似于 virt-manager。但是，与 virt-manager 进行比较，Bxes 提供了较少的管理选项和功能，但更易于使用。

要安装 Boxes，请运行：

```
# yum install gnome-boxes
```

通过 应用程序 工具 打开 Boxes。

主屏幕显示了可用的客户机虚拟机。屏幕右侧有两个按钮：

- 
  - 搜索按钮，按名称搜索 guest 虚拟机，以及
- 
  - 选择按钮。

单击选择按钮，您可以选择一个或多个客户机虚拟机，以单独或作为组执行操作。可用的操作显示在操作栏中屏幕的底部：

图 22.3. 操作栏

有四个操作可以被执行：

- 喜欢 ：为选定的虚拟机添加核心，并将它们移到客户机列表的顶部。随着客户机数量的增加，这变得越来越有帮助。
- 暂停 ：所选客户机虚拟机将停止运行。
- 删除:删除所选 guest 虚拟机。
- 属性 ：显示所选 guest 虚拟机的属性。

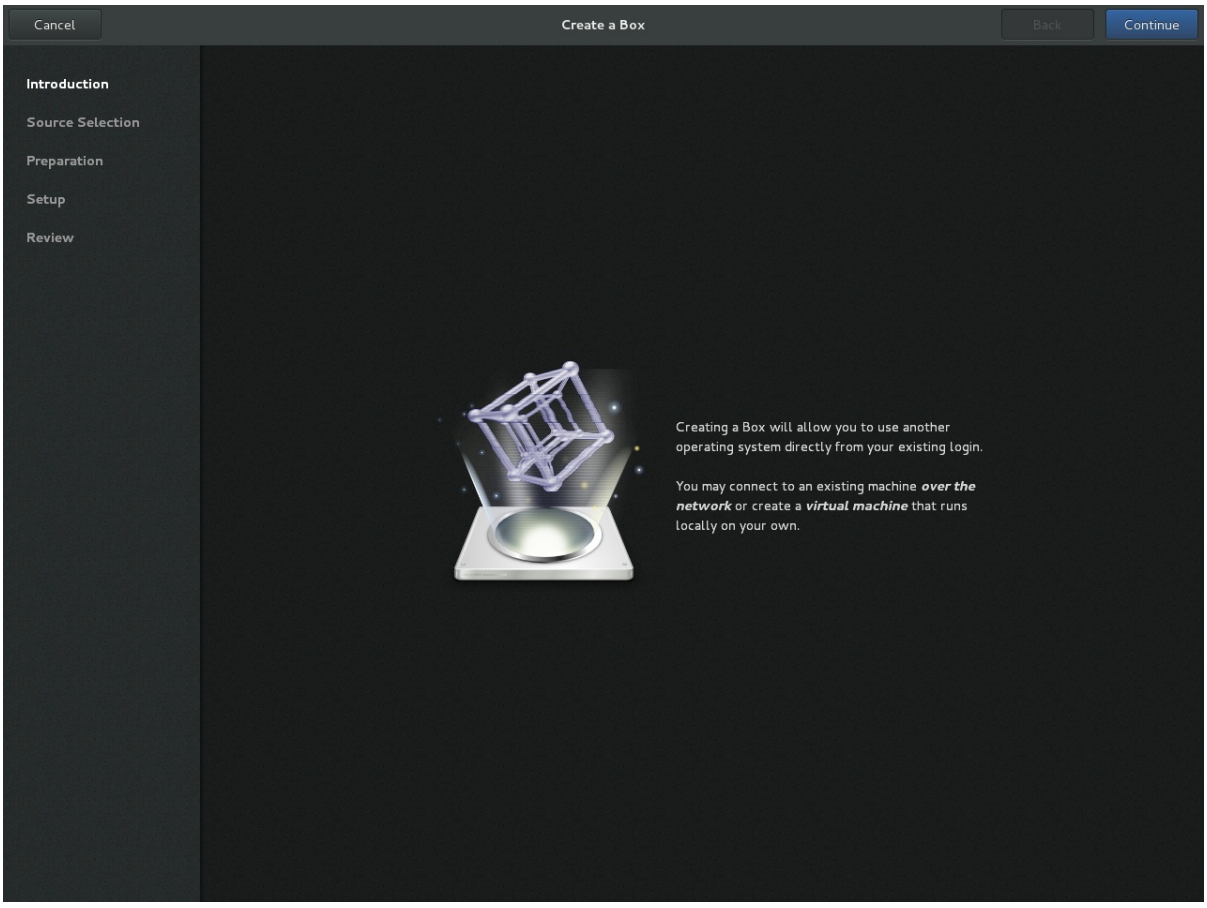
使用主屏幕左侧的 新建按钮，创建新的 guest 虚拟机。

过程 22.1. 使用 Boxes 创建新的客户机虚拟机

1. 点 New

   此时将打开 Introduction 屏幕。点 Continue。

图 22.4. 简介屏幕



2. 选择源

   Source Selection 屏幕有三个选项：

   - 可用介质：此处会显示任何立即可用的安装介质。点击其中任何一个都可以直接进入 Review 屏幕。

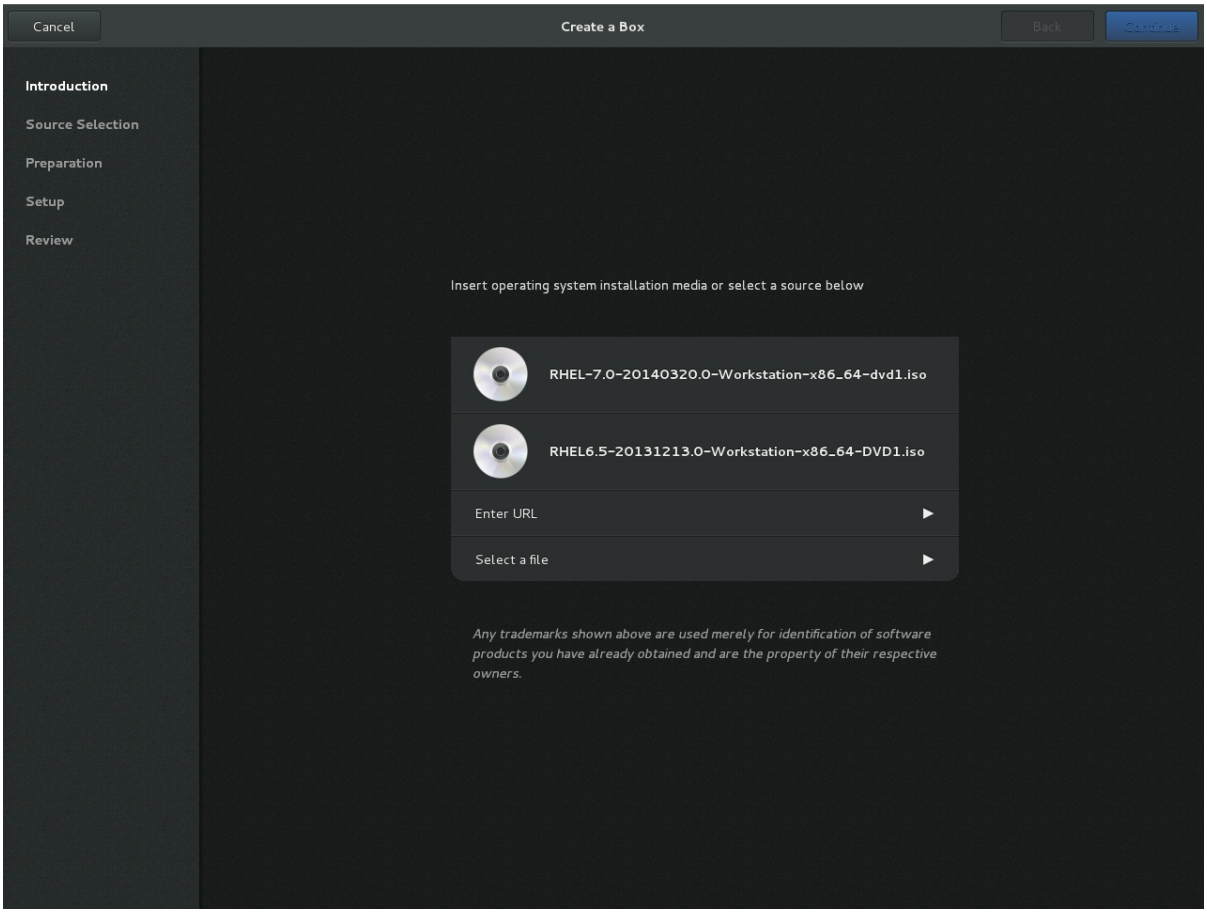
   - 输入 URL:在 URL 中键入，以指定本地 URI 或 ISO 文件的路径。这也可用于访问远程机器。该地址应遵循 协议://IPaddress?端口 的模式，例如：

     ```
     spice://192.168.122.1?port=5906;
     ```

协议可以是 spice://、qemu:// 或 vnc://

- *选择一个文件 ：打开文件目录，手动搜索安装介质。*

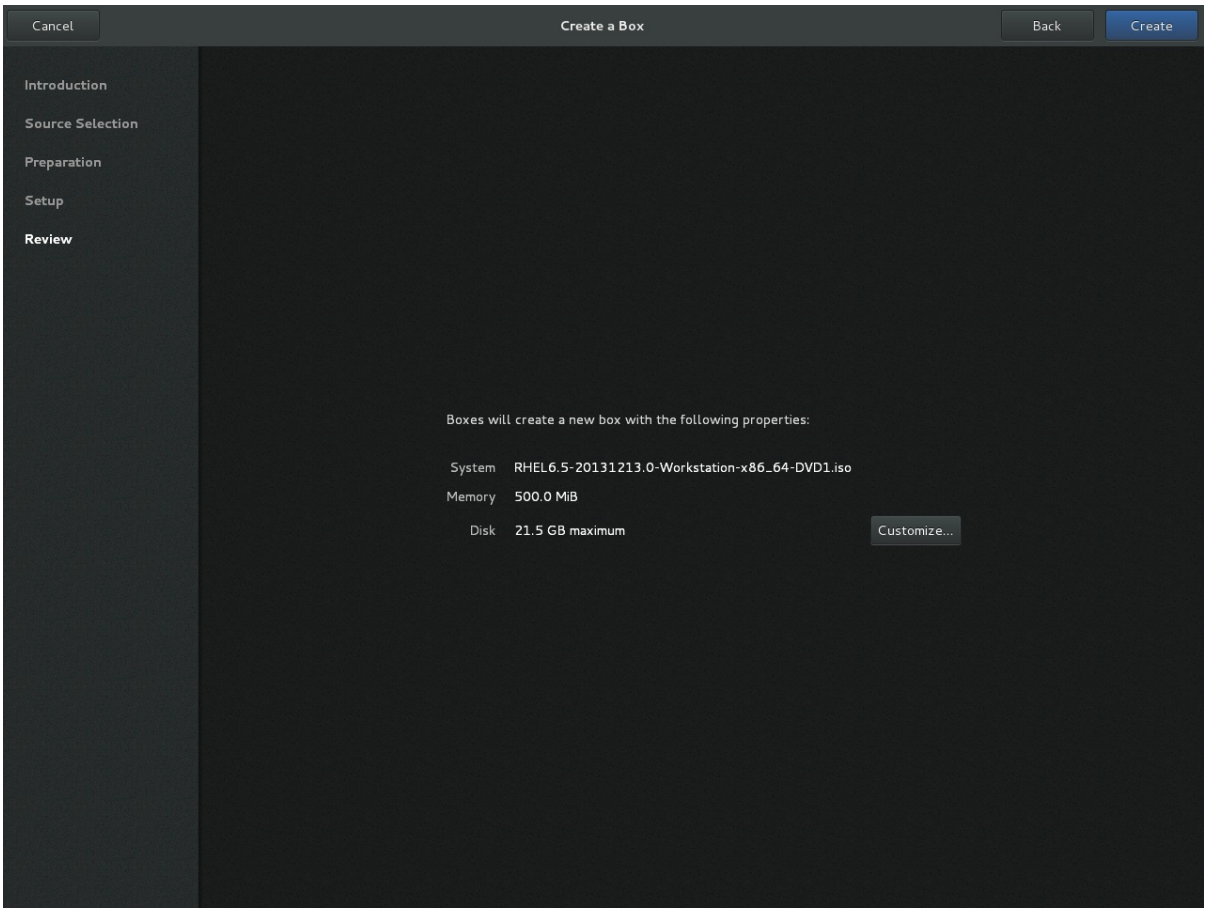
图 22.5. 源选择屏幕



3. *查看详情*

   Review 屏幕显示客户机虚拟机的详细信息。

图 22.6. 查看页面

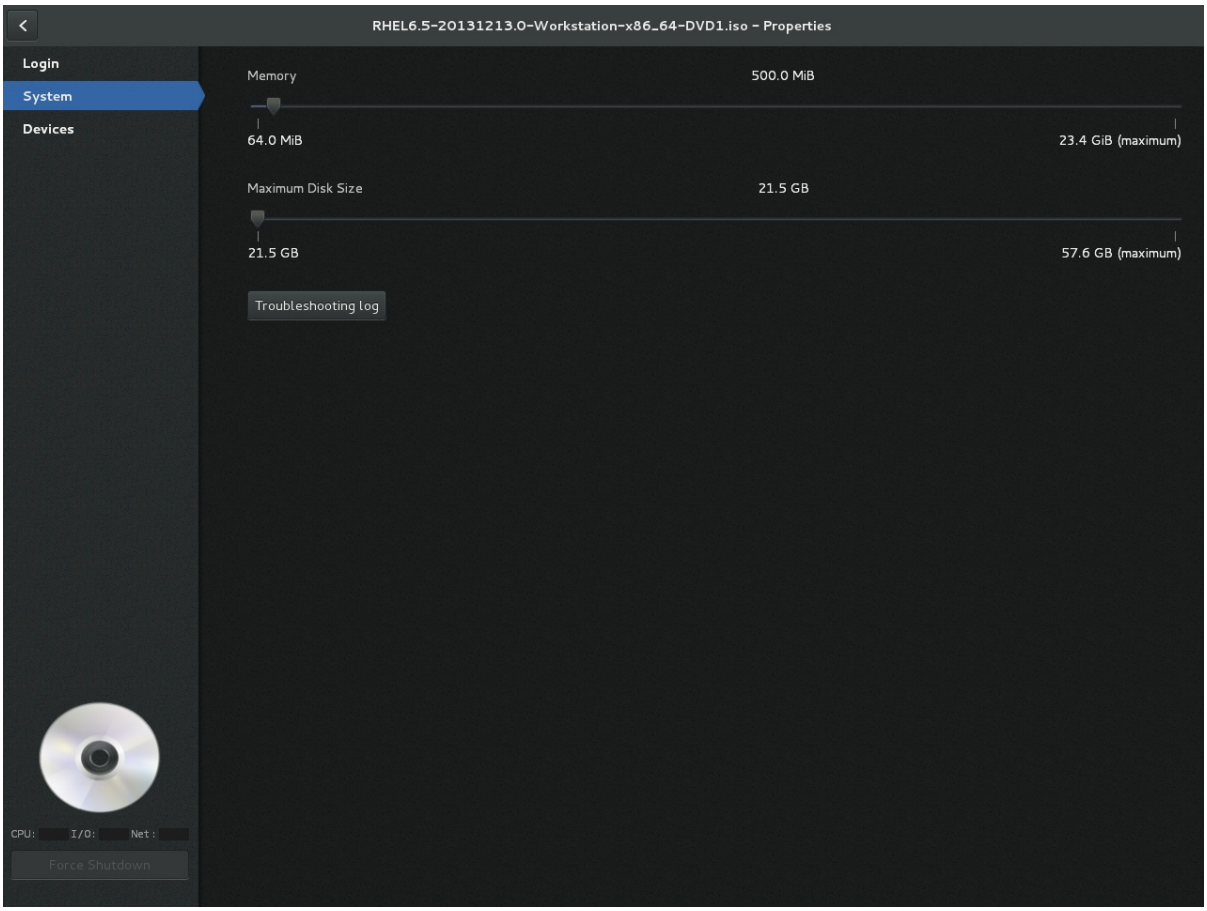


这些详细信息可保留为原样，在那里进入最终步骤，或者：

4. *可选：自定义详情*

   单击 Customize，您可以调整客户机虚拟机的配置，如内存和磁盘大小。

图 22.7. 自定义屏幕



5. *创建*

   点 Create。将打开新的 guest 虚拟机。

第 23 章 操作域 XML

本章解释了虚拟客户机 XML 配置文件的组件，也称为 域 XML。在本章中，术语 域 指的是所有客户机虚拟机所需的 root <域> 元素。域 XML 有两个属性： type 和 id。type 指定用于运行域的虚拟机监控程序。允许的值是特定于驱动程序的，但包括 KVM 和其它值。id 是运行的客户机虚拟机的唯一整数标识符。非活动机器没有 id 值。本章中的部分将描述域 XML 的组件。本手册中的其他章节在操作域 XML 时可能会看到本章。

重要

仅使用支持的管理接口（如 virsh 和 虚拟机管理器）和命令（如 virt-xml）编辑域 XML 文件的组件。不要通过文本编辑器直接打开和编辑域 XML 文件。如果您绝对必须直接编辑域 XML 文件，请使用 virsh edit 命令。

注意

本章基于 libvirt 上游文档。

23.1. 通用信息和元数据

这些信息包括在域 XML 的这个部分：

图 23.1. 域 XML 元数据

```
<domain type='kvm' id='3'>
  <name>fv0</name>
  <uuid>4dea22b31d52d8f32516782e98ab3fa0</uuid>
  <title>A short description - title - of the domain</title>
  <description>A human readable description</description>
  <metadata>
    <app1:foo xmlns:app1="http://app1.org/app1/">..</app1:foo>
    <app2:bar xmlns:app2="http://app1.org/app2/">..</app2:bar>
  </metadata>
  ...
</domain>
```

域 XML 的这一部分的组件如下：

表 23.1. 常规元数据元素

元素	描述
<名称>	为虚拟机分配名称。此名称仅由字母数字字符组成，必须在单一主机物理计算机范围内保持唯一。它通常用于组成存储持久配置文件的文件名。
<uuid>	为虚拟机分配全局唯一标识符。格式必须符合 RFC 4122- 兼容，例如 **3e3fce45-4f53-4fa7-bb32-11f34168b82b**。如果在定义或创建新机器时省略，将生成随机 UUID。也可以使用 **sysinfo** 规范提供 UUID。
<title>	为域的简短描述创建空格。标题不应包含任何新行。
<描述>	与标题不同，libvirt 不会使用此数据。它可以包含用户选择想要显示的任何信息。
<metadata>	应用程序可以使用它以 XML 节点/树的形式存储自定义元数据。应用程序必须在 XML 节点/树上使用自定义命名空间，每个命名空间只有一个顶级元素（如果应用程序需要结构，则应具有子元素作为其命名空间元素）。

23.2. 操作系统引导

引导虚拟机的方法有多种，包括 BIOS 引导装载程序、主机物理机器引导加载程序、直接内核引导和容器引导。

23.2.1. BIOS Boot Loader

启动 BIOS 可用于支持完全虚拟化的虚拟机监控程序。在这种情况下，BIOS 具有引导顺序优先级（floppy、硬盘、CD-ROM、网络）决定从何处定位引导镜像。域 XML 的 <os> 部分包含以下信息：

图 23.2. BIOS 引导装载程序域 XML

```
...
<os>
  <type>hvm</type>
  <boot dev='fd'/>
  <boot dev='hd'/>
  <boot dev='cdrom'/>
  <boot dev='network'/>
  <bootmenu enable='yes'/>
  <smbios mode='sysinfo'/>
  <bios useserial='yes' rebootTimeout='0'/>
</os>
...
```

域 XML 的这一部分的组件如下：

表 23.2. BIOS 引导装载程序元素

元素	描述
<type>	指定要在客户机虚拟机上引导的操作系统类型。**hvm** 表示操作系统设计为在裸机上运行，需要完全虚拟化。**Linux** 指的是支持 KVM 管理程序客户机 ABI 的操作系统。另外有两个可选属性： **arch** 为虚拟化指定 CPU 架构，**机器则** 引用机器类型。如需更多信息，请参阅 *libvirt 上游文档*。
<boot>	指定要考虑使用以下值之一的下一个引导设备：**fd**、hd、**cdrom** 或 **network**。boot 元素可以多次重复，以设置优先级引导设备以便尝试打开。同一类型的多个设备按照目标排序，同时保留总线顺序。定义域后，libvirt 返回的 XML 配置按排序顺序列出设备。进行排序后，第一个设备将标记为可引导。如需更多信息，请参阅 libvirt 上游文档。
<bootmenu>	确定是否在 guest 虚拟机上启用交互式引导菜单提示。**enable** 属性可以是 **yes**，也可以是 **yes**。如果未指定，则使用虚拟机监控程序默认。
<smbios>	决定在客户机虚拟机中如何查看 SMBIOS 信息。必须指定 **mode** 属性，如 **模拟** （允许管理程序生成所有值）、**host** （所有 Block 0 和 Block 1，除了 UUID 外）或 sysinfo（从主机物理机器的 SMBIOS 值）；v**irConnectGetSysinfo** 调用可用于查看要复制的值，或 **sysinfo** （使用 sysinfo 元素中的值）。如果未指定，则使用虚拟机监控程序的默认设置。

元素	描述
<bios>	此元素具有属性 **使用** 可能值 **yes** 或 **no**。属性启用或禁用 Serial Graphics Adapter，它允许用户在串行端口上看到 BIOS 信息。因此，需要定义串行端口。**rebootTimeout** 属性控制客户机虚拟机在引导失败时是否应重新启动的时间（根据 BIOS）。值设为毫秒，最大为 **65535** ；设置 **-1** 可禁用重新启动。

23.2.2. 直接内核引导

安装新的客户端虚拟机操作系统时，直接从存储在主机物理计算机操作系统的内核和 initrd 直接引导，允许命令行参数直接传递给安装程序。这个功能通常用于完全虚拟化和半虚拟化的客户机虚拟机。

图 23.3. 直接内核引导

```
...
<os>
  <type>hvm</type>
  <kernel>/root/f8-i386-vmlinuz</kernel>
  <initrd>/root/f8-i386-initrd</initrd>
  <cmdline>console=ttyS0 ks=http://example.com/f8-i386/os/</cmdline>
  <dtb>/root/ppc.dtb</dtb>
</os>
...
```

域 XML 的这一部分的组件如下：

表 23.3. 直接内核启动元素

元素	描述
<type>	与 BIOS 引导部分所述相同。
<内核>	指定主机物理机器操作系统中内核映像的完全限定路径。
<initrd>	指定主机物理机器操作系统中的 ramdisk 镜像的完全限定域名（可选）。
<cmdline>	指定在引导时要传递给内核（或安装程序）的参数。这通常用于指定备选主控制台（如串行端口），或安装介质源或 kickstart 文件。

23.2.3. 容器引导

当使用基于容器的虚拟化引导域时，使用 init 元素而不是内核或引导镜像时，需要使用 init 二进制文件的路径。默认情况下，它将在没有参数的情况下启动。要指定初始 argv，请使用 initarg 元素，请根据需要多次重复。cmdline 元素提供与 /proc/cmdline 等效的，但不会影响 <initarg>。

图 23.4. 容器启动

```
...
<os>
  <type arch='x86_64'>exe</type>
  <init>/bin/systemd</init>
  <initarg>--unit</initarg>
  <initarg>emergency.service</initarg>
</os>
...
```

23.3. SMBIOS 系统信息

某些虚拟机监控程序可以控制向客户机虚拟机显示哪些系统信息（例如，SMBIOS 字段可以由虚拟机监控程序填充，并使用 guest 虚拟机中的 dmidecode 命令进行检查）。可选的 sysinfo 元素涵盖了此类信息类别。

图 23.5. SMBIOS 系统信息

```
...
<os>
  <smbios mode='sysinfo'/>
  ...
</os>
<sysinfo type='smbios'>
  <bios>
    <entry name='vendor'>LENOVO</entry>
  </bios>
  <system>
    <entry name='manufacturer'>Fedora</entry>
    <entry name='vendor'>Virt-Manager</entry>
  </system>
</sysinfo>
...
```

<sysinfo> 元素具有一个强制属性 类型，用于决定子元素的布局，其定义如下：

- SMBIOS - Sub-elements 调用特定的 SMBIOS 值，如果与 <os> 元素的 <smbios> 子元素结合使用，这将影响客户机虚拟机。<sysinfo> 的每个子元素都命名 SMBIOS 块，在这些元素中，可以是描述块内字段的条目元素列表。可以识别以下块和条目：

- <BIOS> - 这是 SMBIOS 块 0，其条目名取自 供应商的、版本、日期和时间。

- <System> - 这是 SMBIOS 块 1，其条目名取自 manufacturer、产品、版本、serial、uuid、s ku，以及 系列产品。如果一个 uuid 条目与顶层 uuid 元素一起提供，则两个值必须匹配。

23.4. CPU 分配

图 23.6. CPU 分配

```
<domain>
  ...
  <vcpu placement='static' cpuset="1-4,^3,6" current="1">2</vcpu>
  ...
</domain>
```

<vcpu> 元素定义为客户端虚拟机操作系统分配的的最大虚拟 CPU 数量，该 CPU 必须在 1 之间以及虚拟机监控程序支持的最大值。此元素可以包含可选的 cpuset 属性，它是域进程和虚拟 CPU 默认可以固定到的物理 CPU 的逗号分隔列表。

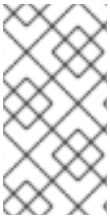
请注意，可以使用 cputune 属性单独指定域进程和虚拟 CPU 的固定策略。如果在 <cputune> 中指定 emulatorpin 属性，则 <vcpu> 指定的cpuset 将会被忽略。

同样，为 vcpupin 设置了值的虚拟 CPU 会导致忽略 cpuset 设置。对于没有指定 vcpupin 的虚拟 CPU，它将固定到 cpuset 指定的物理 CPU。cpuset 列表中的每个元素都是单个 CPU 号、一个 CPU 数字范围，或小心(^)，后面跟着要排除在之前范围内的 CPU 编号。当前 的属性可用于指定是否应启用的最大虚拟 CPU 数。

放置 可选属性可用于指示域进程的 CPU 放置模式。放置 的值可以设置为以下之一：

- static - 将 vCPU 固定到 cpuset 属性定义的物理 CPU。如果未定义 cpuset，域进程将固定到所有可用的物理 CPU。

- auto - 表示域进程将从 Query numad 中固定到公告节点集，如果指定了 attribute cpuset 的值，则忽略它。



注意

如果 cpuset 属性与 放置 一同使用，则 placement 的值默认为 <numatune> 元素的值（如果使用时）或 静态。

23.5. CPU 调整

图 23.7. CPU 调优

```
<domain>
  ...
  <cputune>
    <vcpupin vcpu="0" cpuset="1-4,^2"/>
    <vcpupin vcpu="1" cpuset="0,1"/>
    <vcpupin vcpu="2" cpuset="2,3"/>
    <vcpupin vcpu="3" cpuset="0,4"/>
    <emulatorpin cpuset="1-3"/>
    <shares>2048</shares>
    <period>1000000</period>
    <quota>-1</quota>
    <emulator_period>1000000</emulator_period>
    <emulator_quota>-1</emulator_quota>
  </cputune>
  ...
</domain>
```

虽然所有都是可选的，但域 XML 的这个部分的组件如下：

表 23.4. CPU 调整元素

元素	描述
<cputune>	提供有关域的 CPU 可调参数的详细信息。这是可选的。
<vcpupin>	指定域 vCPU 将固定到的主机物理 CPU。如果省略它，且未指定 **<vcpu>** 元素的 **cpuset** 属性，则 vCPU 会默认固定到所有物理 CPU。它包含两个必要的属性： **<vcpu>** 属性指定 **id**，而 **cpuset** 属性与 **<vcpu>** 元素中的 **cpuset** 属性相同。

元素	描述
<emulatorpin>	指定哪些主机物理机器 CPU 将固定为"emulator"（一个不是包含 **<vcpu>**的域的子集）。如果省略，且 **<vcpu>** 元素中的 **cpuset** 属性没有指定，则"emulator"默认固定到所有物理 CPU。它包含一个 required **cpuset** 属性，指定要将哪些物理 CPU 固定到。如果 **<vcpu>** 元素中的 **placement** 属性被设置为 **auto**，则不允许模拟固定。
<共享>	指定域的比例加权共享。如果省略此项，则默认为操作系统提供的默认值。如果没有值的单位，则会根据其它客户机虚拟机的设置来计算它。例如，配置了 **<共享>** 值 2048 的客户机虚拟机将获得两倍的 CPU 时间，作为配置了 **<共享>** 值 1024 的客户机虚拟机。
<周期>	以微秒为单位指定强制间隔。通过使用 **<句点>**，每个域的 vCPU 将不允许消耗超过其运行时所分配的配额。这个值应该在以下范围内：**1000-1000000**.值为 **0** 的 **<period>** 表示没有值。
<quota>	指定微秒允许的最大带宽。具有 **<配额>** 的域表示域具有无限带宽，这意味着它不控制带宽。该值应在以下范围内：**1000 - 18446744073709551** 或小于 **0。**值为 **0 的配额** 表示没有值。您可以使用此功能来确保所有 vCPU 以相同的速度运行。
<emulator_period>	以微秒为单位指定强制间隔。在 **<emulator_period>** 中，域的仿真线程（不包括 vCPU）不允许消耗超过运行时的 **<emulator_ worthquota>**。**<emulator_period>** 值应该在以下范围内：**1000 - 1000000**.值为 **0** 的 **<emulator_period>** 表示没有值。
<emulator_quota>	指定域的仿真程序线程（不包括 vCPU）的最大允许带宽的最大带宽（以微秒为单位）。带有 **<emulator_quota>** 作为负值的域表示域具有仿真程序线程的无限带宽（除 vCPU 外），这意味着它不控制带宽。该值应位于以下范围内：**1000 - 18446744073709551** 或小于 **0。**值为 **0** 的 **<emulator_quota>** 表示没有值。

23.6. 内存备份

内存后备允许管理程序正确管理客户机虚拟机中的大页。

图 23.8. 内存备份

```
<domain>
  ...
  <memoryBacking>
    <hugepages>
   <page size="1" unit="G" nodeset="0-3,5"/>
    <page size="2" unit="M" nodeset="4"/>
  </hugepages>
  <nosharepages/>
  <locked/>
  </memoryBacking>
  ...
</domain>
```

*有关 **memoryBacking** 元素的详情，请查看 libvirt 上游文档。*

23.7. 内存调整

图 23.9. 内存调整

```
<domain>
  ...
  <memtune>
    <hard_limit unit='G'>1</hard_limit>
    <soft_limit unit='M'>128</soft_limit>
    <swap_hard_limit unit='G'>2</swap_hard_limit>
    <min_guarantee unit='bytes'>67108864</min_guarantee>
  </memtune>
  ...
</domain>
```

*虽然 **<memtune>** 是可选的，但域 XML 的这个部分的组件如下：*

表 23.5. 内存调整元素

元素	描述
<memtune>	提供有关域内存可调参数的详细信息。如果省略此项，则默认为操作系统提供的默认值。由于参数全部应用于进程，在设置限制时，通过将客户机虚拟机 RAM 添加到客户机虚拟机视频 RAM 来确定值，从而允许一些内存开销。对于每个可调整，可以指定在输入中哪个单元数，其值与 **<内存>** 的值相同。为了向后兼容，输出始终处于 kibibytes(KiB)中。

元素	描述
<hard_limit>	客户机虚拟机可以使用的最大内存。这个值以 kibibytes（**1024** 字节块）表示。
<soft_limit>	在内存争用期间强制实施的内存限值。这个值以 kibibytes（1024 字节块）表示。
<swap_hard_limit>	客户机虚拟机可使用的最大内存加上 swap。这个值以 kibibytes（1024 字节块）表示。这必须大于 **<hard_limit>** 值。
<min_guarantee>	保证客户机虚拟机的最小内存分配量。这个值以 kibibytes（1024 字节块）表示。

23.8. 内存分配

如果客户机虚拟机崩溃，可选属性 dumpCore 可用于控制客户机虚拟机内存是否应该包含在生成的内核转储中(dumpCore='on')或未包括(dumpCore='off')。请注意，默认设置 在 上，因此，除非将 参数设置为 off，否则客户机虚拟机内存将包含在 core dumpfile 中。

<maxMemory> 元素决定客户机的最大运行时内存分配。插槽 属性指定可用于向客户机中添加内存的插槽数。

<memory> 元素指定在引导时为客户机分配的最大内存。这也可以使用 NUMA 单元大小配置来设置，也可以通过将内存热插到 maxMemory 指定的限制来增加。

<currentMemory> 元素决定客户机虚拟机的实际内存分配。这个值可能小于最大分配量（由 <内存>设定），以根据需要允许客户机虚拟机内存到 balloon。如果省略，则默认为与 <memory> 元素相同的值。unit 属性的行为与内存相同。

图 23.10. 内存单元

```
<domain>
  <maxMemory slots='16' unit='KiB'>1524288</maxMemory>
  <memory unit='KiB' dumpCore='off'>524288</memory>
  <!-- changes the memory unit to KiB and does not allow the guest virtual machine's memory
to be included in the generated core dumpfile -->
  <currentMemory unit='KiB'>524288</currentMemory>
  <!-- makes the current memory unit 524288 KiB -->
  ...
</domain>
```

23.9. NUMA 节点调优

使用 virsh edit 完成 NUMA 节点调整后，以下域 XML 参数会受到影响：

图 23.11. NUMA 节点调整

```
<domain>
  ...
  <numatune>
    <memory mode="strict" nodeset="1-4,^3"/>
  </numatune>
  ...
</domain>
```

虽然所有都是可选的，但域 XML 的这个部分的组件如下：

表 23.6. NUMA 节点调整元素

元素	描述
<numatune>	介绍如何通过控制域进程的 NUMA 策略来调整 NUMA 主机物理机器的性能。
<memory>	指定如何在 NUMA 主机物理机器上为域进程分配内存。它包含几个可选属性。**mode** 属性可以设置为 **interleave**、**strict** 或 **preferred**。如果没有赋予值，则默认为 **strict**。**nodeset** 属性使用与 **<vcpu>** 元素的 **cpuset** 属性相同的语法来指定 NUMA 节点。属性 **放置** 可用于指示域进程的内存放置模式。其值可以是 **static**，也可以是 **自动**。如果指定了 **<nodeset>** 属性，则默认为 **<vcpu> <的放置>**，或者 为**静态**。**auto** 则表示域进程只会从查询 numad 返回的公告 nodeset 分配内存，如果指定了 numad，则 **nodeset** 属性的值将会被忽略。如果 **vcpu** 中的 **<placement>** 属性设置为 **auto**，并且未指定 **<numatune>** 属性，则将隐式地添加带有 **<放置> 自动** 和 **严格** 模式的默认 **<numatune>** 属性。

23.10. 块 I/O 调整

图 23.12. 块 I/O 调整

```
<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>
```

虽然所有都是可选的，但域 XML 的这个部分的组件如下：

表 23.7. 块 I/O 调优元素

元素	描述
<blkiotune>	此可选元素提供了为域调优 **blkio** cgroup 可调参数的功能。如果省略此项，则默认为操作系统提供的默认值。
<weight>	此可选 weight 元素是客户机虚拟机的整体 I/O 权重。该值应介于 100 - 1000 范围内。
<device>	域可以有多个 **<设备>** 元素，它们可进一步调整域所使用的每个主机物理机器块设备的权重。请注意，多个客户机虚拟机磁盘可以共享单个主机物理机器块设备。另外，由于它们由同一主机物理机器文件系统中的文件提供支持，因此此调整参数位于全局域级别，而不是与每个客户机虚拟机磁盘设备关联（将其应用于单个 **<磁盘的> <iotune>** 元素）。每个 device 元素有两个强制子元素，用于描述设备的绝对路径的路径，**<权重>** 为那个设备的相对权重，其可接受的 100 到 1000 范围。**<>**

23.11. 资源分区

管理程序可能将虚拟机放置在资源分区中，可能带有嵌套所提到的分区。<resource> 元素将与资源分区相关的配置分组在一起。它目前支持子元素分区，其内容定义了要放置域的资源分区的路径。如果没有列出分区，则域将放置在默认分区中。必须先创建分区，然后才能启动 guest 虚拟机。默认情况下，只有特定于管理程序的默认分区。

图 23.13. 资源分区

```
<resource>
  <partition>/virtualmachines/production</partition>
</resource>
```

KVM 和 LXC 驱动程序目前支持资源分区，它将分区路径映射到所有挂载的控制器中的 cgroups 目录。

23.12. CPU 型号和拓扑

这部分论述了 CPU 型号的要求。请注意，每个虚拟机监控程序都有自己的策略，用于客户机默认看到的 CPU 功能。KVM 向客户机呈现的 CPU 功能集合取决于客户机虚拟机配置中选择的 CPU 模型。qemu32 和 qemu64 是基本的 CPU 模型，但也有其他模型（带有附加功能）可用。每个模型及其拓扑的使用域 XML 中的以下元素指定：

图 23.14. CPU 型号和拓扑示例 1

```
<cpu match='exact'>
    <model fallback='allow'>core2duo</model>
    <vendor>Intel</vendor>
    <topology sockets='1' cores='2' threads='1'/>
    <feature policy='disable' name='lahf_lm'/>
</cpu>
```

图 23.15. CPU 型号和拓扑示例 2

```
<cpu mode='host-model'>
  <model fallback='forbid'/>
  <topology sockets='1' cores='2' threads='1'/>
</cpu>
```

图 23.16. CPU 模型和拓扑示例 3

```
<cpu mode='host-passthrough'/>
```

如果没有限制在 CPU 模型或其功能上有限制，则可使用以下更简单的 <cpu> 元素：

图 23.17. CPU 型号和拓扑示例 4

```
<cpu>
  <topology sockets='1' cores='2' threads='1'/>
</cpu>
```

图 23.18. PPC64/PSeries CPU 型号示例

```
<cpu mode='custom'>
  <model>POWER8</model>
</cpu>
```

图 23.19. aarch64/virt CPU 型号示例

```
<cpu mode='host-passthrough'/>
```

域 XML 的这一部分的组件如下：

表 23.8. CPU 模型和拓扑元素

元素	描述
<cpu>	这是描述客户机虚拟机 CPU 要求的主要容器。
<匹配>	指定如何为客户机虚拟机提供虚拟 CPU 如何符合这些要求。如果拓扑是 **<cpu>** 中的唯一元素，则可以省略 **match** 属性。**match** 属性的可能值有： • **最小值** - 指定 CPU 模型和功能描述了最低请求的 CPU。 • **exact** - 提供给客户机虚拟机的虚拟 CPU 将与规格完全匹配。 • **strict** - 除非主机物理机器 CPU 完全匹配，否则不会创建客户机虚拟机。 请注意，可以省略 **match** 属性，并将默认为 **完全匹配**。

元素	描述
<模式>	此可选属性可用于方便配置客户机虚拟机 CPU，尽可能地接近主机物理机器 CPU。**mode** 属性的可能值有： • **Custom** - 描述如何将 CPU 呈现给客户机虚拟机。如果没有指定 **mode** 属性，这是默认设置。这个模式使得持久的 guest 虚拟机将看到同一硬件，无论 guest 虚拟机启动的主机是哪一个。 • **host-model** - 将主机物理机器 CPU 定义从功能 XML 复制到域 XML 中的快捷方式。因为在启动域前复制 CPU 定义，不同主机物理机器上可以使用相同的 XML，同时仍提供最佳的客户机虚拟机 CPU 每台主机物理机器支持。**match** 属性和任何功能元素不能在此模式中使用。如需更多信息，请参阅 libvirt 上游网站。 • 通过这种模式通过 **host-passthrough**，对客户机虚拟机可见的 CPU 与主机物理机器 CPU 完全相同，包括在 libvirt 中导致错误的元素。显而易见，此模式是 guest 虚拟机环境无法在不同的硬件上重现，因此建议使用这种模式非常谨慎。此模式中不允许使用 **模型 和功能** 元素。
<model>	指定客户机虚拟机请求的 CPU 型号。在 libvirt 的数据目录中安装的 **cpu_map.xml** 文件可以找到可用 CPU 模型列表及其定义。如果虚拟机监控程序无法使用确切的 CPU 模型，则 libvirt 会自动回退到虚拟机监控程序支持的最接近的模型，同时保持 CPU 功能列表。可选 **fallback** 属性可用于禁止此行为，在这种情况下，尝试启动请求不支持的 CPU 模型的域将失败。fallback 属性支持的值有：**allow**（默认值）和 **forbid**。可选的 **vendor_id** 属性可用于设置客户机虚拟机可见的供应商 ID。长度必须为 12 个字符。如果没有设置，则使用主机物理机器的供应商 iID。典型的可能值为 **AuthenticAMD** 和 **GenuineIntel**。
<vendor>	指定客户机虚拟机请求的 CPU 供应商。如果缺少此元素，无论其供应商如何，客户机虚拟机在 CPU 匹配的 CPU 上运行。支持的厂商列表可在 **cpu_map.xml** 中找到。
<topology>	指定提供给客户端虚拟机的虚拟 CPU 的请求拓扑。为插槽、内核和线程分配三个非零值：CPU 插槽总数、每个插槽的内核数和每个内核的线程数量。

元素	描述
<功能>	可以包含零或更多元素，用于微调由所选 CPU 模型提供的功能。已知功能名称列表可在 **cpu_map.xml** 文件中找到。每个功能元素的含义取决于其策略属性，必须设置为以下值之一： • **force** - 强制支持虚拟机，无论主机物理机器 CPU 是否实际支持。 • **require** - 指明客户机虚拟机创建将失败，除非主机物理机器 CPU 支持该功能。这是默认设置， • **可选** - 虚拟 CPU 支持此功能，但只有主机物理机器 CPU 支持该功能。 • **disable** - 虚拟 CPU 不支持此功能。 • **forbid** - 如果主机物理机器 CPU 支持该功能，则客户机虚拟机创建将失败。

23.12.1. 更改指定 CPU 的 Feature Set

虽然 CPU 模型具有一项固有功能集，但可以根据需要对功能允许或禁止各个功能组件，从而实现 CPU 更为个性化的配置。

过程 23.1. 启用和禁用 CPU 功能

1. *要开始，请关闭 guest 虚拟机。*
2. *通过运行 virsh edit [domain] 命令打开 guest 虚拟机的配置文件。*
3. *更改功能或 <模型中> 的参数，使其包含属性值 "allow" 以强制允许该功能，或 "forbid" 以拒绝对这个功能的支持。 <>*

图 23.20. 启用或禁用 CPU 功能示例

```
<!-- original feature set -->
<cpu mode='host-model'>
  <model fallback='allow'/>
  <topology sockets='1' cores='2' threads='1'/>
</cpu>

<!--changed feature set-->
<cpu mode='host-model'>
  <model fallback='forbid'/>
  <topology sockets='1' cores='2' threads='1'/>
</cpu>
```

图 23.21. 启用或禁用 CPU 功能示例 2

```
<!--original feature set-->
<cpu match='exact'>
   <model fallback='allow'>core2duo</model>
   <vendor>Intel</vendor>
   <topology sockets='1' cores='2' threads='1'/>
   <feature policy='disable' name='lahf_lm'/>
</cpu>

<!--changed feature set-->
<cpu match='exact'>
   <model fallback='allow'>core2duo</model>
   <vendor>Intel</vendor>
   <topology sockets='1' cores='2' threads='1'/>
   <feature policy='enable' name='lahf_lm'/>
 </cpu>
```

4. 完成更改后，保存配置文件并启动客户机虚拟机。

23.12.2. 虚拟机 NUMA 拓扑

可以使用域 XML 中的 <numa> 元素指定客户机虚拟机 NUMA 拓扑：

图 23.22. 客户机虚拟机 NUMA 拓扑

```
<cpu>
  <numa>
    <cell cpus='0-3' memory='512000'/>
    <cell cpus='4-7' memory='512000'/>
  </numa>
</cpu>
...
```

每个单元元素都指定了一个 NUMA 单元或 NUMA 节点。cpus 指定作为节点一部分的 CPU 或范围。内存 以 kibibytes（1024 字节块）指定节点内存。每个单元格或节点都会被分配一个 cellid 或 nodeid 来增加从 0 开始的顺序。

23.13. 事件配置

使用域 XML 的以下部分可以覆盖各种事件的默认操作：

图 23.23. 事件配置

```
<on_poweroff>destroy</on_poweroff>
<on_reboot>restart</on_reboot>
<on_crash>restart</on_crash>
<on_lockfailure>poweroff</on_lockfailure>
```

以下元素集合允许在虚拟机操作系统触发生命周期操作时指定操作。常见用例是强迫在进行初始操作系统安装时将重新启动视为关闭。这允许为首次安装后引导时重新配置虚拟机。

域 XML 的这一部分的组件如下：

表 23.9. 事件配置元素

状态	描述

状态	描述
<on_poweroff>	指定在客户机虚拟机请求关闭时执行的操作。可以有四个参数： • **destroy** - 此操作会完全终止域，并释放所有资源。 • **restart** - 此操作会完全终止域，并使用同一配置重新启动它。 • **preserve** - 此操作会完全终止域，但保留其资源以便未来分析。 • **rename-restart** - 此操作完全终止域，然后使用新名称重新启动域。
<on_reboot>	指定客户机虚拟机请求重新引导时执行的操作。可以有四个参数： • **destroy** - 此操作会完全终止域，并释放所有资源。 • **restart** - 此操作会完全终止域，并使用同一配置重新启动它。 • **preserve** - 此操作会完全终止域，但保留其资源以便未来分析。 • **rename-restart** - 此操作完全终止域，然后使用新名称重新启动域。
<on_crash>	指定要在客户机虚拟机崩溃时执行的操作。另外，它支持这些额外操作： • **coredump-destroy** - 崩溃的域内核被转储，域将完全终止，所有资源都会释放。 • **coredump-restart** - 崩溃的域的内核会被转储，且域会使用相同的配置设置重启。 可以有四个参数： • **destroy** - 此操作会完全终止域，并释放所有资源。 • **restart** - 此操作会完全终止域，并使用同一配置重新启动它。 • **preserve** - 此操作会完全终止域，但保留其资源以便未来分析。 • **rename-restart** - 此操作完全终止域，然后使用新名称重新启动域。

状态	描述
<on_lockfailure>	指定锁定管理器丢失资源锁定时要执行的操作。以下的操作可由 libvirt 识别，但不需要单独锁定管理器支持它们。如果没有指定操作，每个锁定管理器将采取其默认操作。以下参数可以： • **poweroff** - 强制关闭域。 • **restart** - 重启域以重新退出其锁定。 • **暂停** - 暂停域，以便在锁定问题解决时手动恢复。 • **ignore** - 使域正常运行（如果没有任何发生）。

23.14. HYPERVISOR 功能

Hypervisor 可能允许启用某些 CPU 或机器功能(state='on')或禁用(state='off')。

图 23.24. 虚拟机监控程序功能

```
...
<features>
  <pae/>
  <acpi/>
  <apic/>
  <hap/>
  <privnet/>
  <hyperv>
    <relaxed state='on'/>
  </hyperv>
</features>
...
```

如果没有指定 <状态>，则所有功能都列在 <features> 元素中。可通过调用 capabilities XML 发现可用功能，但为完全虚拟化域的通用集合是：

表 23.10. 管理程序功能元素

状态	描述
<pae>	物理地址扩展模式允许 32 位客户机虚拟机处理超过 4 GB 内存。

状态	描述
<acpi>	对于电源管理非常有用。例如，在 KVM 客户机虚拟机中，安全关闭需要它。
<apic>	允许使用可编程的 IRQ 管理。此元素有一个可选属性 **eoi**，其值为 **on** 和 **off**，这会设置客户机虚拟机的 EOI（中断终止）的可用性。
<hap>	如果硬件中有可用，则启用硬件辅助分页的使用。

23.15. TIMEKEEPING

客户机虚拟机时钟通常从主机物理机器时钟初始化。大多数操作系统都预期硬件时钟保存在 UTC 中，这是默认设置。

准确的客户机虚拟机计时是虚拟化平台的关键挑战。不同的管理程序试图以各种方式处理计时的问题。libvirt 使用域 XML 中的 <clock> 和 <timer> 元素提供独立于管理程序的配置设置。可以使用 virsh edit 命令编辑域 XML。详情请查看 第 20.22 节 “编辑虚拟机 XML 配置设置”。

图 23.25. timekeeping

```
...
<clock offset='localtime'>
  <timer name='rtc' tickpolicy='catchup' track='guest'>
    <catchup threshold='123' slew='120' limit='10000'/>
  </timer>
  <timer name='pit' tickpolicy='delay'/>
</clock>
...
```

域 XML 的这一部分的组件如下：

表 23.11. 计时元素

状态	描述

状态	描述
<clock>	**<clock>** 元素用于确定如何将客户机虚拟机时钟与主机物理机器时钟同步。**offset** 属性使用四个可能值，允许对将客户机虚拟机时钟与主机物理机器同步的方式进行精细控制。请注意，管理程序不需要在所有时间源中支持所有策略 • UTC - 引导时将时钟同步到 UTC。**utc** 模式可转换为 **变量** 模式，该模式可使用 adjustment 属性来控制。**如果 重置了 值，则不会完成转换。数值强制将值转换为变量 模式，作为初始调整。默认调整是特定于虚拟机监控程序的。** • **localtime - 引导时，将客户机虚拟机时钟与主机物理计算机配置的时区同步。adjustment 属性的行为与 utc 模式 相同。** • **timezone - 将客户机虚拟机时钟与请求的时区同步。** • 变量 **- 根据基础属性，**为 **guest 虚拟机时钟提供相对于 UTC 或 本地时间应用的任意偏移量。使用 adjust 属性来指定与 UTC 相关的增量（或 本地时间）的 delta。客户机虚拟机可以自由地调整 RTC，并期望在下次重启时它会被遵守。这与 utc 和 localtime 模式不同（带有可选的属性 adjustment='reset'），其中 RTC 在每次重启时都会丢失。此外，base 属性也可以是 utc （默认）或 localtime。clock 元素可以有零个或多个 <计时器> 元素。**
<timer>	请参阅备注
<存在>	指定特定计时器可用于客户机虚拟机。可以设置为 **yes** 或 **no**。

注意

<时钟> 元素可以有零个或多个 <计时器> 元素作为子项。<timer> 元素指定用于客户机虚拟机时钟同步的时间源。

在每个 <计时器> 元素中，需要 名称，所有其他属性都是可选的：

- **Name - 选择修改哪个 计时器。可接受以下值： kvmclock、pit 或 rtc。**
- **track - 指定计时器跟踪。可接受以下值： boot、guest 或 wall.track 仅对 name="rtc" 有效。**
- **tickpolicy - 确定在在向客户机虚拟机注入 tick 时会发生什么。可以分配以下值：**
  - **delay - 继续以正常率交付行程。由于相关空点，客户机虚拟机时间将延迟。**
  - **捕获 - 按更高的速率提供电话，以便跟上空缺的支票。在捕获完成后，客户机虚拟机时间不会被显示。另外，可以有三个可选属性，每个正整数：threshold、slew 和 limit。**
  - **merge - 将错过的支票放入一个空行并注入它们。根据合并的方式，客户机虚拟机时间可能会延迟。**
  - **discard - 放弃错过的提示，并在其默认间隔设置中继续使用将来的注入。客户端虚拟机时间可能会延迟，除非有明确的声明可用于处理丢失的 ticks。**

注意

默认情况下，utc 被设置为虚拟机中的时钟偏移。但是，如果客户机虚拟机时钟使用 localtime 值运行，则需要将时钟偏移更改为不同的值，以便让客户机虚拟机时钟与主机物理机器时钟同步。

例 23.1. 始终与 UTC 同步

```
<clock offset="utc" />
```

例 23.2. 始终与主机物理计算机时区同步

```
<clock offset="localtime" />
```

例 23.3. 与任意时区同步

```
<clock offset="timezone" timezone="Europe/Paris" />
```

例 23.4. 与 UTC 同步 + 任意偏移

```
<clock offset="variable" adjustment="123456" />
```

23.16. 计时器元素属性

name 元素包含要使用的时间源的名称。它可以具有以下值：

表 23.12. 名称属性值

订阅价值	描述
pit	可编程的间隔时间 - 一个带有定期中断的计时器。使用此属性时，**tickpolicy** 延迟会成为默认设置。
rtc	实时时钟 - 不断运行带有定期中断的计时器。此属性支持 **tickpolicy catchup** 子元素。
kvmclock	KVM clock - KVM 客户机虚拟机的推荐时钟源。KVM pvclock 或 kvm-clock 允许客户机虚拟机读取主机物理机器的 wall clock 时间。

track 属性指定计时器跟踪的内容，并且仅对 rtc 的名称 值有效。

表 23.13. 跟踪属性值

订阅价值	描述
boot	对应于旧的 *主机物理计算机* 选项，这是不受支持的跟踪选项。

订阅价值	描述
guest	RTC 始终跟踪客户机虚拟机时间。
Wall	RTC 始终跟踪主机时间。

tickpolicy 属性和值指示用于将 ticks 传递到客户机虚拟机的策略。

表 23.14. tickpolicy 属性值

订阅价值	描述
delay	继续以正常率交付（ticks 为延迟）。
catchup	以更高的速度实现.
merge	将合并成一个 tick。
discard	所有错过的提示都将被丢弃。

present 属性用于覆盖对客户机虚拟机可见的默认计时器集。present 属性可以采用以下值：

表 23.15. 介绍属性值

订阅价值	描述
是	强制此计时器对 guest 虚拟机可见。
否	强制此计时器对 guest 虚拟机不可见。

23.17. DEVICES

这一组 XML 元素都用于描述提供给客户机虚拟机域的设备。以下所有设备都声明为主设备元素的子项。<>

支持以下虚拟设备：

- *virtio-scsi-pci - PCI 总线存储设备*

- *virtio-blk-pci - PCI 总线存储设备*
- *virtio-net-pci - PCI 总线网络设备也称为 virtio-net*
- *virtio-serial-pci - PCI 总线输入设备*
- *virtio-balloon-pci - PCI 总线内存 balloon 设备*
- *virtio-rng-pci - PCI 总线虚拟随机数生成器设备*

重要

如果创建 virtio 设备，其中将向量数设定为高于 32 的值，则设备的行为就像在 Red Hat Enterprise Linux 6 中被设置为零值而在 Enterprise Linux 7 中一样。如果平台中的任何 virtio 设备中的向量数设置为 33 或更高版本，则生成的向量设置不匹配会导致迁移错误。因此，建议您不要将向量值设置为大于 32。除 virtio-balloon-pci 和 virtio-rng-pci 以外的所有 virtio 设备都接受 向量 参数。

图 23.26. 设备 - 子元素

```
...
<devices>
  <emulator>/usr/libexec/qemu-kvm</emulator>
</devices>
...
```

仿真器元素的内容指定到设备模型 <模拟器> 二进制文件的完全限定路径。capabilities XML 指定要用于每个特定域类型或架构组合的推荐默认模拟器。

23.17.1. 硬盘驱动器、软盘和 CD-ROMs

域 XML 的这一部分指定了类似磁盘的任何设备，包括 <磁盘> 元素中指定的任何软盘、硬盘、CD-ROM 或泛虚拟化驱动程序。

图 23.27. devices - 硬盘、软盘、CD-ROM 示例

```
<disk type='network'>
  <driver name="qemu" type="raw" io="threads" ioeventfd="on" event_idx="off"/>
  <source protocol="sheepdog" name="image_name">
    <host name="hostname" port="7000"/>
  </source>
  <target dev="hdb" bus="ide"/>
  <boot order='1'/>
  <transient/>
  <address type='drive' controller='0' bus='1' unit='0'/>
</disk>
```

图 23.28. devices - 硬盘、软盘、CD-ROM 示例 2

```
<disk type='network'>
  <driver name="qemu" type="raw"/>
  <source protocol="rbd" name="image_name2">
    <host name="hostname" port="7000"/>
  </source>
  <target dev="hdd" bus="ide"/>
  <auth username='myuser'>
    <secret type='ceph' usage='mypassid'/>
  </auth>
</disk>
```

图 23.29. 设备 - 硬盘、软盘、CD-ROM 示例 3

```
<disk type='block' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <target dev='hdc' bus='ide' tray='open'/>
  <readonly/>
</disk>
<disk type='network' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <source protocol="http" name="url_path">
    <host name="hostname" port="80"/>
  </source>
  <target dev='hdc' bus='ide' tray='open'/>
  <readonly/>
</disk>
```

图 23.30. 设备 - 硬盘、软盘、CD-ROM 示例 4

```
<disk type='network' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <source protocol="https" name="url_path">
    <host name="hostname" port="443"/>
  </source>
  <target dev='hdc' bus='ide' tray='open'/>
  <readonly/>
</disk>
<disk type='network' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <source protocol="ftp" name="url_path">
    <host name="hostname" port="21"/>
  </source>
  <target dev='hdc' bus='ide' tray='open'/>
  <readonly/>
</disk>
```

图 23.31. 设备 - 硬盘、软盘、CD-ROM 示例 5

```
<disk type='network' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <source protocol="ftps" name="url_path">
    <host name="hostname" port="990"/>
  </source>
  <target dev='hdc' bus='ide' tray='open'/>
  <readonly/>
</disk>
<disk type='network' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <source protocol="tftp" name="url_path">
    <host name="hostname" port="69"/>
  </source>
  <target dev='hdc' bus='ide' tray='open'/>
  <readonly/>
</disk>
<disk type='block' device='lun'>
  <driver name='qemu' type='raw'/>
  <source dev='/dev/sda'/>
  <target dev='sda' bus='scsi'/>
  <address type='drive' controller='0' bus='0' target='3' unit='0'/>
</disk>
```

图 23.32. devices - 硬盘、软盘、CD-ROM 示例 6

```
<disk type='block' device='disk'>
  <driver name='qemu' type='raw'/>
  <source dev='/dev/sda'/>
  <geometry cyls='16383' heads='16' secs='63' trans='lba'/>
  <blockio logical_block_size='512' physical_block_size='4096'/>
  <target dev='hda' bus='ide'/>
</disk>
<disk type='volume' device='disk'>
  <driver name='qemu' type='raw'/>
  <source pool='blk-pool0' volume='blk-pool0-vol0'/>
  <target dev='hda' bus='ide'/>
</disk>
<disk type='network' device='disk'>
  <driver name='qemu' type='raw'/>
  <source protocol='iscsi' name='iqn.2013-07.com.example:iscsi-nopool/2'>
    <host name='example.com' port='3260'/>
  </source>
  <auth username='myuser'>
    <secret type='chap' usage='libvirtiscsi'/>
  </auth>
  <target dev='vda' bus='virtio'/>
</disk>
```

图 23.33. devices - 硬盘、软盘、CD-ROM 示例 7

```
<disk type='network' device='lun'>
  <driver name='qemu' type='raw'/>
  <source protocol='iscsi' name='iqn.2013-07.com.example:iscsi-nopool/1'>
   iqn.2013-07.com.example:iscsi-pool
    <host name='example.com' port='3260'/>
  </source>
  <auth username='myuser'>
    <secret type='chap' usage='libvirtiscsi'/>
  </auth>
  <target dev='sda' bus='scsi'/>
</disk>
<disk type='volume' device='disk'>
  <driver name='qemu' type='raw'/>
  <source pool='iscsi-pool' volume='unit:0:0:1' mode='host'/>
  <auth username='myuser'>
    <secret type='chap' usage='libvirtiscsi'/>
  </auth>
  <target dev='vda' bus='virtio'/>
</disk>
```

图 23.34. devices - 硬盘、软盘、CD-ROM 示例 8

```
...
  <disk type='volume' device='disk'>
    <driver name='qemu' type='raw'/>
    <source pool='iscsi-pool' volume='unit:0:0:2' mode='direct'/>
    <auth username='myuser'>
      <secret type='chap' usage='libvirtiscsi'/>
    </auth>
    <target dev='vda' bus='virtio'/>
  </disk>
   <disk type='file' device='disk'>
    <driver name='qemu' type='raw' cache='none'/>
    <source file='/tmp/test.img' startupPolicy='optional'/>
    <target dev='sdb' bus='scsi'/>
    <readonly/>
  </disk>
  <disk type='file' device='disk'>
    <driver name='qemu' type='raw' discard='unmap'/>
    <source file='/var/lib/libvirt/images/discard1.img'/>
    <target dev='vdb' bus='virtio'/>
    <alias name='virtio-disk1'/>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x09' function='0x0'/>
  </disk>
</devices>
...
```

23.17.1.1. 磁盘元素

<disk> 元素是描述磁盘的主容器。属性 类型 可与 <disk> 元素一起使用。允许以下类型：

- **file**
- **block**
- **dir**
- **network**

如需更多信息，请参阅 libvirt 上游页面。

23.17.1.2. 源元素

代表磁盘源。磁盘源取决于磁盘类型属性，如下所示：

- <file> - file 属性指定到磁盘所在的文件的完全限定路径。
- <block> - dev 属性指定充当磁盘的主机设备的完全限定路径。
- <dir> - dir 属性指定用作磁盘的目录的完全限定路径。
- <network> - protocol 属性指定用于访问所请求镜像的协议。可能的值有：nbd、isci、rbd、sheepdog 和 gluster。
  - 如果 协议 属性是 rbd、sheepdog 或 gluster，则 其他属性 是必须的。此属性指定将使用哪些卷和镜像。
  - 如果 协议 属性是 nbd，则 name 属性是可选的。
  - 如果 protocol 属性 为，则 name 属性可以包含逻辑单元号，用斜杠将其与目标的名称分开。例如：iqn.2013-07.com.example:iscsi-pool/1。如果未指定，则默认 LUN 为零。
- <卷> - 底层磁盘源由 池和 卷 属性表示。
  - <pool> - 磁盘源所在的存储池（由 libvirt管理）的名称。
  - <卷> - 用作磁盘源的存储卷（由 libvirt管理）的名称。

  volume 属性的值是 virsh vol-list [pool-name]的 Name 列的输出。

当磁盘类型是 网络 时，源 可能具有零个或更多 主机 子元素，用于指定要连接的主机物理计算机，包括： type='dir' 和 type='network'。对于代表 CD-ROM 或软盘（ device 属性）的文件磁盘类型，可以在无法访问源文件时为磁盘定义操作的策略。这可以通过将 startupPolicy 属性设置为以下值之一来实现：

- 如果因任何原因而丢失，强制 会导致失败。这是默认的设置。
- 如果在引导时缺少，则必需会导致失败，如果迁移、恢复或恢复缺少，则会导致失败。
- 如果开始尝试中缺少 可选 操作，则可选 drops。

23.17.1.3. mirror 元素

如果虚拟机监控程序已启动了一个 BlockCopy 操作，其中属性文件中的镜像位置最终具有与源相同的内容，并且采用属性格式的文件格式（这可能与源的格式不同）。<>如果属性就绪，则代表磁盘已已知为 pivot；否则，磁盘可能仍会复制。现在，这个元素只在输出中有效，在输入时会忽略它。

23.17.1.4. target 元素

<target> 元素控制磁盘被公开给客户端虚拟机操作系统的总线或设备。dev 属性指示逻辑设备名称。指定的实际设备名称不能保证映射到客户端虚拟机操作系统中的设备名称。可选的 bus 属性指定要模拟的磁盘设备类型；可能的值特定于驱动程序，典型的值是 ide、scsi、virtio 、kvm、usb 或 sata。如果省略，总线类型会从设备名称的样式中推断出来。例如：名为 "sda" 的设备通常使用 SCSI 总线导出。可选的属性 tray 表示可移动磁盘（如 CD-ROM 或 Floppy 磁盘）的托盘状态，其中值可以是 打开或关闭。默认设置 关闭。

23.17.1.5. iotune 元素

可选的 <iotune> 元素提供额外的每个设备 I/O 调优功能，但每个设备的值可能会有所不同（这与 blkiotune 元素不同（将全局应用到域）。此元素具有以下可选子元素（请注意，任何未指定或所有子元素的任何子元素，值为 0， 代表没有限制）：

- <total_bytes_sec> - 每秒的吞吐量总量（以字节/秒为单位）。这个元素不能与 <read_bytes_sec> 或 <write_bytes_sec> 一同使用。
- <read_bytes_sec> - 每秒读取吞吐量限制（以字节/秒为单位）。
- <write_bytes_sec> - 写入吞吐量限制（以字节/秒为单位）。
- <total_iops_sec> - 每秒 I/O 操作总数。这个元素不能与 <read_iops_sec> 或 <write_iops_sec> 一起使用。

- <read_iops_sec> - 每秒读取 I/O 操作。

- <write_iops_sec> - 每秒写入 I/O 操作。

23.17.1.6. Driver 元素

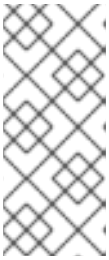
可选的 <driver> 元素允许您指定与用于提供磁盘的虚拟机监控程序驱动程序相关的更多详情。可以使用以下选项：

- 如果虚拟机监控程序支持多个后端驱动程序，则 name 属性选择主后端驱动程序名称，而可选类型属性则提供子类型。

- 可选的 cache 属性控制缓存机制。可能的值有：default,none,writethrough,writeback,directsync （与 writethrough 类似，但它会绕过主机物理机器页面缓存）和 不安全 （主机物理机器可能会缓存所有磁盘 I/O，并忽略来自客户机虚拟机同步请求）。

- 可选的 error_policy 属性控制管理程序在磁盘读取或写入错误上的行为方式。可能的值有停止、报告、忽略，以及 enospace。error_policy 的默认设置是 report。还有一个可选的 rerror_policy，用于控制仅读取错误的行为。如果没有给出 rerror_policy，则 error_policy 同时用于读取和写入错误。如果给出 rerror_policy，它将覆盖 error_policy 的 read 错误。另请注意，enospace 不是读取错误的有效策略，因此如果 error_policy 设为 enospace，并且没有给出 rerror_policy，则将使用读取错误设置 报告。

- 可选的 io 属性控制 I/O 上的特定策略； kvm 客户机虚拟机支持 线程 和 原生。可选的 ioeventfd 属性允许用户为 virtio 磁盘设备设置域 I/O 异步处理。默认值由虚拟机监控程序决定。接受的值为 on 和 off。启用此项可允许客户机虚拟机在单独线程处理 I/O 时执行。通常，在 I/O 期间，遇到高系统 CPU 利用率的客户机虚拟机将从此中受益。另一方面，过载主机物理计算机可能会增加客户机虚拟机 I/O 延迟。但是，建议您不要更改默认设置，并允许虚拟机监控程序确定设置。



注意

ioeventfd 属性包含 在磁盘 XML 部分的 < <driver> > 元素中，以及 设备 XML 部分的驱动程序元素。在以前的情形中，它会影响 virtIO 磁盘，后者则影响 SCSI 磁盘。

- 可选的 event_idx 属性控制设备事件处理的各个方面，并可设置为 on 或 off。如果设为 上的，它将减少中断数量，并退出客户机虚拟机。默认值由虚拟机监控程序决定，默认设置在 上。如果不需要此行为，则 设置会 强制关闭该功能。但是，强烈建议您不要更改默认设置，并允许虚拟机监控程序指示设置。

- 可选的 copy_on_read 属性控制是否将读取后备文件复制到镜像文件中。接受的值可以是 on 或 <off>。copy-on-read 避免重复访问同一后备文件扇区，并在后备文件超过较慢时很有用。默认情况下，copy-on-read 会为 off。

- 可以设置 discard='unmap' 来启用丢弃支持。可以使用 discard='ignore' 来替换同一行，以禁用。discard='ignore' 是默认设置。

23.17.1.7. 其他设备元素

在 device 元素中可能会使用以下属性：

- <boot> - 指定磁盘可引导。

  其他引导值

  - <order> - 确定在引导序列中尝试设备的顺序。
  - <per-device> Boot 元素不能与 BIOS 引导装载程序中的一般引导元素一起使用。

- <encryption> - 指定卷加密方式。

- <ReadOnly> - 表示 guest 虚拟机无法修改该设备。此设置是带有 属性 <device='cdrom'> 的磁盘的默认设置。

- <应> 共享该设备在域间共享（只要虚拟机监控程序和操作系统支持）。如果使用 可共享，则 cache='no' 应该为该设备使用。

- <瞬态> - 表示当客户机虚拟机退出时，应自动恢复对设备内容的更改。对于某些虚拟机监控程序，标记磁盘 临时 可防止域参与迁移或快照。

- *<serial> - 指定客户机虚拟机的硬盘的序列号。例如，<串行>WD-WMAP9A966149</串行>。*

- *<WWN> - 指定虚拟硬盘或 CD-ROM 驱动器的 World Wide Name(WWN)。它必须由 16 位十六进制组成。*

- *<vendor> - 指定虚拟硬盘或 CD-ROM 设备的供应商。它不能超过 8 个可打印字符。*

- *<product> - 指定虚拟硬盘或 CD-ROM 设备的产品。它不能超过 16 个可打印字符*

- *<主机> - 支持以下属性：*

  - *name - 指定主机名*

  - *port - 指定端口号*

  - *transport - 指定传输类型*

  - *socket - 指定套接字的路径*

此元素的含义和元素的数量取决于 协议 属性，如中所示 基于协议的其他主机属性

基于协议的其他主机属性

- *nbd - 指定运行 nbd-server 的服务器，且只能用于一台主机物理机器。此 protcol 的默认端口为 10809。*

- *RBD - 监控 RBD 类型的服务器，并可用于一个或多个主机物理计算机。*

- *sheepdog - 指定 sheepdog 服务器之一（默认为 localhost:7000）），并可与任何主机物理机器一起使用。*

- Gluster - 指定运行 glusterd 守护进程的服务器，只能用于一台主机物理计算机。transport 属性的有效值是 tcp、rdma或 unix。如果未指定任何内容，则假定 tcp。如果传输是 unix，则 socket 属性指定 unix 套接字的路径。

- <address> - 将磁盘放在控制器的给定插槽中。实际的 <控制器> 设备通常被推断，但也可以明确指定它。type 属性是强制的，通常为 pci 或 驱动器。对于 pci 控制器，必须存在适用于总线、插槽 和功能的附加属性，以及可选的 域和 多功能。多功能 默认为 off。对于 驱动器 控制器，提供了更多属性 控制器、总线、目标 和 单元，每个属性的默认设置为 0。

- auth - 提供访问源所需的身份验证凭据。它包括一个强制属性 username，用于标识要在身份验证期间使用的用户名，以及含有强制属性 type 的子元素 机密。

- geometry - 提供覆盖 geometry 设置的功能。这对 S390 DASD-disks 或旧的 DOS-disks 非常有用。它可以有以下参数：

  - cyls - 指定柱面的数量。
  - heads - 指定头数。
  - secs - 指定每个跟踪的扇区数。
  - trans - 指定 BIOS-Translation-Modes，并具有以下值： none、lba 或 auto。

- blockio - 允许使用下面列出的任何块设备属性覆盖块设备：

  blockio 选项

  - logical_block_size - 报告客户机虚拟机操作系统，并描述磁盘 I/O 的最小单位。
  - physical_block_size - 报告客户端虚拟机操作系统，并描述磁盘硬件扇区大小（与磁盘数据对齐相关）。

23.17.2. 设备地址

许多设备具有可选 <的地址> 子元素，用于描述在虚拟客户机中放置于虚拟总线中的设备的位置。如果在输入中省略了地址（或地址中的任何可选属性），libvirt 将生成适当的地址；但是，如果需要更多地控制布局，则需要明确的地址。有关包括 address 元素的设备示例，请参见以下示例。

每个地址都有一个强制属性 类型，用于描述该设备所在的总线。选择为给定设备使用的地址将在设备和客户机虚拟机的架构中限制。例如，磁盘设备使用 type='disk'，而控制台设备则在 32 位 AMD 和 Intel 上使用 type='pci'，或者 AMD64 和 Intel 64、客户机虚拟机，或者 键入='spapr-vio' on PowerPC64 p series guest 虚拟机。每个地址 <类型都> 有额外的可选属性，用于控制要放置该设备的总线的位置。其他属性如下：

- type='pci' - PCI 地址有以下额外属性：
  - 域 （KVM 当前不使用 2 字节十六进制整数）
  - 总线 （包含 0 到 0xff 的十六进制值）
  - 插槽 （包含 0x0 到 0x1f 的十六进制值）
  - 功能 （包含 0 到 7 之间的值）
  - 还提供 多功能 属性，它控制 PCI 控制寄存器中特定插槽或功能的多功能。此多功能属性默认为 'off'，但应设置为 'on'，用于将使用多个功能的插槽 0。
- type='drive' - 驱动器 地址有以下额外属性：
  - controller - （2 位控制器号）
  - 总线 - （2 位总线号）
  - target - （2 位总线号）

- 单元 - （总线中的 2 位单元号）

- type='virtio-serial' - 每个 virtio-serial 地址有以下额外属性：
  - controller - （2 位控制器号）
  - 总线 - （2 位总线号）
  - 插槽 - （总线中的 2 位插槽）
- type='ccid' - 用于 smart-cards 的 CCID 地址，有以下额外属性：
  - 总线 - （2 位总线号）
  - 插槽 - （总线中的 2 位插槽）
- type='usb' - USB 地址有以下额外属性：
  - 总线 - （包含 0 到 0xfff 的十六进制值）
  - port - （最多四个八位字节的点表示法，如 1.2 或 2.1.3.1）
- type='spapr-vio' - On PowerPC pseries guest 虚拟机，可以将设备分配给 SPAPR-VIO 总线。它具有扁平 64 位地址空间；按照惯例，设备通常分配到非零 0x1000 的倍数，但其他地址有效并由 libvirt 允许。可以将额外的 reg 属性（决定开始寄存器的十六进制值地址）分配给此属性。

23.17.3. Controller

根据客户机虚拟机架构，可以将多个虚拟设备分配到单一总线。在一般情况下，libvirt 可以自动推断控制器用于总线。但是，可能需要在客户机虚拟机 XML 中提供显式 <控制器> 元素：

图 23.35. 控制器元素

```
...
<devices>
  <controller type='ide' index='0'/>
  <controller type='virtio-serial' index='0' ports='16' vectors='4'/>
  <controller type='virtio-serial' index='1'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x0a' function='0x0'/>
  <controller type='scsi' index='0' model='virtio-scsi' num_queues='8'/>
  </controller>
  ...
</devices>
...
```

每个控制器都有强制属性 类型，它必须是 "ide "、"fdc"、"scsi"、"sata"、"usb"、"ccid"或"virtio-serial" 或 "virtio-serial" 的一个必需属性 索引，这是描述总线控制器已遇到的十进制整数（用于 地址 元素的控制器属性）。"virtio-serial" 控制器具有两个额外的可选属性，即 端口和 向量 （控制可以通过控制器连接的设备数量）。

<控制器 type='scsi'> 具有一个可选属性 模型，它是 "auto"、"buslogic"、"ibmvscsi"、"lsilogic"、"lsias1068"、"virtio-scsi 或 "vmpvscsi"。<控制器 type='scsi'> 也具有 num_queues 属性，它为指定的队列数启用多队列支持。另外，可以使用 ioeventfd 属性，它指定控制器是否应该对 SCSI 磁盘使用异步处理。可接受的值是 "on" 和 "off"。

"usb" 控制器具有一个可选的属性 模型，它是 "piix3-uhci"、"piix4-uhci"、"ehci"、"ich9-ehci1"、"ich9-uhci1"、"ich9-uhci2"、"ich9-uhci3"、"vt82c686b-uhci1"、"pci-ohci2"、"ich9-uhci2"、"ich9-uhci2"另外，如果需要为客户机虚拟机明确禁用 USB 总线，可以使用 model='none'。PowerPC64 "spapr-vio" 地址没有关联的控制器。

对于本身位于 PCI 或 USB 总线中的控制器，可选的子元素 地址 可以指定控制器到其主总线的确切关系，以及上述语义。

USB 配套控制器有一个可选的子元素 master，以指定与其主控制器相配的确切关系。配套控制器位于与其 master 的同一总线上，因此相应的索引值应该相等。

图 23.36. devices - controllers - USB

```
...
<devices>
  <controller type='usb' index='0' model='ich9-ehci1'>
    <address type='pci' domain='0' bus='0' slot='4' function='7'/>
  </controller>
  <controller type='usb' index='0' model='ich9-uhci1'>
    <master startport='0'/>
    <address type='pci' domain='0' bus='0' slot='4' function='0' multifunction='on'/>
  </controller>
  ...
</devices>
...
```

23.17.4. 设备租用

在使用锁定管理器时，您可以选择针对客户机虚拟机记录设备租用。锁定管理器将确保客户机虚拟机无法启动，除非可以获取租期。当使用传统管理工具进行配置时，域 XML 的以下部分会受到影响：

图 23.37. devices - 设备租期

```
...
<devices>
  ...
  <lease>
    <lockspace>somearea</lockspace>
    <key>somekey</key>
    <target path='/some/lease/path' offset='1024'/>
  </lease>
  ...
</devices>
...
```

lease 部分可以有以下参数：

- lockspace - 识别持有该密钥的锁定空间的任意字符串。锁定管理器可能会对格式或锁定空间名称的长度施加额外限制。
- key - 唯一标识要获取租期的任意字符串。锁定管理器可能会对 格式或密钥的长度施加额外限制。
-

target - 与锁定空间关联的文件的完全限定路径。offset 指定租期存储在文件中的位置。如果锁定管理器不需要偏移，请将值设为 0。

23.17.5. 主机物理机器设备分配

23.17.5.1. USB / PCI 设备

主机物理机器的 USB 和 PCI 设备可以使用 hostdev 元素传递给客户机虚拟机，方法是使用管理工具修改主机物理机器，配置域 XML 文件的以下部分：

图 23.38. 设备 - 主机物理机器设备分配

```
...
<devices>
  <hostdev mode='subsystem' type='usb'>
    <source startupPolicy='optional'>
      <vendor id='0x1234'/>
      <product id='0xbeef'/>
    </source>
    <boot order='2'/>
  </hostdev>
</devices>
...
```

另外，也可以执行以下操作：

图 23.39. 设备 - 主机物理机器设备分配替代

```
...
<devices>
  <hostdev mode='subsystem' type='pci' managed='yes'>
    <source>
      <address bus='0x06' slot='0x02' function='0x0'/>
    </source>
    <boot order='1'/>
    <rom bar='on' file='/etc/fake/boot.bin'/>
  </hostdev>
</devices>
...
```

另外，也可以执行以下操作：

图 23.40. 设备 - 主机物理计算机 scsi 设备分配

```
...
<devices>
  <hostdev mode='subsystem' type='scsi'>
    <source>
      <adapter name='scsi_host0'/>
      <address type='scsi' bus='0' target='0' unit='0'/>
    </source>
    <readonly/>
    <address type='drive' controller='0' bus='0' target='0' unit='0'/>
  </hostdev>
</devices>
..
```

域 XML 的这一部分的组件如下：

表 23.16. 主机物理机器设备分配元素

参数	描述

hostdev	这是描述主机物理设备的主要元素。它接受以下选项：
	• **mode** - 该值始终是 USB 和 PCI 设备的**子系统**。
	• **type** - **usb** for USB 设备和**pci** for PCI 设备。
	• *Managed - 切换设备的**受管** 模式* ：
	  ○ 当为 PCI 设备设置为 **yes** 时，它会附加到客户机机器并从客户机机器分离，并根据需要重新连接到主机机器。**managed='yes'** 用于一般使用设备分配。
	  ○ 为 PCI 和 USB 设备设置为 **no** 或 omitted 时，该设备会保持附加到客户机上。要使设备可用于主机，用户必须在启动客户机或热插拔设备前使用 **virNodeDeviceDettach** 或 **virsh nodedev-dettach** 命令。另外，它们必须在热拔出设备或停止客户机后使用 **virNodeDeviceReAttach** 或 **virsh nodedev-reattach**。**managed='no'** 主要建议用于专用于特定客户机的设备。

参数	描述
source	描述在主机物理计算机中看到的设备。USB 设备可以使用供应商和产品 ID，或使用 address 元素处理设备在主机物理机器上 **的地址。 另一方面上的 PCI 设备仅可以通过其地址进行描述。请注意，USB 设备的源元素可以包含 startupPolicy 属性，该属性可用于定义当未找到指定主机物理机器 USB 设备时要做什么的规则。该属性接受以下值：** • **必需 - 如果因任何原因（默认）缺失，则失败。** • **requisite - 在引导时缺少 Fails，如果在 migrate/restore/revert 上缺少，则断开。** • **可选 - 如果开始尝试时缺少 Drops。**
vendor, 产品	这些元素各自有一个 **id** 属性，用于指定 USB 厂商和产品 ID。可以用十进制、十六进制数（以 0x 开始）或八进制（以 0 开始）指定 ID。
boot	指定该设备是可引导的。属性的顺序决定了在引导序列期间尝试设备的顺序。per-device 引导元素无法与 BIOS 引导装载程序中的一般引导元素一起使用。
rom	用于更改 PCI 设备的 ROM 如何出现在客户机虚拟机中。可选 **bar** 属性可以设置为 **on** 或 **关闭**，并确定该设备的 ROM 是否可以在客户机虚拟机的内存映射中看到。（在 PCI 文档中，**rom bar** 设置会控制 ROM 的 Base Address Register。如果没有指定 **rom bar**，则将使用默认设置。可选的 **file** 属性用于指向要作为设备的 ROM BIOS 向 guest 虚拟机呈现的二进制文件。这可用于为具有 SR-IOV 功能的虚拟功能（VF 没有引导 ROM）提供 PXE 引导 ROM。

参数	描述
address	也具有 总线 和设备属性，用于指定设备出现在主机物理计算机上的 USB 总线和设备编号。**这些属性的值可以以十进制、十六进制（以 0x 开始）或八进制（以 0 开始）形式指定。**对于 **PCI** 设备，该元素包含 **3 个属性，以便通过 lspci 或 virsh nodedev-list** 获取指定设备。

23.17.5.2. 块 / 字符设备

主机物理计算机的块 / 字符设备可以通过管理工具修改域 XML hostdev 元素传递到客户机虚拟机。请注意，这只能通过基于容器的虚拟化来实现。

图 23.41. devices - 主机物理机器设备分配块字符设备

```
...
<hostdev mode='capabilities' type='storage'>
  <source>
    <block>/dev/sdf1</block>
  </source>
</hostdev>
...
```

另一种方法是：

图 23.42. 设备 - 主机物理机器设备分配块字符设备替代 1

```
...
<hostdev mode='capabilities' type='misc'>
  <source>
    <char>/dev/input/event3</char>
  </source>
</hostdev>
...
```

另一种替代方法是：

图 23.43. 设备 - 主机物理机器设备分配块字符设备替代 2

```
...
<hostdev mode='capabilities' type='net'>
  <source>
    <interface>eth0</interface>
  </source>
</hostdev>
...
```

域 XML 的这一部分的组件如下：

表 23.17. 块/字符设备元素

参数	描述
hostdev	这是介绍主机物理设备的主要容器。对于块/字符设备，**直通模式** 始终是 **能力**，**类型为 块设备** 的块，而字符设备的 **char** 代表字符设备。
source	这描述了在主机物理机器中看到的设备。对于块设备，在嵌套块元素中提供了到主机物理机器操作系统中的 **块设备的路径**，而对于 字符设备，将使用 **char** 元素。

23.17.6. 重定向的设备

通过修改域 XML 的以下部分来配置通过字符设备重定向的 USB 设备：

图 23.44. devices - 重定向设备

```
...
<devices>
  <redirdev bus='usb' type='tcp'>
    <source mode='connect' host='localhost' service='4000'/>
    <boot order='1'/>
  </redirdev>
  <redirfilter>
    <usbdev class='0x08' vendor='0x1234' product='0xbeef' version='2.00' allow='yes'/>
    <usbdev allow='no'/>
  </redirfilter>
</devices>
...
```

域 XML 的这一部分的组件如下：

表 23.18. 重定向的设备元素

参数	描述
redirdev	这是描述重定向设备的主要容器。USB 设备的 总线 必须是 **usb**。需要额外的属性类型，与其中一个支持的串行设备类型匹配，以描述隧道的主机物理机器侧： **type='tcp'** 或 **type='spicevmc'**（它使用 SPICE 图形设备的 usbredir 频道）是典型的。**redirdev** 元素具有一个可选的子元素(sub-element) **地址**，可将设备绑定到特定控制器。根据给定 类型，但不需要 **目标** 子元素（因为字符设备的使用者是虚拟机监控程序本身，而不是在客户机虚拟机中可见的设备），所以可能需要 **source**。
boot	指定该设备是可引导的。**order** 属性决定了在引导序列期间尝试设备的顺序。per-device 引导元素无法与 BIOS 引导装载程序中的一般引导元素一起使用。
redirfilter	这用于创建过滤器规则，以过滤从重定向过滤某些设备。它使用 sub-element **usbdev** 定义每个过滤规则。**class** 属性是 USB 类代码。

23.17.7. SmartCard 设备

*虚拟智能卡设备可以通过 **smartcard** 元素提供给客户端虚拟机。主机物理机器上的 USB 智能卡读取器设备不能用于有设备透传的客户机虚拟机。这是因为，主机物理机和客户机虚拟机均不能使用，并且在从客户机虚拟机中删除时，它可以锁定主机物理计算机。因此，一些虚拟机监控程序提供了一个特殊的虚拟设备，可向客户机虚拟机提供智能卡接口，有几种模式来描述如何从主机物理机器获取凭证，甚至是从创建到第三方智能卡供应商的频道中获取的模式。*

使用管理工具配置 USB 设备重定向，以修改域 XML 的以下部分：

图 23.45. devices - 智能卡设备

```
...
<devices>
  <smartcard mode='host'/>
  <smartcard mode='host-certificates'>
    <certificate>cert1</certificate>
    <certificate>cert2</certificate>
    <certificate>cert3</certificate>
    <database>/etc/pki/nssdb/</database>
  </smartcard>
  <smartcard mode='passthrough' type='tcp'>
    <source mode='bind' host='127.0.0.1' service='2001'/>
    <protocol type='raw'/>
    <address type='ccid' controller='0' slot='0'/>
  </smartcard>
  <smartcard mode='passthrough' type='spicevmc'/>
</devices>
...
```

智能卡元素 具有强制属性 模式。在每个模式下，客户机虚拟机会在其 USB 总线中看到类似物理 USB CCID（Chip/Smart 卡接口设备）的设备。

模式属性如下：

表 23.19. SmartCard 模式元素

参数	描述
mode='host'	在这个模式中，虚拟机监控程序会将客户机虚拟机的所有请求中继到通过 NSS 对主机物理机器的智能卡进行直接访问。不需要其他属性或子元素。请参阅以下关于使用可选的 **地址** 子元素的信息。
mode='host-certificates'	这个模式允许您提供三个驻留在主机物理机器数据库中的 NSS 证书名称，而不是要求将智能卡插入到主机物理机器中。这些证书可以使用命令 **certutil -d /etc/pki/nssdb -x -t CT,CT,CT -S -s CN=cert1 -n cert1** -n cert1，并且生成的三个证书名称必须作为三个 **证书子元素** 的内容提供。额外的子元素 **数据库** 可以指定到备用目录的绝对路径（与创建证书时的 **-d** 标志匹配）；如果不存在，则默认为 **/etc/pki/nssdb**。

参数	描述
mode='passthrough'	使用此模式，您可以通过二级字符设备将所有请求连接到第三方提供程序（这样又可能会与智能卡通信或使用三个证书文件），而不是让管理程序直接与主机物理机器通信。在这种运行模式中，需要额外的属性**类型**，与支持的串行设备类型之一匹配，用于描述隧道的主机物理机器侧；**type='tcp'** 或 **type='spicevmc'**（使用 SPICE 图形设备的智能卡频道）是典型的。可根据给定类型需要其他子元素，如 **源**，但不需要 **目标** 子元素（因为字符设备的使用者是虚拟机监控程序本身，而不是在客户机虚拟机中可见的设备）。

每个模式支持可选的子元素 地址，它可以微调智能卡和合计总线控制器之间的相关性。如需更多信息，请参阅 第 23.17.2 节 “设备地址”。

23.17.8. 网络接口

使用管理工具修改网络接口设备，以配置域 XML 的以下部分：

图 23.46. devices - 网络接口

```
...
<devices>
  <interface type='direct' trustGuestRxFilters='yes'>
    <source dev='eth0'/>
    <mac address='52:54:00:5d:c7:9e'/>
    <boot order='1'/>
    <rom bar='off'/>
  </interface>
</devices>
...
```

为客户机虚拟机配置网络接口有几种可能性。这可以通过将值设为 interface 元素的 type 属性来实现。可以使用以下值：

- *"direct" - 将 guest 虚拟机的 NIC 附加到主机物理计算机上的物理 NIC。详情请查看 第 23.17.8.6 节 “将附件直接附加到物理接口”。*
- *"网络" - 建议使用动态或无线网络配置在主机物理机器上连接常规客户机虚拟机的建议配置。详情请查看 第 23.17.8.1 节 “虚拟网络”。*

- "bridge" - 这是在主机物理机器上使用静态有线网络配置的客户机虚拟机连接的建议配置设置。详情请查看 第 23.17.8.2 节 “网桥到 LAN”。

- "Ethernet" - 提供了一种方法，供管理员执行任意脚本以将客户机虚拟机的网络连接到 LAN。详情请查看 第 23.17.8.5 节 “通用以太网连接”。

- "hostdev" - 允许将 PCI 网络设备直接分配给客户机虚拟机，使用通用设备透传。详情请查看 第 23.17.8.7 节 “PCI 透传”。

- "mcast" - 多播组可用于代表虚拟网络。详情请查看 第 23.17.8.8 节 “多播隧道”。

- "用户" - 使用 user 选项设置用户空间 SLIRP 堆栈参数，为外界提供包含 NAT 的虚拟 LAN。详情请查看 第 23.17.8.4 节 “用户空间 SLIRP 堆栈”。

- "服务器" - 使用 server 选项创建 TCP 客户端-服务器架构，以便提供一个虚拟网络，其中一个客户机虚拟机提供网络服务器端，所有其他客户端虚拟机都被配置为客户端。详情请查看 第 23.17.8.9 节 “TCP 隧道”。

每个选项都有一个链接，用于提供更多详细信息。另外，可以使用可选的 <trustGuestRxFilters> 属性定义 <每个接口> 元素，以允许主机物理计算机检测和信任从客户机虚拟机接收的报告。每次接口接收对过滤器的更改时，会发送这些报告。这包括对主 MAC 地址、设备地址过滤器或 vlan 配置的更改。出于安全考虑，<trustGuestRxFilters> 属性会被默认禁用。另请注意，对此属性的支持取决于客户机网络设备模型以及主机物理计算机的连接类型。目前，它只支持 virtio 设备模型以及主机物理计算机上的 macvtap 连接。建议设置可选参数 <trustGuestRxFilters> 的简单用例是，如果您希望为客户机虚拟机提供控制主机物理机器侧过滤器的权限，因为客户机设置的任何过滤器也将在主机上进行镜像。

除了上面列出的属性外，每个 <interface> 元素还可使用可选 <地址> 子元素，可以将接口绑定到特定的 PCI 插槽，属性 type='pci'。如需更多信息，请参阅 第 23.17.2 节 “设备地址”。

23.17.8.1. 虚拟网络

这是在带有动态或无线网络配置的主机物理机器上进行常规虚拟客户机连接的建议配置（或主机物理机硬件详情的多主机物理机器详情），建议在 <网络定义> 中单独描述。此外，它还提供了一个连接，其中包含由指定网络定义描述的详细信息。根据虚拟网络的 转发模式 配置，网络可以完全隔离（无所给的 <转发> 元素），使用 NAT 连接到显式网络设备或默认路由(forward mode='nat')、路由 no NAT（转发

模式='route'），或者直接连接到主机物理机器的网络接口（使用 macvtap）或网桥设备（使用 macvtap 模式(forward mode='bridge|private|passve)）的名称。

对于具有 网桥、private、vepa 和 passthrough 模式的网络，则假定主机物理计算机已在 libvirt 范围之外设置。对于隔离、nat 和路由网络，DHCP 和 DNS 由 libvirt 在虚拟网络中提供，可通过使用 virsh net-dumpxml [networkname] 检查虚拟网络配置来确定 IP 范围。"默认"虚拟网络是设置开箱即用的，它使用 NAT 连接到默认路由，并且 IP 范围为 192.168.122.0/255.255.255.0。每个客户机虚拟机都会创建一个关联的 tun 设备，其名称为 vnetN，它也可通过 <target> 元素覆盖（请参考 第 23.17.8.11 节 “覆盖 target 元素”）。

当接口源是网络时，可以指定一个端口组以及网络的名称；一个网络可能会定义了多个端口组，每个 portgroup 含有不同类网络连接的不同配置信息。另外，类似于 <直接> 网络连接（如下所示），类型为 network 的连接可能会指定 <虚拟> 端口元素，其中要转发到 802.1Qbg 或 802.1Qbh 兼容 虚拟以太网端口聚合器（VEPA)交换机或 Open vSwitch 虚拟交换机。

由于交换机类型取决于主机物理计算机上的 <network> 元素中的配置设置，因此可以接受省略 <virtualport 类型> 属性。您需要指定一次或多次 <指定虚拟> 端口类型。当域启动时，通过组合到定义的类型和属性来构成完整 <的虚拟端口> 元素。这会生成新构建的虚拟端口。请注意，较低虚拟端口的属性无法对高虚拟端口中定义的属性进行更改。接口具有最高优先级，而端口组优先级最低。

例如，若要创建具有 802.1Qbh 交换机和 Open vSwitch 交换机正确工作的网络，您可以选择指定 no 类型，但必须同时提供 profileid 和 interfaceid。从虚拟端口填写的其他属性（如 managerid、typeid 或 profileid ）是可选的。

如果要限制客户机虚拟机仅连接到特定类型的交换机，您可以指定虚拟端口类型，并且只连接具有指定端口类型的交换机。您还可以通过指定附加参数来进一步限制连接。因此，如果指定端口，且主机物理计算机的网络有不同类型的虚拟端口，接口的连接将失败。虚拟网络参数通过使用管理工具来定义，它修改域 XML 的以下部分：

图 23.47. devices - 网络接口

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
  </interface>
  ...
  <interface type='network'>
    <source network='default' portgroup='engineering'/>
    <target dev='vnet7'/>
    <mac address="00:11:22:33:44:55"/>
    <virtualport>
      <parameters instanceid='09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f'/>
    </virtualport>

  </interface>
</devices>
...
```

23.17.8.2. 网桥到 LAN

如 第 23.17.8 节 “网络接口” 所述，这是在带有静态有线网络配置的主机物理机器上的客户机虚拟机连接的建议配置设置。

网桥到 LAN，直接从客户机虚拟机桥接到 LAN。这假定主机物理计算机上有一个网桥设备，其中有一个或多个主机物理机器的物理 NIC enslaved。客户机虚拟机将关联 tun 设备使用名称 <vnetN> 创建，该设备也可通过 <target> 元素覆盖（请参考 第 23.17.8.11 节 “覆盖 target 元素”）。<tun> 设备将从属于网桥。IP 范围或网络配置与 LAN 中使用的内容相同。这为 guest 虚拟机提供完全传入和传出的网络访问，就像物理计算机一样。

在 Linux 系统上，网桥设备通常是标准 Linux 主机物理机器桥接。在支持 Open vSwitch 的主机物理机器上，也可以通过在接口定义中添加 virtualport type='openvswitch'/ 来连接到 Open vSwitch 网桥设备。Open vSwitch 类型 虚拟 端口接受 其参数 元素中的两个参数： interfaceid 是一个用于唯一标识此特定接口的标准 UUID（如果您不指定，则首先定义接口时将生成一个随机的 interfaceid ），以及发送到 Open vSwitch 作为接口 <port-profile> 的可选 配置文件id。要将网桥设置为 LAN 设置，请使用用来配置以下域 XML 部分的管理工具：

图 23.48. devices - 网络接口到 LAN

```
...
<devices>
  ...
  <interface type='bridge'>
    <source bridge='br0'/>
  </interface>
  <interface type='bridge'>
    <source bridge='br1'/>
    <target dev='vnet7'/>
    <mac address="00:11:22:33:44:55"/>
  </interface>
  <interface type='bridge'>
    <source bridge='ovsbr'/>
    <virtualport type='openvswitch'>
      <parameters profileid='menial' interfaceid='09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f'/>
    </virtualport>
  </interface>
  ...
</devices>
```

23.17.8.3. 设置端口伪装范围

如果您要设置端口伪装范围，请按如下所示设置端口：

图 23.49. 端口伪装范围

```
<forward mode='nat'>
   <address start='1.2.3.4' end='1.2.3.10'/>
</forward>  ...
```

这些值应该使用 iptables 命令设置，如 所示 第 17.3 节 “网络地址转换”

23.17.8.4. 用户空间 SLIRP 堆栈

设置用户空间 SLIRP 堆栈参数为外界提供包含 NAT 的虚拟 LAN。虚拟网络具有 DHCP 和 DNS 服务，将为客户机虚拟机提供一个从 10.0.2.15 开始的 IP 地址。默认路由器是 10.0.2.2，DNS 服务器为 10.0.2.3。这个网络是唯一需要客户机虚拟机进行传出访问权限的非特权用户选项。

用户空间 SLIRP 堆栈参数在域 XML 的以下部分中定义：

图 23.50. 设备 - 网络接口 SLIRP 堆栈

```
...
<devices>
  <interface type='user'/>
  ...
  <interface type='user'>
    <mac address="00:11:22:33:44:55"/>
  </interface>
</devices>
...
```

23.17.8.5. 通用以太网连接

这样，管理员可以执行任意脚本以将客户机虚拟机的网络连接到 LAN。客户机虚拟机将具有名称 vnetN 创建的 <tun> 设备，该设备也可通过 <target> 元素覆盖。创建 tun 设备后，将运行 shell 脚本并完成所需的主机物理机器网络集成。默认情况下，此脚本称为 /etc/qemu-ifup，但可以被覆盖（请参阅 第 23.17.8.11 节 “覆盖 target 元素”）。

通用以太网连接参数在域 XML 的以下部分中定义：

图 23.51. devices - 网络接口通用以太网连接

```
...
<devices>
  <interface type='ethernet'/>
  ...
  <interface type='ethernet'>
    <target dev='vnet7'/>
    <script path='/etc/qemu-ifup-mynet'/>
  </interface>
</devices>
...
```

23.17.8.6. 将附件直接附加到物理接口

这将把客户机虚拟机的 NIC 直接附加到主机物理机器的物理接口（如果指定了物理接口）。

这需要 Linux macvtap 驱动程序可用。以下 模式 属性值 vepa （"虚拟以太网端口聚合器"）之一，可以为 macvtap 设备的操作模式选择 桥接 或私有。vepa 是默认模式。

操作直接连接到物理接口涉及在域 XML 在本节中设置以下参数：

图 23.52. devices - 网络接口直接附加到物理接口

```
...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='vepa'/>
  </interface>
</devices>
...
```

独立模式会导致传输数据包的行为，如 表 23.20 “将附件直接附加到物理接口元素” 所示。

表 23.20. 将附件直接附加到物理接口元素

元素	描述
vepa	所有客户机虚拟机的数据包都发送到外部网桥。目的地为同一主机物理计算机上的虚拟客户机虚拟机的数据包，该数据包源自于 VEPA 能力网桥（日常网桥通常不是 VEPA 功能）发送至主机物理计算机。
bridge	其目的地位于与它们源自于目标 macvtap 设备的相同主机物理计算机上的数据包。origin 和 destination 设备都需要处于网桥模式，以进行直接发送。如果其中任何一个处于 **vepa** 模式，则需要一个有 VEPA 的网桥。
私有	所有数据包都发送到外部网桥，并且仅发送到同一主机物理计算机上的目标虚拟机，如果它们通过外部路由器或网关发送，设备会将它们发回到主机物理计算机。如果源或者目标设备处于私有模式，则会显示这个过程。
passthrough	此功能将具有 SR-IOV 功能的虚拟功能直接附加到客户机虚拟机，而不丢失迁移功能。所有数据包都发送到配置的网络设备的 VF/IF。根据设备的功能，可能会应用额外的先决条件或限制；例如，这需要内核 2.6.38 或更高版本。

直接附加的虚拟机的网络访问权限可由硬件交换机管理，这些硬件交换机可以连接到主机物理机器的物理接口。

如果交换机符合 IEEE 802.1Qbg 标准，接口可以具有如下所示的附加参数。virtualport 元素的参数在 IEEE 802.1Qbg 标准中详细记录。值是特定于网络，应当由网络管理员提供。在 802.1Qbg 术语中，虚拟工作站接口(VSI)代表虚拟机的虚拟接口。

请注意，IEEE 802.1Qbg 需要 VLAN ID 的非零值。

可在 表 23.21 “将附件直接附加到物理接口额外元素” 中描述可操作的其他元素：

表 23.21. 将附件直接附加到物理接口额外元素

元素	描述
managerid	VSI Manager ID 标识包含 VSI 类型和实例定义的数据库。这是一个整数值，值 **0** 为保留。
typeid	VSI 类型 ID 标识了 VSI 类型字符化网络访问。VSI 类型通常由网络管理员管理。这是一个整数值。
typeidversion	VSI Type Version 允许多个 VSI 类型版本。这是一个整数值。
instanceid	当创建 VSI 实例（虚拟机的虚拟接口）时，将生成 VSI 实例 ID 标识符。这是一个全局唯一标识符。
profileid	配置集 ID 包含要应用到此接口的端口配置文件的名称。此名称由 port profile 数据库解析为网络参数，这些网络参数将应用到这个接口。

域 XML 中的附加参数包括：

图 23.53. devices - 网络接口直接附加到物理接口附加参数

```
...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0.2' mode='vepa'/>
    <virtualport type="802.1Qbg">
      <parameters managerid="11" typeid="1193047" typeidversion="2" instanceid="09b11c53-8b5c-
4eeb-8f00-d84eaa0aaa4f"/>
    </virtualport>
  </interface>
</devices>
...
```

如果交换机符合 IEEE 802.1Qbh 标准，接口可以具有额外的参数，如下所示。值是特定于网络，应当由网络管理员提供。

域 XML 中的附加参数包括：

图 23.54. devices - 网络接口 - 直接附加到物理接口更多的附加参数

```
...
<devices>
  ...
  <interface type='direct'>
    <source dev='eth0' mode='private'/>
    <virtualport type='802.1Qbh'>
      <parameters profileid='finance'/>
    </virtualport>
  </interface>
</devices>
...
```

profileid 属性包含要应用到此接口的端口配置集的名称。此名称由 port profile 数据库解析为网络参数，这些网络参数将应用到这个接口。

23.17.8.7. PCI 透传

PCI 网络设备（由 源 元素指定）直接分配给使用通用设备透传的客户机虚拟机，然后选择性地将设备的 MAC 地址设置为配置的值，并使用可选指定 的虚拟端口 元素将设备与 802.1Qbh 功能关联（请参阅上面为 type='direct' 网络设备提供的虚拟端口的示例）。请注意，由于标准单一端口 PCI 以太网卡驱动程序设计的限制，只能以这种方式分配 SR-IOV（单根 I/O 虚拟化）虚拟功能(VF)设备。要为客户机虚拟机分配标准单端口 PCI 或 PCIe 以太网卡，请使用传统的 hostdev 设备定义。

请注意，此网络设备的"智能直通"与标准 hostdev 设备的功能非常相似，此方法允许为传递设备指定 MAC 地址 和虚拟 端口。如果不需要这些功能，如果您有一个标准的单端口 PCI、PCIe 或 USB 网卡，且不支持 SR-IOV（因此，在分配给客户机虚拟机域后，或者在将设备分配给客户机虚拟机域后仍然会丢失配置 MAC 地址，或者使用 0.9.11 旧的 libvirt 版本，请使用标准 hostdev 定义，而不是将设备分配给客户机虚拟机而不是 ='host 类型 ）。

图 23.55. devices - 网络接口 PCI 透传

```
...
<devices>
  <interface type='hostdev'>
    <driver name='vfio'/>
    <source>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
    </source>
    <mac address='52:54:00:6d:90:02'>
    <virtualport type='802.1Qbh'>
      <parameters profileid='finance'/>
    </virtualport>
  </interface>
</devices>
...
```

23.17.8.8. 多播隧道

多播组可用于表示虚拟网络。任何具有相同多播组中的网络设备的客户机虚拟机都将相互通信，即使它们驻留在多个物理主机物理机器上。此模式可以作为非特权用户使用。没有默认的 DNS 或 DHCP 支持，且不进行传出网络访问。要提供传出网络访问，一个客户机虚拟机应具有第二个 NIC，它连接到第一个 4 网络类型之一，以提供适当的路由。多播协议也与 用户模式 Linux 客户机虚拟机使用的协议兼容。请注意，使用的源地址必须来自多播地址块。通过使用管理工具操作 接口类型 并创建多播隧道，并将其设置为 **mcast**，并提供 **mac** 地址和 源地址 ，例如：

图 23.56. 设备 - 网络接口多播隧道

```
...
<devices>
  <interface type='mcast'>
    <mac address='52:54:00:6d:90:01'>
    <source address='230.0.0.1' port='5558'/>
  </interface>
</devices>
...
```

23.17.8.9. TCP 隧道

创建 TCP 客户端-服务器架构是提供虚拟网络的另一种方式，其中一个客户机虚拟机提供网络服务器端，所有其他客户机虚拟机都配置为客户端。客户机虚拟机之间的所有网络流量都通过配置为服务器的客户机虚拟机进行路由。此模型也可用于非特权用户。没有默认的 DNS 或 DHCP 支持，且不进行传出网络访问。为提供传出网络访问，一个客户机虚拟机应具有第二个 NIC，该 NIC 连接到第一个 4 个网络类型之一，从而提供适当的路由。通过使用管理工具操作 接口类型 创建 TCP 隧道，并将其设置为 **mcast**，并提供 **mac** 地址和 源地址 ，例如：

图 23.57. devices - 网络接口 TCP 隧道

```
...
<devices>
  <interface type='server'>
    <mac address='52:54:00:22:c9:42'>
    <source address='192.168.0.1' port='5558'/>
  </interface>
  ...
  <interface type='client'>
    <mac address='52:54:00:8b:c9:51'>
    <source address='192.168.0.1' port='5558'/>
  </interface>
</devices>
...
```

23.17.8.10. 设置 NIC 驱动程序的特定选项

有些 NIC 可能具有可调整驱动程序的特定选项。这些选项被设置为接口定义 的驱动程序 子元素的属性。这些选项通过使用管理工具来配置域 XML 的以下部分来设置：

图 23.58. devices - 网络接口设置 NIC 驱动程序特定选项

```
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <model type='virtio'/>
    <driver name='vhost' txmode='iothread' ioeventfd='on' event_idx='off'/>
  </interface>
</devices>
...
```

以下属性可用于 "virtio" NIC 驱动程序：

表 23.22. VirtIO NIC 驱动程序元素

参数	描述
名称	可选 **name** 属性强制使用哪个类型的后端驱动程序。该值可以是 **kvm** （用户空间后端）或 **vhost** （一个内核后端，它需要由内核提供 vhost 模块）；尝试要求 vhost 驱动程序（没有内核支持）将被拒绝。如果 **vhost** 驱动程序存在，默认设置为 vhost，但若不存在，将静默回退到 **kvm**。

参数	描述
txmode	指定在传输缓冲区已满时如何处理数据包传输。该值可以是 **iothread** 或 **timer**。如果设置为 **iothread**，则数据包 tx 在驱动程序的下半的 iothread 中完成（此选项转换为将 **"tx=bh"** 添加到 **kvm** 命令行 "-net-pci 选项）。如果设置为 **timer**，则 KVM 中已执行 tx 工作，如果当前时间有超过 tx 数据，则会在 KVM 迁移到其他操作前设置一个计时器才能执行其他操作；当计时器触发时，将另一个尝试发送更多数据。不建议更改此值。
ioeventfd	为接口设备设置域 I/O 异步处理。默认值由虚拟机监控程序自行决定。接受的值为 **on** 和 **off**。启用此选项可让 KVM 在单独线程处理 I/O 时执行客户机虚拟机。通常，在 I/O 期间，遇到高系统 CPU 利用率的客户机虚拟机将从此中受益。另一方面，过载物理主机计算机也可以增加客户机虚拟机 I/O 延迟。不建议更改此值。
event_idx	**event_idx** 属性控制设备事件处理的一些方面。在 **上**，值可以是 . 或 **off**，它可减少中断次数并退出客户机虚拟机。**如果这种行为是 sub-optimal，则此属性提供了一种强制关闭该功能的方法。不建议更改此值。**

23.17.8.11. 覆盖 target 元素

要覆盖 target 元素，请使用管理工具对域 XML 进行以下更改：

图 23.59. devices - 网络接口覆盖 target 元素

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
  </interface>
</devices>
...
```

如果没有指定目标，某些虚拟机监控程序将自动为创建的 tun 设备生成名称。此名称可以手动指定，但名称不能以 vnet 或 vif 开头，它们是 libvirt 和某些虚拟机监控程序保留的前缀。使用这些前缀手动指定目标将被忽略。

23.17.8.12. 指定引导顺序

要指定引导顺序，请使用管理工具对域 XML 进行以下更改：

图 23.60. 指定引导顺序

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <boot order='1'/>
  </interface>
</devices>
...
```

在支持它的虚拟机监控程序中，您可以设置用于网络引导的特定 NIC。属性顺序决定了在引导序列期间尝试设备的顺序。请注意，每个设备引导元素无法与 BIOS 引导装载程序中的常规引导元素一起使用。

23.17.8.13. Interface ROM BIOS 配置

要指定 ROM BIOS 配置设置，请使用管理工具对域 XML 进行以下更改：

图 23.61. Interface ROM BIOS 配置

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet1'/>
    <rom bar='on' file='/etc/fake/boot.bin'/>
  </interface>
</devices>
...
```

对于支持它的虚拟机监控程序，您可以更改将 PCI 网络设备的 ROM 呈现给客户机虚拟机的方式。bar 属性可以设置为 on 或 关闭，并确定该设备的 ROM 是否可以在客户机虚拟机的内存映射中看到。（在 PCI 文档中，rom bar 设置会控制 ROM 的 Base Address Register。如果没有指定 rom 条，将使用 KVM 默认版本（默认情况下使用 KVM 的旧版本，而较新的 KVM 管理程序则默认使用）。 可选的 file 属性用于指向要作为设备的 ROM BIOS 向 guest 虚拟机呈现的二进制文件。这对为网络设备提供替代引导 ROM 非常有用。

23.17.8.14. 服务质量(QoS)

传入和传出流量可以独立形成，以设置服务质量(QoS)。带宽 元素最多可以有一个入站和一个出站子元素。离开任何这些子元素后，不会将 QoS 应用到该流量方向。因此，要仅形成域的传入流量，仅使用

入站流量，反之亦然。

每个元素 的平均 属性均有一个强制属性（ 或 下文所述）。average 指定接口生成的平均位速率。另外，有两个可选属性：

- peak - 此属性指定网桥可以发送数据的最大速率，以千字节为单位。这种实现的一个限制是出站元素中的此属性，因为 Linux 入口过滤器尚不知道它。

- burst - 指定在峰值速度上可以突发的字节数。属性接受的值是整数。

平均 和 峰值 属性的单位是每秒的 KB 数，而 突发 仅以 KB 为单位设置。另外，入站流量也可以有 floor 属性。这保证了组成接口的最小吞吐量。使用 floor 要求所有流量都经过一个点，因为 QoS 决策可以发生的地方。因此，当 接口 type='network'/ 带有 转发 类型为 route、nat 或 no forward 时，才可以使用它。请注意，在虚拟网络中，所有连接接口至少需要入站 QoS 设置（至少平均为 ），但 floor 属性不需要指定 平均。但是，峰值 和 突发 属性仍需要 平均。目前，ingress qdiscs 可能没有任何类，因此 floor 只能应用到入站流量，而不能为出站流量。

要指定 QoS 配置设置，请使用管理工具对域 XML 进行以下更改：

图 23.62. 服务质量

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet0'/>
    <bandwidth>
      <inbound average='1000' peak='5000' floor='200' burst='1024'/>
      <outbound average='128' peak='256' burst='256'/>
    </bandwidth>
  </interface>
<devices>
...
```

23.17.8.15. 设置 VLAN 标签（仅支持网络类型）

要指定 VLAN 标签配置设置，请使用管理工具对域 XML 进行以下更改：

图 23.63. 设置 VLAN 标签（仅支持网络类型）

```
...
<devices>
  <interface type='bridge'>
    <vlan>
      <tag id='42'/>
    </vlan>
    <source bridge='ovsbr0'/>
    <virtualport type='openvswitch'>
      <parameters interfaceid='09b11c53-8b5c-4eeb-8f00-d84eaa0aaa4f'/>
    </virtualport>
  </interface>
<devices>
...
```

如果客户机虚拟机使用的网络连接支持对客户机虚拟机透明的 VLAN 标记，可选的 vlan 元素可以指定一个或多个 VLAN 标签，以应用到客户机虚拟机的网络流量。只有 OpenvSwitch 和 type='hostdev' 的 SR-IOV 接口支持透明的 VLAN 标记客户机虚拟机流量；其他接口（包括标准 Linux 网桥和 libvirt 自身的虚拟网络）不支持它。802.1Qbh(vn-link)和 802.1Qbg(VEPA)交换机提供自己的方法（在 libvirt 外），以在特定 VLAN 上标记客户机虚拟机流量。要允许多个标签规格（如果是 VLAN 中继），tag 子元素指定要使用的 VLAN 标签（例如，标签 id='42'/）。如果接口定义了多个 vlan 元素，则假定用户希望利用所有指定标签进行 VLAN 中继。如果需要使用单一标签的 VLAN 中继，可选属性 trunk='yes' 可以添加到顶级 vlan 元素中。

23.17.8.16. 修改虚拟链接状态

此元素设置虚拟网络链接状态。属性 状态 的可能值是 up 和 down。如果将 down 指定为值，接口的行为就像网络电缆的断开连接一样。如果这个元素未指定，则默认行为为 。

要指定虚拟链接状态配置设置，请使用管理工具对域 XML 进行以下更改：

图 23.64. 修改虚拟链接状态

```
...
<devices>
  <interface type='network'>
    <source network='default'/>
    <target dev='vnet0'/>
    <link state='down'/>
  </interface>
<devices>
...
```

23.17.9. 输入设备

输入设备允许在客户机虚拟机中与图形帧缓冲的交互。在启用帧缓冲时，会自动提供一个输入设备。可以显式添加其他设备，例如为光标移动提供图形表t。

要指定输入设备配置设置，请使用管理工具对域 XML 进行以下更改：

图 23.65. 输入设备

```
...
<devices>
  <input type='mouse' bus='usb'/>
</devices>
...
```

<input> 元素具有一个强制属性：键入，它可以设置为 鼠标或 tablet .tablet 提供绝对光标移动，而 鼠标 使用相对移动。可以使用可选的 bus 属性来优化确切的设备类型，并可设置为 kvm （半虚拟化）、ps2 和 usb。

input 元素具有可选的子元素 <地址>，它可以将设备绑定到特定的 PCI 插槽（如上所述）。

23.17.10. hub Devices

hub 是一个将单个端口扩展到多个设备的设备，以便将设备连接到主机物理机器系统有更多端口。

要指定 hub 设备配置设置，请使用管理工具对域 XML 进行以下更改：

图 23.66. hub 设备

```
...
<devices>
  <hub type='usb'/>
</devices>
...
```

hub 元素有一个强制属性，键入，它只能设置为 usb。hub 元素有一个可选的 sub-element 地址为，类型为='usb'，可将设备绑定到特定控制器。

23.17.11. 图形帧缓冲

图形设备允许图形化与客户机虚拟机操作系统交互。客户机虚拟机通常具有帧缓冲器或配置了文本控制台，以允许与用户交互。

要指定图形帧缓冲设备配置设置，请使用管理工具对域 XML 进行以下更改：

图 23.67. 图形帧缓冲

```
...
<devices>
  <graphics type='sdl' display=':0.0'/>
  <graphics type='vnc' port='5904'>
    <listen type='address' address='1.2.3.4'/>
  </graphics>
  <graphics type='rdp' autoport='yes' multiUser='yes' />
  <graphics type='desktop' fullscreen='yes'/>
  <graphics type='spice'>
    <listen type='network' network='rednet'/>
  </graphics>
</devices>
...
```

graphics 元素具有强制 类型 属性，它取值 sdl、vnc、rdp、desktop 或 spice，如下表中所述：

表 23.23. 图形帧缓冲主要元素

参数	描述
sdl	这会在主机物理机器桌面上显示一个窗口。它接受以下可选参数： • 要显示的 **display** 属性 • 身份验证标识符的 **xauth** 属性 • 可选 **的完整screen** 属性接受值 **yes** 或 **no**

参数	描述
vnc	启动 VNC 服务器。 • **port** 属性指定 TCP 端口号（使用**-1** 作为旧语法，表示它应该被自动分配）。 • **autoport** 属性是指示要使用的 TCP 端口的自动分配的首选语法。 • **listen** 属性是服务器要侦听的 IP 地址。 • **passwd** 属性以明文形式提供 VNC 密码。 • **keymap** 属性指定要使用的键映射。可以对密码的有效性设置限制，可为 **时间戳 passwdValidTo='2010-04-09T15:51:00** 假定在 UTC 中。 • **连接** 的属性允许在密码更改期间控制连接的客户端。VNC 仅接受 **keep** 值；请注意，所有管理程序可能不支持它。 • KVM 支持在 UNIX 域套接字路径上侦听的 socket 属性，而不是使用 listen/port。
spice	启动 SPICE 服务器. • **port** 属性指定 TCP 端口号（使用 -1 作为旧语法，表示应自动分配它），而 **tlsPort** 提供备选的安全端口号。 • **autoport** 属性是用来指示两个端口号自动分配的新首选语法。 • **listen** 属性是服务器要侦听的 IP 地址。 • **passwd** 属性以明文格式提供 SPICE 密码。 • **keymap** 属性指定要使用的键映射。可以对密码的有效性设置限制，可为 **时间戳 passwdValidTo='2010-04-09T15:51:00** 假定在 UTC 中。 • **连接** 的属性允许在密码更改期间控制连接的客户端。SPICE 接受 **始终保持** 客户端连接，**断开与** 客户端断开连接，且无法更改密码。**请注意，这不受所有虚拟机监控程序的支持。** • **defaultMode 属性设置默认频道安全策略；有效的值为 secure 、insecure 和 default any （如果可能，则会安全使用，但回退到 不安全，如果没有安全路径不可用）。**

参数	描述

当 SPICE 同时配置了普通的并且配置了 TLS 保护的 TCP 端口时，可能需要限制每个端口上可以运行哪些频道。为此，请在主图形元素中添加一个或多个 频道 元素。有效频道名称包括 主、显示、输入、光标、回放、记录、智能卡 和 usbredir。

要指定 SPICE 配置设置，请使用人工工具对域 XML 进行以下更改：

图 23.68. SPICE 配置示例

```
<graphics type='spice' port='-1' tlsPort='-1' autoport='yes'>
  <channel name='main' mode='secure'/>
  <channel name='record' mode='insecure'/>
  <image compression='auto_glz'/>
  <streaming mode='filter'/>
  <clipboard copypaste='no'/>
  <mouse mode='client'/>
</graphics>
```

SPICE 支持对音频、映像和流进行变量压缩设置。这些设置使用以下元素中的 compression 属性来配置：

- *镜像 设置映像压缩（接受 auto_glz、auto_lz 、quic、glz、lz、off）*
- *通过 WAN 对 JPEG 压缩进行 JPEG 的 JPEG 压缩（接受 自动，永不 是 ）*
- *用于配置 WAN 镜像压缩（接受 自动 压缩，从、always）和 回放 的 zlib 启用音频流压缩（接受 或关闭）*

streaming 元素设置流模式。mode 属性可以设置为 filter、all或 off。

另外，复制和粘贴功能（通过 SPICE 代理）由 剪贴板 元素设置。默认情况下启用它，可将 copypaste 属性设置为 no 来禁用。

mouse 元素设置鼠标模式。mode 属性可以设置为 server 或 client。如果没有指定模式，则使用 KVM 默认（客户端 模式）。

其他元素包括：

表 23.24. 其他图形帧缓冲元素

参数	描述
rdp	启动 RDP 服务器。 • **port** 属性指定 TCP 端口号（使用 -1 作为旧语法，表示它应该被自动分配）。 • **autoport** 属性是指示要使用的 TCP 端口的自动分配的首选语法。 • **replaceUser** 属性是一个布尔值，决定是否允许多个同时连接到虚拟机。 • **multiUser** 属性决定现有连接是否必须被丢弃，当新客户端以单一连接模式连接时，VRDP 服务器必须建立新的连接。

参数	描述
desktop	这个值目前为 VirtualBox 域保留。它显示了主机物理机器桌面上的窗口，与 **sdl** 类似，但使用 VirtualBox viewer。与 **sdl** 一样，它接受 **显示的** 可选属性和 **全屏**。
listen	可以指定单独的子元素（请参阅上面的示例），而不是输入用于图形类型的侦听套接字 **vnc** 和 **spice**，而是输入单独的子元素（请参阅上面的示例）： • **键入 -** 设置为 **地址 或网络 。这将告知此 listen 元素是指定要直接使用的地址，还是通过命名网络（然后用于决定要侦听的相应地址）。** • **address -** 该属性将包含要侦听的 **IP 地址或主机名（通过 DNS 查询解析为 IP 地址）。在正在运行的域的"实时" XML 中，此属性将设置为用于侦听的 IP 地址，即使 type='network 的 也是如此。** • **network - 如果 type='network'，network 属性将在 libvirt 的已配置网络列表中包含网络的名称。将检查指定网络配置以确定适当的侦听地址。例如，如果网络的配置中有一个 IPv4 地址（例如，它的转发类型为路由、NAT 或隔离类型），将使用网络配置中列出的第一个 IPv4 地址。如果网络描述主机物理计算机网桥，则将使用与该网桥设备关联的第一个 IPv4 地址。如果网络描述了其中一个 'direct'(macvtap)模式，则将使用第一个转发 dev 的第一个 IPv4 地址。**

23.17.12. 视频设备

要指定视频设备配置设置，请使用管理工具对域 XML 进行以下更改：

图 23.69. 视频设备

```
...
<devices>
  <video>
    <model type='vga' vram='8192' heads='1'>
      <acceleration accel3d='yes' accel2d='yes'/>
    </model>
  </video>
</devices>
...
```

graphics 元素具有一个强制 类型 属性，它取值"sdl"、"vnc"、"rdp"或"desktop"，如下所述：

表 23.25. 图形帧缓冲元素

参数	描述
video	**视频** 元素是描述视频设备的容器。为了向后兼容，如果没有设置视频，但域 XML 中有一个图形元素，则 libvirt 将根据客户机虚拟机类型添加默认 **视频**。**如果没有提供 "ram" 或 "vram"，则会使用默**认值。
model	这具有一个强制 **类型** 属性，它取值 **vga**、**cirrus**、**vmvga**、**kvm**、**vbox** 或 **qxl**，具体取决于可用的虚拟机监控程序功能。您还可以使用 vram 提供以 kibibytes（1024 字节块）的视频内存量以及头条数据的数量。
加速	如果支持加速，则应使用 **加速** 元素中的 **accel3d** 和 **accel2d** 属性启用。
address	可以使用可选的地址子元素将视频设备绑定到特定的 PCI 插槽。

23.17.13. Console、Serial 和 Channel Devices

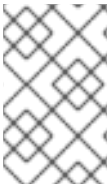
字符设备提供了与虚拟机交互的方式。半虚拟化控制台、串行端口 和频道都归类为字符设备，并使用相同的语法表示。

要指定控制台、频道和其他设备配置设置，请使用管理工具对域 XML 进行以下更改：

图 23.70. 控制台、串行和频道设备

```
...
<devices>
  <serial type='pty'>
    <source path='/dev/pts/3'/>
    <target port='0'/>
  </serial>
  <console type='pty'>
    <source path='/dev/pts/4'/>
    <target port='0'/>
  </console>
  <channel type='unix'>
    <source mode='bind' path='/tmp/guestfwd'/>
    <target type='guestfwd' address='10.0.2.1' port='4600'/>
  </channel>
</devices>
...
```

在每个指令中，顶级元素名称（、控制台、频道）都描述了如何将设备出现在客户机虚拟机中。客户机虚拟机接口由 target 元素配置。向主机物理计算机呈现的界面在顶级元素的 type 属性中给出。主机物理机器接口由 source 元素配置。source 元素可以包含可选的 seclabel，以覆盖在套接字路径上进行标记的方式。如果这个元素不存在，则安全标签将从每个域设置中继承。每个字符 device 元素都有一个可选的子元素 地址，可将设备绑定到特定控制器或 PCI 插槽。



注意

不再支持并行端口和 isa-parallel 设备。

23.17.14. 客户端虚拟机接口

字符设备作为以下类型之一显示在客户机虚拟机中。

要设置串行端口，请使用管理工具对域 XML 进行以下更改：

图 23.71. 客户端虚拟机接口串行端口

```
...
<devices>
  <serial type='pty'>
    <source path='/dev/pts/3'/>
    <target port='0'/>
  </serial>
</devices>
...
```

<目标> 可以有一个 port 属性，用于指定端口号。端口从 0 开始编号。通常有 0、1 或 2 个串行端口。还有一个可选的 type 属性，其值有两个选择，即 isa-serial 或 usb-serial。如果缺少 类型，则默认为 isa-serial。对于 usb-serial，带有 type='usb' 的可选子元素 <地址> 可将设备绑定到特定控制器（如上所述）。

<console> 元素用于表示交互式控制台。根据正在使用的客户机虚拟机类型，控制台可能是半虚拟设备，或者可能是串行设备的克隆，具体取决于以下规则：

- *如果没有设置 targetType 属性，则默认 设备类型 将根据虚拟机监控程序的规则进行。当将 XML 重新查询到 libvirt 时，将添加默认类型。对于完全虚拟化的虚拟机，默认设备类型通常是串行端口。*
- *如果 targetType 属性是 serial，如果没有 <serial> 元素，则 console 元素将复制到 <serial> 元素。如果 <serial> 元素已经存在，则 console 元素将被忽略。*
- *如果 targetType 属性不是 serial，它将被正常对待。*
- *只有第一个 <console> 元素才能使用 serial 的 targetType。辅助控制台必须全部是半虚拟化的。*
- *在 s390 中，console 元素可以使用 sclp 或 sclplm 的 targetType (line mode)。SCLP 是 s390 的原生控制台类型。没有与 SCLP 控制台关联的控制器。*

在以下示例中，在客户机虚拟机上以 /dev/hvc[0-7]（更多信息，请参阅 Fedora 项目的 virtio-serial 页面）中公开 virtio控制台设备：

图 23.72. 客户机虚拟机接口 - virtio 控制台设备

```
...
<devices>
  <console type='pty'>
    <source path='/dev/pts/4'/>
    <target port='0'/>
  </console>

  <!-- KVM virtio console -->
  <console type='pty'>
    <source path='/dev/pts/5'/>
    <target type='virtio' port='0'/>
  </console>
</devices>
...

...
<devices>
  <!-- KVM s390 sclp console -->
  <console type='pty'>
    <source path='/dev/pts/1'/>
    <target type='sclp' port='0'/>
  </console>
</devices>
...
```

如果控制台呈现为串行端口，则 <target> 元素具有与串行端口相同的属性。通常只有一个控制台。

23.17.15. Channel

这代表主机物理机和客户机虚拟机之间的专用通信通道。通过使用管理工具编辑域 XML 的以下部分对客户机虚拟机进行更改来操作：

图 23.73. Channel

```
...
<devices>
  <channel type='unix'>
    <source mode='bind' path='/tmp/guestfwd'/>
    <target type='guestfwd' address='10.0.2.1' port='4600'/>
  </channel>

  <!-- KVM virtio channel -->
  <channel type='pty'>
    <target type='virtio' name='arbitrary.virtio.serial.port.name'/>
  </channel>
  <channel type='unix'>
    <source mode='bind' path='/var/lib/libvirt/kvm/f16x86_64.agent'/>
    <target type='virtio' name='org.kvm.guest_agent.0'/>
  </channel>
  <channel type='spicevmc'>
    <target type='virtio' name='com.redhat.spice.0'/>
  </channel>
</devices>
...
```

这可以多种方式实施。在 <target> 元素的 type 属性中给出具体 <频道类型>。不同的频道类型具有不同的目标属性，如下所示：

- guestfwd - 将客户机虚拟机发送的 TCP 流量发送到给定的 IP 地址，端口转发到主机物理计算机上的频道设备。target 元素必须具有地址和端口属性。
- virtio - 半虚拟化 virtio <频道。频道> 在 /dev/vport* 下的客户机虚拟机中公开，如果指定了可选元素 名称 /dev/virtio-ports/$name （更多信息，请参阅 Fedora 项目的 virtio-serial 页面）。可选元素 地址 可以将频道绑定到特定 类型='virtio-serial' 控制器。使用 KVM 时，如果 name 为 "org.kvm.guest_agent.0"，则 libvirt 可以与客户机虚拟机中安装的客户机代理进行交互，用于进行客户机虚拟机关闭或文件系统等操作。
- spicevmc - 半虚拟化 SPICE 频道.域还必须有 SPICE 服务器作为图形设备，其指向主机物理机器在主频道之间恢复消息。target 元素必须存在，属性 type='virtio'；可选属性名称控制虚拟客户机如何对频道的访问权限，并且默认为 name ='com.redhat.spice.0'。可选的 <address> 元素可将频道绑定到特定 类型='virtio-serial' 控制器。

23.17.16. 主机物理接口

字符设备会将自己的主机物理机器作为以下类型之一：

表 23.26. 字符设备元素

参数	描述	XML 片断
域日志文件	禁用字符设备上的所有输入，并将输出发送到虚拟机的日志文件中。	<devices> <console type='stdio'> <target port='1'/> </console> </devices>
设备日志文件	一个文件被打开，发送到字符设备的所有数据都会写入该文件。请注意，目标目录必须具有 **virt_log_t** SELinux 标签，以便具有此设置的 guest 标签才能成功启动。	<devices> <serial type="file"> <source path="/var/log/vm/vm-serial.log"/> <target port="1"/> </serial> </devices>
虚拟控制台	在虚拟控制台中将字符设备连接到图形帧缓冲。这通常使用一个特殊的热键序列，如 "ctrl+alt+3"。	<devices> <serial type='vc'> <target port="1"/> </serial> </devices>
null 设备	将字符设备连接到 void。没有为输入提供任何数据。写入的所有数据都会被丢弃。	<devices> <serial type='null'> <target port="1"/> </serial> </devices>
伪 TTY	使用 **/dev/ptmx** 分配 Pseudo TTY。**virsh console** 等合适的客户端可以在本地与串行端口交互。	<devices> <serial type="pty"> <source path="/dev/pts/3"/> <target port="1"/> </serial> </devices>
NB 特殊情况	如果 **<控制台类型='pty'，则 TTY 路径也会作为顶层 <控制台> >** 标签上的属性 **tty='/dev/pts/3'** 重复。这为 **<console>** 标签提供了与现有的语法。	

参数	描述	XML 片断
主机物理机器设备代理	字符设备通过 传递到底层物理字符设备。设备类型必须匹配，例如，模拟串行端口应仅连接到主机物理机器串行端口 - 不将串行端口连接到并行端口。	<devices> <serial type="dev"> <source path="/dev/ttyS0"/> <target port="1"/> </serial> </devices>
命名管道	字符设备将输出写入命名管道。有关详细信息，请参见 pipe(7)手册页。	<devices> <serial type="pipe"> <source path="/tmp/mypipe"/> <target port="1"/> </serial> </devices>

参数	描述	XML 片断
TCP client-server	字符设备充当连接到远程服务器的 TCP 客户端。	<devices> <serial type="tcp"> <source mode="connect" host="0.0.0.0" service="2445"/> <protocol type="raw"/> <target port="1"/> </serial> </devices> 或者作为 TCP 服务器等待客户端连接。 <devices> <serial type="tcp"> <source mode="bind" host="127.0.0.1" service="2445"/> <protocol type="raw"/> <target port="1"/> </serial> </devices> 或者，您可以使用 telnet 而不是原始 TCP。另外，您还可以使用 telnet（安全 telnet）和 tls。 <devices> <serial type="tcp"> <source mode="connect" host="0.0.0.0" service="2445"/> <protocol type="telnet"/> <target port="1"/> </serial> <serial type="tcp"> <source mode="bind" host="127.0.0.1" service="2445"/> <protocol type="telnet"/> <target port="1"/> </serial> </devices>

参数	描述	XML 片断
UDP 网络控制台	字符设备充当 UDP netconsole 服务、发送和接收数据包。这是丢失的服务。	<devices> <serial type="udp"> <source mode="bind" host="0.0.0.0" service="2445"/> <source mode="connect" host="0.0.0.0" service="2445"/> <target port="1"/> </serial> </devices>
UNIX 域套接字客户端-服务器	字符设备充当 UNIX 域套接字服务器，接受本地客户端的连接。	<devices> <serial type="unix"> <source mode="bind" path="/tmp/foo"/> <target port="1"/> </serial> </devices>

23.17.17. 声音设备

可以使用 sound 元素将虚拟声卡附加到主机物理计算机上。

图 23.74. 虚拟声卡

```
...
<devices>
  <sound model='ac97'/>
</devices>
...
```

sound 元素包含一个强制属性，即 model，它指定模拟良好的设备。有效值特定于底层虚拟机监控程序，但典型的选择是 'sb16'、'ac97' 和 'ich6'。此外，设置了 "ich6" 模型的声音元素也可以有可选的 codec 子元素，用于将各种音频代码附加到音频设备。如果没有指定，则会附加默认的 codec 以允许回放和记录。有效值为 'duplex'（非行和换行符）和 'micro'（非发言人和微电话）。

图 23.75. 声音设备

```
...
<devices>
  <sound model='ich6'>
    <codec type='micro'/>
  <sound/>
</devices>
...
```

每个声音元素都有一个可选的子元素 <地址>，可以将设备绑定到特定的 PCI 插槽（上面记录的）。

注意

Red Hat Enterprise Linux 7 不再支持 es1370 声音设备。使用 ac97 替代。

23.17.18. watchdog 设备

可以使用 watchdog 元素向客户机虚拟机添加虚拟硬件 <watchdog> 设备。watchdog 设备需要在客户机虚拟机中附加驱动程序和管理守护进程。目前，当 watchdog 触发时不支持通知。

图 23.76. watchdog 设备

```
...
<devices>
  <watchdog model='i6300esb'/>
</devices>
...

...
<devices>
  <watchdog model='i6300esb' action='poweroff'/>
</devices>
...
```

这个 XML 中声明以下属性：

- model - 必需的 model 属性指定模拟真正的 watchdog 设备。有效值特定于底层虚拟机监控程序。

- model 属性可能采用以下值：

- *i6300esb - 建议的设备，模拟 PCI Intel 6300ESB*
- *ib700 - 模拟 ISA iBase IB700*

- *action - 可选 action 属性描述了 watchdog 过期时要执行的操作。有效值特定于底层虚拟机监控程序。action 属性可以具有以下值：*
  - *重置 - 默认设置，强制重置 guest 虚拟机*
  - *shutdown - 正常关闭 guest 虚拟机（不推荐）*
  - *poweroff - 强制关闭客户机虚拟机*
  - *暂停 - 暂停客户机虚拟机*
  - *none - 不执行任何操作*
  - *dump - 自动转储客户机虚拟机。*

请注意，"shutdown"操作要求客户机虚拟机响应 ACPI 信号。在 watchdog 已过期的情况下，客户机虚拟机通常无法响应 ACPI 信号。因此，不建议使用 'shutdown'。另外，在文件 /etc/libvirt/kvm.conf 中，可以使用 auto_dump_path 来配置保存转储文件的目录。

23.17.19. 设置 Panic 设备

Red Hat Enterprise Linux 7 hypervisor 可以使用 pvpanic 机制检测 Linux 客户机虚拟机内核 panic。调用 pvpanic 时，pvpanic 会将一条信息发送到 libvirtd 守护进程，后者将启动预配置的反应。

要启用 pvpanic 设备，请执行以下操作：

- 在主机计算机上的 /etc/libvirt/qemu.conf 文件中添加或取消注释以下行：

```
auto_dump_path = "/var/lib/libvirt/qemu/dump"
```

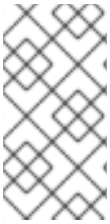
- 运行 virsh edit 命令以编辑指定 guest 的域 XML 文件，并将 panic 添加到 设备 父级元素中。

```
<devices>
  <panic>
    <address type='isa' iobase='0x505'/>
  </panic>
</devices>
```

<address> 元素指定 panic 的地址。默认 ioport 为 0x505。在大多数情况下，不需要指定地址。

libvirtd 响应崩溃的方式是由域 XML 的 <on_crash> 元素决定的。可能的操作如下：

- coredump-destroy - 捕获客户机虚拟机的内核转储并关闭客户端。
- coredump-restart - 捕获客户机虚拟机的内核转储并重启客户机。
- preserve - 将客户机虚拟机引导至等待进一步操作。



注意

如果启用了 kdump 服务，它将优先于 <on_crash> 设置，且不会执行所选 <on_crash> 操作。

有关 pvpanic 的更多信息，请参阅 相关知识库文章。

23.17.20. 内存 Balloon 设备

Balloon 设备可以指定虚拟机 RAM 的一部分，不被使用（称为" Bon tion the balloon"的进程），以便释放该内存供主机释放，或用于该主机上的其他虚拟机。当虚拟机再次需要内存时，可以暂停 balloon，主机可以将 RAM 重新分发到虚拟机。

内存气球大小由 <当前内存> 与 <内存设置> 之间的区别决定。例如，如果 <内存> 设置为 2 GiB，并且 <当前Memory> 设为 1 GiB，则 balloon 包含 1 GiB。如果需要手动配置，可以使用 virsh setmem 命令设置 <当前的Memory> 值，并使用 virsh setmaxmem 命令设置 <内存值>。

警告

如果 <修改当前内存> 的值，请确保为客户机操作系统有足够的内存才能正常工作。如果设置值太低，客户机可能会变得不稳定。

虚拟内存 Balloon 设备会自动添加到所有 KVM 客户机虚拟机。在 XML 配置中，这由 <memballoon> 元素表示。内存膨胀由 libvirt 服务管理，并在适当的情况下自动添加。因此，除非需要分配一个特定的 PCI 插槽，否则不需要在客户机虚拟机 XML 中显式添加此元素。请注意，如果需要显式禁用 <memballoon> 设备，可以使用 model='none'。

以下示例显示了 libvirt 自动添加的 memballoon 设备：

图 23.77. 内存 balloon 设备

```
...
<devices>
  <memballoon model='virtio'/>
</devices>
...
```

以下示例显示了手动添加使用静态 PCI 插槽 2 请求的设备：

图 23.78. 手动添加内存 balloon 设备

```
...
<devices>
  <memballoon model='virtio'>
    <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/>
  </memballoon>
</devices>
...
```

所需的 模型 属性指定提供哪种类型的 balloon 设备。有效值特定于虚拟化平台；在 KVM 管理程序中，'virtio' 是默认设置。

23.18. 存储池

虽然所有存储池后端共享相同的公共 API 和 XML 格式，但它们具有不同的功能级别。有些卷可能允许创建卷，另一些仅允许使用预先存在的卷。有些情况下可能会对卷大小或放置有限制。

存储池文档的顶级是 <池>。它只有一个属性 类型，它可采用以下值：dir、fs、netfs、disk、iscsi、logical、scsi、mpath、rbd、sheepdog 或 gluster。

23.18.1. 为存储池提供元数据

以下 XML 示例显示了可添加到存储池中的元数据标签。在这个示例中，池是一个 iSCSI 存储池。

图 23.79. 常规元数据标签

```
<pool type="iscsi">
  <name>virtimages</name>
  <uuid>3e3fce45-4f53-4fa7-bb32-11f34168b82b</uuid>
  <allocation>10000000</allocation>
  <capacity>50000000</capacity>
  <available>40000000</available>
  ...
</pool>
```

本例中使用的元素在 表 23.27 “virt-sysprep 命令” 中进行了说明。

表 23.27. virt-sysprep 命令

元素	描述
<名称>	为存储池提供一个名称，对于主机物理机器必须是唯一的。在定义存储池时这是必须的。
<uuid>	为存储池提供必须全局唯一的标识符。虽然提供 UUID 是可选的，但如果创建存储池时没有提供 UUID，则会自动生成 UUID。
<allocation>	为存储池提供总存储分配。由于元数据开销，这可能大于所有存储卷中分配的总和。这个值以字节为单位表示。此元素为只读，不应更改值。
<capacity>	提供池的总存储容量。由于底层设备限制，可能无法对存储卷使用完整的容量。这个值以字节为单位。此元素为只读，不应更改值。
<可用>	提供可在存储池中分配新存储卷的可用空间。由于底层设备限制，可能无法将所有可用空间分配给单个存储卷。这个值以字节为单位。此元素为只读，不应更改值。

23.18.2. 源元素

<在池> 元素内，可以定义单一 <源> 元素（只能一个）。<源的子元素> 取决于存储池类型。以下是可以使用的 XML 的一些示例：

图 23.80. 源元素选项 1

```
 ...
<source>
  <host name="iscsi.example.com"/>
  <device path="demo-target"/>
  <auth type='chap' username='myname'>
    <secret type='iscsi' usage='mycluster_myname'/>
  </auth>
  <vendor name="Acme"/>
  <product name="model"/>
</source>
...
```

图 23.81. 源元素选项 2

```
...
<source>
  <adapter type='fc_host' parent='scsi_host5' wwnn='20000000c9831b4b'
wwpn='10000000c9831b4b'/>
</source>
...
```

源接受的子元素在 表 23.28 “源子元素命令” 中阐述。<>

表 23.28. 源子元素命令

元素	描述
<device>	为由主机物理机器设备支持的存储池提供源（基于 **<池 type=>**（如 第 23.18 节 “存储池” 所示）。根据后端驱动程序，可以重复多次。包含一个属性 **路径**，它是块设备节点的完全限定路径。
<dir>	为由目录（**<池类型='dir'）支持的存储池>**提供源，或选择基于文件系统的存储池中的子目录（**<池类型='gluster'>**）。此元素可能仅每一次（**<池>**）。此元素接受单个属性（**<路径>**），这是到后备目录的完整路径。
<adapter>	为 SCSI 适配器支持的存储池提供源（**<池 type='scsi'>**）。此元素可能仅每一次（**<池>**）。属性名称是 SCSI 适配器名称（ex. "scsi_host1"。虽然 "host1" 仍支持向后兼容，但不建议这样做。属性 **类型** 指定适配器类型。有效值为 **'fc_host'| 'scsi_host'**。如果省略并指定了 **name** 属性，则默认为 **type='scsi_host'**。为保持向后兼容性， **type ='scsi_host'** 适配器的属性类型是可选的，但 **type='fc_host'** adapter 是必需的。属性 **wwn** (Word Wide Node Name)和 **wwpn** (Word Wide Port Name)由 **type='fc_host'** 适配器用来唯一标识光纤通道存储结构中的设备（该设备可以是 HBA 或 vHBA）。必须指定 **wwn 和** wwpn。有关如何获得 (v)HBA 的 wwn/wwpn 的说明，请参考 第 20.27.11 节 “收集设备配置设置”。可选属性 **parent** 指定 **type='fc_host'** adapter 的父设备。
<主机>	为从远程服务器存储支持的存储池提供源 (**type='netfs'|'iscsi'|'sheepdog'|'gluster'|'sheepdog'|'gluster'**)。此元素应当与 **<目录或> <设备元素组合使用>**。包含属性名称，即 **服务器的主机名或** IP 地址。可以选择性地包含协议特定端口号的 **port** 属性。

元素	描述
<auth>	如果存在，**<auth>** 元素通过设置 **type** 属性(pool **type='iscsi'|'rbd'**)提供访问源所需的身份验证凭证。**类型** 必须是 **='chap'** 或 **type='ceph'**。将"ceph"用于 Ceph RBD（Rados 块设备）网络源，将 "iscsi" 用于 CHAP(Challenge-Handshake Authentication Protocol)iSCSI 目标。另外，必需属性 **username** 可识别在身份验证期间使用的用户名以及带有强制属性类型的子元素 secret，以将其绑定到含有实际密码或其他凭证的 libvirt secret 对象。域 XML 有意不会公开密码，只向管理密码的对象的引用。secret 元素需要带有 secret 对象的 UUID 的 uuid 属性，或者与 secret 对象中指定的密钥匹配的用法属性。
<名称>	从指定元素类型提供由存储设备支持的存储池源，它可取值：（**<键入> ='logical'|'rbd'|'sheepdog',gluster'**）。
<format>	提供有关存储池类型格式的信息，它可采用以下值：**<type> ='logical'|'disk'|'fs'|'netfs'**）。请注意，此值特定于后端。这通常用来指示文件系统类型或网络文件系统类型，或者分区表类型或者 LVM 元数据类型。因为所有驱动程序都需要具有默认值，所以该元素是可选的。
<vendor>	提供有关存储设备供应商的可选信息。它包含单一 **<属性名称>**，其值特定于后端。
<product>	提供有关存储设备产品名称的可选信息。它包含单一 **<属性名称>**，其值特定于后端。

23.18.3. 创建目标元素

一个 <target> 元素包含在以下类型的顶层 <池> 元素中：（键入 ='dir'\|'fs'\|'netfs'\|'disk'\|'disk'\|'iscsi'\|'scsi'\|'mpath'）。该标签用于描述存储池到主机文件系统的映射。它可以包含以下子元素：

图 23.82. 目标元素 XML 示例

```
<pool>
  ...
  <target>
    <path>/dev/disk/by-path</path>
    <permissions>
      <owner>107</owner>
      <group>107</group>
      <mode>0744</mode>
      <label>virt_image_t</label>
    </permissions>
    <timestamps>
      <atime>1341933637.273190990</atime>
      <mtime>1341930622.047245868</mtime>
      <ctime>1341930622.047245868</ctime>
    </timestamps>
    <encryption type='...'>
      ...
    </encryption>
  </target>
</pool>
```

表(表 23.29 “目标子元素”)解释了父 <目标> 元素中有效的子元素：

表 23.29. 目标子元素

元素	描述
<路径>	提供存储池将映射到本地文件系统命名空间的位置。对于基于文件系统或者目录的存储池，它将是创建存储卷的目录的名称。对于基于设备的存储池，它将是设备节点所在的目录的名称。对于后者，**/dev/**可能像逻辑选择一样，但是，设备的节点无法保证系统重启后稳定，因为它们根据需要进行分配。最好使用稳定位置，例如其中一个 **/dev/disk/by-{path,id,uuid,label}** 位置。
<权限>	目前，这仅适用于基于目录或文件系统的存储池，它们作为目录映射到本地文件系统命名空间中。它提供有关构建存储池时用于最终目录的权限的信息。**<mode>** 元素包含八进制权限集。**<owner>** 元素包含数字用户 ID。**<group>** 元素包含数字组 ID。**<label>** 元素包含 MAC（如 SELinux）标签字符串。

元素	描述
<timestamps>	提供有关存储卷的时间信息。最多四个子元素存在，其中 timestamp **='atime'|'ctime|'ctime'|'ctime'** |'mtime' 包含访问、更改、更改和修改存储卷的时间。使用的时间格式为。**< <nanoseconds> >** 自 epoch 的开头（1 Jan 1970 年 1 月 1 日）开始。如果 host 操作系统或文件系统不支持 nano 秒分辨率为 0，则省略纳秒部分。这是只读属性，在创建存储卷时忽略。
<encryption>	如果存在，指定存储卷的加密方式。如需更多信息，请参阅 libvirt 上游页面。

23.18.4. 设置设备扩展

如果存储池公开有关其底层放置或分配方案的信息，<源> 元素中的 <device> 元素可能包含可用扩展的信息。某些存储池有一个约束，必须完全在一个约束（如磁盘分区池）内分配存储卷。因此，应用程序可以通过扩展信息来确定新存储卷的最大可能大小。

对于支持扩展信息的存储池，<每个设备> 元素内都将有零个或多个 <自由> 的元素。这些元素各自包含两个属性，即 <start> 和 <end>，它可提供设备的扩展边界，单位为字节。

23.19. 存储卷

存储卷通常是文件或设备节点；从 1.2.0 开始，可选的 output-only 属性类型列出了实际类型（文件、块、dir、网络或 netdir），

23.19.1. 常规元数据

<volume> 元素的 top 部分包含称为元数据的信息，如这个 XML 示例所示：

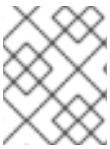
图 23.83. 存储卷的一般元数据

```
...
<volume type='file'>
  <name>sparse.img</name>
  <key>/var/lib/libvirt/images/sparse.img</key>
  <allocation>0</allocation>
  <capacity unit="T">1</capacity>
  ...
</volume>
```

表(表 23.30 “卷子元素”)解释了父 <卷> 元素中有效的子元素：

表 23.30. 卷子元素

元素	描述
<名称>	为存储卷提供一个名称，它对存储池是唯一的。在定义存储卷时这是必须的。
<key>	提供用于标识单个存储卷的存储卷的标识符。在某些情况下，可以有不同的键标识单个存储卷。在创建存储卷时无法设置此字段，因为它总是被生成。
<allocation>	提供存储卷的总存储分配。如果存储卷被稀疏分配，这可能会小于逻辑容量。如果存储卷消耗大量元数据，则可能大于逻辑容量。这个值以字节为单位。如果在创建存储卷时省略，存储卷会在创建时被完全分配。如果设置为小于容量的值，则存储池可以选择决定为稀疏分配存储卷。不同类型的存储池可能会以不同的方式对待稀疏存储卷。例如，逻辑池不会在其满时自动扩展存储卷的分配；用户负责配置它或将 **dmeventd** 配置为自动这样做。默认情况下，这以字节为单位指定。请查看 注意
<capacity>	为存储卷提供逻辑容量。默认情况下，这个值以字节为单位，但可以使用与 注意 所述相同的语义指定 **<单位>** 属性。**<>**在创建存储卷时这是组合的。
<source>	提供有关存储卷的基本存储分配的信息。这可能不适用于某些存储池类型。
<目标>	提供有关本地主机物理机器上存储卷的表示信息。



注意

必要时，可以指定可选属性 单元 来调整传递值。此属性可与元素 <分配> <和容量> 一起使用。属性 单元可接受的值包括 ：

- *b 或 bytes （以字节为单位）*
- *KB 代表 KB*
- *k 或 KiB 用于 kibibytes*
- *MB 表示 MB*
- *M 或 MiB 代表兆字节*
- *GB 表示 GB*
- *g 或 GiB 用于千兆字节*
- *TB 代表 TB*
- *T 或 TiB 代表 tebibytes*
- *PB for PB for PB*
- *P 或 PiB 代表 pebibytes*
- *EB 用于 exabytes*
- *e 或 EiB 表示 exbibytes*

23.19.2. 设置目标元素

<target> 元素可以放入 <卷> 顶级元素中。它被用来描述存储卷上在主机物理计算机文件系统中完成的映射。这个元素可能采用以下子元素：

图 23.84. 目标子元素

```
<target>
  <path>/var/lib/libvirt/images/sparse.img</path>
  <format type='qcow2'/>
  <permissions>
    <owner>107</owner>
    <group>107</group>
    <mode>0744</mode>
    <label>virt_image_t</label>
  </permissions>
  <compat>1.1</compat>
  <features>
    <lazy_refcounts/>
  </features>
 </target>
```

<目标> 的特定子元素在 表 23.31 “目标子元素” 中阐述：

表 23.31. 目标子元素

元素	描述
<路径>	提供在本地文件系统上访问存储卷的位置，作为绝对路径。这是一个只读属性，不应在创建卷时指定。
<format>	提供有关池特定卷格式的信息。对于基于磁盘的存储池，它将提供分区类型。对于文件系统或基于目录的存储池，它将提供文件格式类型（如 cow、qcow、vmdk、raw）。如果在创建存储卷时省略，则使用存储池的默认格式。实际格式由 **type** 属性指定。如需有效值列表，请参阅 第 13.2 节 “使用存储池” 中特定存储池的部分。
<权限>	提供有关创建存储卷时要使用的默认权限的信息。目前，这仅适用于目录或基于文件系统的存储池，其中分配的存储卷是简单文件。对于存储卷为设备节点的存储池，热插拔脚本决定权限。它包含四个子元素：**<mode>** 元素包含八进制权限集。**<owner>** 元素包含数字用户 ID。**<group>** 元素包含数字组 ID。**<label>** 元素包含 MAC（如 SELinux）标签字符串。

元素	描述
<compat>	指定兼容性级别。目前，这仅用于 **<type='qcow2'>** 卷。有效值为 **<compat>**0.10**</compat>** for qcow2（版本 2）和**</ <compat>>** 1.1 compat for qcow2（版本 3），目前用于指定镜像应兼容的 QEMU 版本。如果存在 **<feature>** 元素，则会使用**</ <compat>>** 1.1 compat。如果省略，则使用 qemu-img default。
<功能>	特定于格式的功能。介绍仅用于 **<格式 type='qcow2'/>**（版本 3）。有效子元素包括 **<lazy_refcounts/>**。这可减少元数据写入和清除量，从而改进初始写入性能。这种改进特别适用于直写缓存模式，在崩溃后需要修复镜像并允许延迟引用计数器更新。建议您在 qcow2（版本 3）中使用此功能，因为实现此功能时速度更快。

23.19.3. 设置备份存储元素

个 <后备Store> 元素包含在顶级 <卷元素中>。此标签用于描述存储卷的可选写时复制后备存储。它可以包含以下子元素：

图 23.85. 后备存储子元素

```
<backingStore>
  <path>/var/lib/libvirt/images/master.img</path>
  <format type='raw'/>
  <permissions>
    <owner>107</owner>
    <group>107</group>
    <mode>0744</mode>
    <label>virt_image_t</label>
  </permissions>
</backingStore>
```

表 23.32. 后备存储子元素

元素	描述
<路径>	提供在本地文件系统上访问后备存储的位置，作为绝对路径。如果省略，则没有这个存储卷的后备存储。

元素	描述
<format>	提供有关池特定后备存储格式的信息。对于基于磁盘的存储池，它将提供分区类型。对于文件系统或基于目录的存储池，它将提供文件格式类型（如 cow、qcow、vmdk、raw）。实际格式通过 **<type>** 属性指定。如需有效值列表，请参阅特定于池的文档。大多数文件格式都需要一种相同格式的后备存储，但 qcow2 格式则允许不同的后备存储格式。
<权限>	提供有关备份文件的权限的信息。它包含四个子元素：**<owner>** 元素包含数字用户 ID。**<group>** 元素包含数字组 ID。**<label>** 元素包含 MAC（如 SELinux）标签字符串。

23.20. 安全标签

<seclabel> 元素允许控制安全驱动程序的运行。有三种基本操作模式：'dynamic'，libvirt 会自动生成唯一的安全标签 "静态"，其中应用程序/管理员选择标签或 "none"（其中禁用限制）。使用动态标签生成时，libvirt 将始终自动重新标记与虚拟机关联的任何资源。使用静态标签分配时，管理员或应用程序必须确保在任何资源上正确设置标签，但在需要时可以启用自动重新标记。

如果 libvirt 使用了多个安全驱动程序，则可以使用多个 seclabel 标签，每个驱动程序一个，各个标签引用的安全驱动程序可以通过属性 模型 来定义。顶级安全标签的有效输入 XML 配置有：

图 23.86. 安全标签

```
<seclabel type='dynamic' model='selinux'/>

<seclabel type='dynamic' model='selinux'>
  <baselabel>system_u:system_r:my_svirt_t:s0</baselabel>
</seclabel>

<seclabel type='static' model='selinux' relabel='no'>
  <label>system_u:system_r:svirt_t:s0:c392,c662</label>
</seclabel>

<seclabel type='static' model='selinux' relabel='yes'>
  <label>system_u:system_r:svirt_t:s0:c392,c662</label>
</seclabel>

<seclabel type='none'/>
```

如果未在输入 XML 中提供 "type" 属性，则将使用安全驱动程序默认设置，这可能是 "无" 或 "动态"。如果设置了 <baselabel> 但没有设置 'type'，则类型会假定为 'dynamic'。当查看带有自动资源重

新标记 active 的正在运行的虚拟客户机的 XML 时，将包括额外的 XML 元素 imagelabel。这是一个只输出的元素，在用户提供的 XML 文档中将忽略它。

可以使用以下值操作以下元素：

- 类型 - 使用静态、动态 或无 操作来确定 libvirt 是否自动生成唯一的安全标签。
- model - 与当前激活的安全模型匹配的有效安全模型名称。
- 重新标记 - 是的，或 没有。如果使用动态标签分配，则必须始终为 yes。使用静态标签分配时，将默认为 no。
- <label> - 如果使用静态标签，则必须指定分配给虚拟域的完整安全标签。内容的格式取决于所使用的安全驱动程序：
  - SELinux ：SELinux 上下文.
  - AppArmor ：Armor 配置集。
  - DAC: owner 和 group 以冒号分隔。它们可以定义为用户/组名称或 UID/GID。驱动程序将首先尝试将这些值解析为名称，但前导加号可用于强制驱动程序将其解析为 UID 或 GID。
- <baselabel> - 如果使用动态标签，可以选择性地使用它来指定基本安全标签。内容的格式取决于所使用的安全驱动程序。
- <imagelabel> - 这是一个输出元素，它显示与虚拟域关联的资源上所使用的安全标签。内容的格式取决于所使用的安全驱动程序。当重新标记生效时，还可以通过禁用标签（如果 NFS 或其它文件系统中缺少安全标签）或请求备用标签（当管理应用创建特殊标签时，可以微调标签）或请求备用标签（在某个域间共享特殊标签时，使用该文件）。当 seclabel 元素附加到特定路径而不是顶级域分配时，只支持属性 relabel 或 sub-element 标签。

23.21. 示例虚拟机 XML 配置

下表显示了客户机虚拟机(VM)的 XML 配置示例，也称为 域 XML，并解释了配置的内容。

要获得虚拟机的 XML 配置，请使用 virsh dumpxml 命令。有关编辑虚拟机配置的详情，请参考 虚拟化入门指南。

表 23.33. 示例域 XML 配置

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> <vcpu placement='static'>1</vcpu>	这是名为 Testguest1 的 KVM，分配了 1024 MiB 的 RAM。有关配置常规虚拟机参数的详情，请参考 第 23.1 节 “通用信息和元数据”。
<vcpu placement='static'>1</vcpu>	客户机虚拟机有 1 个分配的 vCPU。有关 CPU 分配的详情，请参考 第 23.4 节 “CPU 分配”。
<os> <type arch='x86_64' machine='pc-i440fx-2.9'>hvm</type> <boot dev='hd'/> </os>	机器构架被设置为 AMD64 和 Intel 64 架构，并使用 Intel 440FX 机器类型来决定功能兼容性。从硬盘引导操作系统。有关修改操作系统参数的详情，请参考 第 23.2 节 “操作系统引导”。
<features> <acpi/> <apic/> <vmport state='off'/> </features>	虚拟机监控程序功能 **acpi** 和 **apic** 被禁用，并关闭 VMWare IO 端口。有关修改虚拟机监控程序功能的详情，请参考 -第 23.14 节 “Hypervisor 功能”。
<cpu mode='host-passthrough' check='none'/>	客户机 CPU 功能与主机 CPU 上的功能相同。有关修改 CPU 功能的详情，请参考 -第 23.12 节 “CPU 型号和拓扑”。

域 XML 部分	描述
<clock offset='utc'> <timer name='rtc' tickpolicy='catchup'/> <timer name='pit' tickpolicy='delay'/> <timer name='hpet' present='no'/> </clock>	客户机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。有关修改时间保存设置的详情，请参考 -第 23.15 节 “timekeeping”。
<on_poweroff>destroy</on_poweroff> <on_reboot>restart</on_reboot> <on_crash>destroy</on_crash>	当虚拟机关闭或操作系统意外终止时，libvirt 会终止客户机并释放其所有分配的资源。重新引导 guest 后，它会使用相同的配置来重新启动。有关配置这些设置的详情请参考 -第 23.13 节 “事件配置”。
<pm> <suspend-to-mem enabled='no'/> <suspend-to-disk enabled='no'/> </pm>	这个客户机虚拟机的 S3 和 S4 ACPI 睡眠状态被禁用。
<devices> <emulator>/usr/bin/qemu-kvm</emulator> <disk type='file' device='disk'> <driver name='qemu' type='qcow2'/> <source file='/var/lib/libvirt/images/Testguest.qcow2'/> <target dev='hda' bus='ide'/> <address type='drive' controller='0' bus='0' target='0' unit='0'/> </disk> <disk type='file' device='cdrom'> <driver name='qemu' type='raw'/> <target dev='hdb' bus='ide'/> <readonly/> <address type='drive' controller='0' bus='0' target='0' unit='1'/> </disk>	虚拟机使用 **/usr/bin/qemu-kvm** 二进制文件模拟。另外，它连接了两个磁盘。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，其逻辑设备名称被设置为 **hda**。有关管理磁盘的更多信息，请参阅 -第 23.17.1 节 “硬盘驱动器、软盘和 CD-ROMs”。

域 XML 部分	描述
<controller type='usb' index='0' model='ich9-ehci1'> <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x7'/> </controller> <controller type='usb' index='0' model='ich9-uhci1'> <master startport='0'/> <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0' multifunction='on'/> </controller> <controller type='usb' index='0' model='ich9-uhci2'> <master startport='2'/> <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x1'/> </controller> <controller type='usb' index='0' model='ich9-uhci3'> <master startport='4'/> <address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x2'/> </controller> <controller type='pci' index='0' model='pci-root'/> <controller type='ide' index='0'> <address type='pci' domain='0x0000' bus='0x00' slot='0x01' function='0x1'/> </controller> <controller type='virtio-serial' index='0'> <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/> </controller>	虚拟机使用四个控制器来附加 USB 设备，以及用于 PCI-Express(PCIe)设备的 root 控制器。另外，可以使用 **virtio-serial** 控制器，它可让虚拟机以各种方式与主机交互，如串行控制台。有关配置控制器的更多信息，请参阅 -第 23.17.3 节 “Controller”。
<interface type='network'> <mac address='52:54:00:65:29:21'/> <source network='default'/> <model type='rtl8139'/> <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/> </interface>	在虚拟机中设置了网络接口，它使用 **default** 虚拟网络和 **rtl8139** 网络设备模型。有关配置网络接口的详情，请参考 -第 23.17.8 节 “网络接口”。

域 XML 部分	描述
<serial type='pty'> <target port='0'/> </serial> <console type='pty'> <target type='serial' port='0'/> </console> <channel type='spicevmc'> <target type='virtio' name='com.redhat.spice.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，它可启用与主机通信最多的辅助虚拟机。控制台使用泛虚拟化 **SPICE** 频道。这个设置是自动设置的，我们不推荐修改这些设置。有关字符设备的概述，请参考 -第 23.17.8 节 “网络接口”。有关 **串行端口** 和 **控制台的详情，** 请参考 第 23.17.14 节 “客户端虚拟机接口”。有关 **频道** 的详情，请参考 第 23.17.15 节 “Channel”。
<input type='mouse' bus='ps2'/> <input type='keyboard' bus='ps2'/>	虚拟机使用虚拟 **ps2** 端口，该端口设置为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。如需更多信息，请参阅 第 23.17.9 节 “输入设备”。
<graphics type='spice' autoport='yes'> <listen type='address'/> <image compression='off'/> </graphics>	虚拟机使用 **SPICE** 协议，通过自动分配的端口号和关闭映像压缩渲染其图形输出。有关配置图形设备的详情，请参考 第 23.17.11 节 “图形帧缓冲”。
<sound model='ich6'> <address type='pci' domain='0x0000' bus='0x00' slot='0x04' function='0x0'/> </sound> <video> <model type='qxl' ram='65536' vram='65536' vgamem='16384' heads='1' primary='yes'/> <address type='pci' domain='0x0000' bus='0x00' slot='0x02' function='0x0'/> </video>	为虚拟机设置了 **ICH6** HDA 声音设备，QEMU **QXL** 半虚拟帧缓冲设备则设置为视频加速器。这个设置是自动设置的，我们不推荐修改这些设置。有关配置 **声设备** 的详情，请参考 第 23.17.17 节 “声音设备”。有关配置 **视频设备**，请参考 第 23.17.12 节 “视频设备”。

域 XML 部分	描述
<redirdev bus='usb' type='spicevmc'> <address type='usb' bus='0' port='1'/> </redirdev> <redirdev bus='usb' type='spicevmc'> <address type='usb' bus='0' port='2'/> </redirdev> <memballoon model='virtio'> <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/> </memballoon> </devices> </domain>	虚拟机有两个重定向程序来远程附加 USB 设备，打开 内存 ballooning。这个设置是自动设置的，我们不推荐修改这些设置。详情请查看 第 23.17.6 节 “重定向的设备”

部分 III. 附录

这部分论述了虚拟化问题的常见问题和解决方案，提供了如何在多个构架中使用 KVM 虚拟化的说明，以及如何使用 IOMMU 组。这部分还包含有关虚拟化软件包的额外支持和产品限制的信息。

附录 A. 故障排除

本章论述了有关 Red Hat Enterprise Linux 7 虚拟化问题的常见问题和解决方案。

阅读本章，了解与虚拟化技术相关的一些常见问题。建议您在 Red Hat Enterprise Linux 7 上试验并测试虚拟化，以培养您的故障排除技能。

如果您在本文档中找到答案，则虚拟化社区可能会在线找到答案。如需 Linux 虚拟化网站列表，请参阅第 D.1 节 “在线资源”。

另外，您可以在 Red Hat Knowledgebase 中了解有关 RHEL 7 中虚拟化故障排除的更多信息。

A.1. 调试和故障排除工具

本节总结了系统管理员应用程序、联网实用程序和调试工具。您可以使用这些标准系统管理工具和日志来帮助进行故障排除：

- kvm_stat - 检索 KVM 运行时统计信息.如需更多信息，请参阅 第 A.4 节 “kvm_stat”。
- ftrace - Traces 内核事件。如需更多信息，请参阅什么是 ftrace 以及如何使用？ 解决方案文章（需要订阅）。
- vmstat - 显示虚拟内存统计信息。有关更多信息，请使用 man vmstat 命令。
- iostat - 提供 I/O 负载统计数据.如需更多信息，请参阅 Red Hat Enterprise Linux 性能调优指南
- lsof - 列出打开文件.有关更多信息，请使用 man lsof 命令。
- SystemTap - 用于监控操作系统的脚本实用程序。如需更多信息，请参阅 Red Hat Enterprise Linux 开发人员指南。
- 崩溃 - 分析内核崩溃转储数据或实时系统。如需更多信息，请参阅 Red Hat Enterprise

Linux Kernel Crash 转储指南。

- sysrq - 内核在控制台不响应时响应的关键组合。如需更多信息，请参阅 Red Hat 知识库。

这些联网工具可以帮助对虚拟化网络问题进行故障排除：

- ip addr,ip route, 和 ip monitor

- tcpdump - 诊断网络上的数据包流量。此命令可用于查找网络身份验证中的网络异常和问题。tcpdump 的图形版本名为 wireshark。

- brctl - 检查并配置 Linux 内核中的以太网网桥配置的网络实用程序。例如：

```
$ brctl show
bridge-name    bridge-id          STP  enabled  interfaces
-----------------------------------------------------------------------------
virtbr0            8000.fefffffff       yes        eth0

$ brctl showmacs virtbr0
port-no          mac-addr                local?       aging timer
1                fe:ff:ff:ff:ff:          yes            0.00
2                fe:ff:ff:fe:ff:          yes            0.00
$ brctl showstp virtbr0
virtbr0
bridge-id              8000.fefffffffff
designated-root        8000.fefffffffff
root-port              0                 path-cost           0
max-age                20.00             bridge-max-age      20.00
hello-time             2.00              bridge-hello-time   2.00
forward-delay          0.00              bridge-forward-delay  0.00
aging-time             300.01
hello-timer            1.43              tcn-timer           0.00
topology-change-timer  0.00              gc-timer            0.02
```

下面是一些用于对虚拟化进行故障排除的有用命令：

- strace 是一个命令，可跟踪由另一个进程接收和使用的系统调用和事件。

- vncviewer 连接到在服务器或虚拟机上运行的 VNC 服务器。使用 yum install tigervnc 命令安装 vncviewer。

- vncserver 在您的服务器中启动远程桌面。为您提供使用远程会话运行图形用户界面（如 virt-manager）的能力。使用 yum install tigervnc-server 命令安装 vncserver。

除了上面列出的所有命令外，检查虚拟化日志也很有用。如需更多信息，请参阅 第 A.6 节 “虚拟化日志”。

A.2. 创建转储文件

您可以向文件请求客户机虚拟机核心的转储，以便可以诊断虚拟机中的错误，例如 crash 实用程序。

警告

在 Red Hat Enterprise Linux 7.5 及更新的版本中，kernel Address Space Randomization(KASLR) 功能可防止客户机转储文件被 崩溃 读取。要解决这个问题，将 <vmcoreinfo/> 元素添加到客户机 XML 配置文件的 <features> 部分。

但请注意，如果目标主机使用不支持 <vmcoreinfo/> 的操作系统，迁移 使用 <vmcoreinfo/> 的客户机会失败。这包括 Red Hat Enterprise Linux 7.4 及更早版本，以及 Red Hat Enterprise Linux 6.9 及更早版本。

A.2.1. 创建 virsh Dump 文件

执行 virsh dump 命令请求会将虚拟客户机核心转储到文件的请求，以便诊断虚拟机中的错误。运行此命令可能需要您手动确保对通过参数 corefilepath 指定的文件和路径的正确权限。virsh dump 命令与内核转储（或 crash 实用程序）类似。

如需更多信息，请参阅创建虚拟客户机内核的转储文件。

A.2.2. 使用 Python 脚本保存内核转储

dump-guest-memory.py python 脚本实施 GNU Debugger(GDB)扩展，用于提取并保存主机上 qemu-kvm 进程崩溃后内核转储中的客户机内存。如果主机 QEMU 进程崩溃与客户机操作相关，则在

QEMU 进程崩溃时调查客户机状态会很有用。

python 脚本实施 GDB 扩展。这是 GDB 的新命令。在使用 GDB 打开原始(crashed)QEMU 进程的内核转储文件后，可将 python 脚本加载到 GDB 中。然后可从 GDB 提示符执行新命令。这会将 QEMU 内核转储中的客户机内存转储提取到新的本地文件。

使用 dump-guest-memory.py python 脚本：

1. 在系统中安装 qemu-kvm-debuginfo 软件包。

2. 启动 GDB，打开为崩溃的 /usr/libexec/qemu-kvm 二进制文件保存的核心转储文件。debug 符号会自动加载。

3. 在 GDB 中载入新命令：

```
# source /usr/share/qemu-kvm/dump-guest-memory.py
```

注意

加载 python 脚本后，内置 GDB 帮助命令 可以提供有关 dump-guest-memory 扩展的详细信息。

4. 在 GDB 中运行命令。例如：

```
# dump-guest-memory /home/user/extracted-vmcore X86_64
```

5. 使用 crash 实用程序打开 /home/user/extracted-vmcore，以便虚拟客户机内核分析。

有关从 QEMU 核心文件 提取用于 崩溃 实用程序的客户机虚拟机内核的更多信息，请参阅如何从 'gcore' 生成的 qemu core 文件中提取用于"crash"实用程序的 ELF 内核。

A.3. 使用 SYSTEMTAP FLIGHT RECORDER 在 CONSTANT BASIS 上捕获 TRACE 数据

您可以使用 qemu-kvm 软件包中提供的 systemtap initscript 来捕获 QEMU 跟踪数据。这个软件包

使用 SystemTap 的航班记录器模式跟踪所有正在运行的客户端虚拟机，并将结果保存到主机上的固定大小缓冲中。填充缓冲区时新条目将覆盖旧的 trace 条目。

过程 A.1. 配置并运行 systemtap

1. 安装软件包

   运行以下命令安装 systemtap-initscript 软件包：

```
# yum install systemtap-initscript
```

2. 复制配置文件

   运行以下命令，将 systemtap 脚本和配置文件复制到 systemtap 目录中：

```
# cp /usr/share/qemu-kvm/systemtap/script.d/qemu_kvm.stp /etc/systemtap/script.d/
# cp /usr/share/qemu-kvm/systemtap/conf.d/qemu_kvm.conf /etc/systemtap/conf.d/
```

   要启用的追踪事件集合在 qemu_kvm.stp 中给出。可以自定义此 SystemTap 脚本来添加或删除 /usr/share/systemtap/tapset/qemu-kvm-simpletrace.stp 中提供的 trace 事件。

   可以对 qemu_kvm.conf 进行 SystemTap 自定义，以控制动态记录器缓冲区的大小，以及是否要将 trace 存储到内存中，或在磁盘中。

3. 启动服务

   运行以下命令启动 systemtap 服务：

```
# systemctl start systemtap qemu_kvm
```

4. 使 systemtap 已启用在系统引导时运行

   运行以下命令，启用 systemtap 服务以在引导时运行：

```
# systemctl enable systemtap qemu_kvm
```

5. 确认服务正在运行

   运行以下命令确认该服务是否正常工作：

```
# systemctl status systemtap qemu_kvm
qemu_kvm is running...
```

过程 A.2. 检查 trace 缓冲

1. 创建 trace 缓冲转储文件

   运行以下命令，创建一个名为 trace.log 的 trace 缓冲转储文件，并将其放置在 tmp 目录中：

```
# staprun -A qemu_kvm >/tmp/trace.log
```

   您可以将文件名和位置更改为其他内容。

2. 启动服务

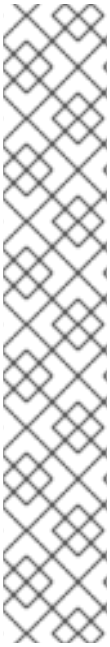
   当上一步停止该服务时，运行以下命令再次启动该服务：

```
# systemctl start systemtap qemu_kvm
```

3. 将 trace 内容转换为可读的格式

   要将 trace 文件内容转换为更易读的格式，请输入以下命令：

```
# /usr/share/qemu-kvm/simpletrace.py --no-header /usr/share/qemu-kvm/trace-events
/tmp/trace.log
```



注意

应该注意以下几点和限制：

- systemtap 服务默认为禁用。
- 启用此服务时，性能较小，但它取决于启用了哪些事件。
- /usr/share/doc/qemu-kvm-*/README.systemtap 中有一个 README 文件。

A.4. KVM_STAT

kvm_stat 命令是一个 python 脚本，可从 kvm 内核模块检索运行时统计信息。kvm_stat 命令可用于诊断对 kvm 可见的客户机行为。特别是，与客户机相关的问题。目前，报告的统计信息适用于整个系

统；报告所有正在运行的 guest 的行为。要运行此脚本，您需要安装 qemu-kvm-tools 软件包。如需更多信息，请参阅 第 2.2 节 “在现有 Red Hat Enterprise Linux 系统上安装虚拟化软件包”。

kvm_stat 命令要求加载 kvm 内核模块并挂载 debugfs。如果没有启用这些功能，命令将输出启用 debugfs 或 kvm 模块所需的步骤。例如：

```
# kvm_stat
Please mount debugfs ('mount -t debugfs debugfs /sys/kernel/debug')
and ensure the kvm modules are loaded
```

如果需要，挂载 debugfs ：

```
# mount -t debugfs debugfs /sys/kernel/debug
```

kvm_stat 输出

kvm_stat 命令输出所有客户机和主机的统计信息。在命令终止前更新输出（使用 Ctrl+c 或 q 键）。请注意，您屏幕上看到的输出可能有所不同。有关输出元素的说明，请点击任何术语来链接到该失效。

```
# kvm_stat

kvm statistics
 kvm_exit                 17724       66
  Individual exit reasons follow, see kvm_exit (NAME) for more information.
 kvm_exit(CLGI)              0        0
 kvm_exit(CPUID)             0        0
 kvm_exit(CR0_SEL_WRITE)     0        0
 kvm_exit(EXCP_BASE)         0        0
 kvm_exit(FERR_FREEZE)       0        0
 kvm_exit(GDTR_READ)         0        0
 kvm_exit(GDTR_WRITE)        0        0
 kvm_exit(HLT)              11       11
 kvm_exit(ICEBP)             0        0
 kvm_exit(IDTR_READ)         0        0
 kvm_exit(IDTR_WRITE)        0        0
 kvm_exit(INIT)              0        0
 kvm_exit(INTR)              0        0
 kvm_exit(INVD)              0        0
 kvm_exit(INVLPG)            0        0
 kvm_exit(INVLPGA)           0        0
 kvm_exit(IOIO)              0        0
 kvm_exit(IRET)              0        0
 kvm_exit(LDTR_READ)         0        0
 kvm_exit(LDTR_WRITE)        0        0
 kvm_exit(MONITOR)           0        0
 kvm_exit(MSR)              40       40
 kvm_exit(MWAIT)             0        0
 kvm_exit(MWAIT_COND)        0        0
```

```
kvm_exit(NMI)                 0        0
kvm_exit(NPF)                 0        0
kvm_exit(PAUSE)               0        0
kvm_exit(POPF)                0        0
kvm_exit(PUSHF)               0        0
kvm_exit(RDPMC)               0        0
kvm_exit(RDTSC)               0        0
kvm_exit(RDTSCP)              0        0
kvm_exit(READ_CR0)            0        0
kvm_exit(READ_CR3)            0        0
kvm_exit(READ_CR4)            0        0
kvm_exit(READ_CR8)            0        0
kvm_exit(READ_DR0)            0        0
kvm_exit(READ_DR1)            0        0
kvm_exit(READ_DR2)            0        0
kvm_exit(READ_DR3)            0        0
kvm_exit(READ_DR4)            0        0
kvm_exit(READ_DR5)            0        0
kvm_exit(READ_DR6)            0        0
kvm_exit(READ_DR7)            0        0
kvm_exit(RSM)                 0        0
kvm_exit(SHUTDOWN)            0        0
kvm_exit(SKINIT)              0        0
kvm_exit(SMI)                 0        0
kvm_exit(STGI)                0        0
kvm_exit(SWINT)               0        0
kvm_exit(TASK_SWITCH)         0        0
kvm_exit(TR_READ)             0        0
kvm_exit(TR_WRITE)            0        0
kvm_exit(VINTR)               1        1
kvm_exit(VMLOAD)              0        0
kvm_exit(VMMCALL)             0        0
kvm_exit(VMRUN)               0        0
kvm_exit(VMSAVE)              0        0
kvm_exit(WBINVD)              0        0
kvm_exit(WRITE_CR0)           2        2
kvm_exit(WRITE_CR3)           0        0
kvm_exit(WRITE_CR4)           0        0
kvm_exit(WRITE_CR8)           0        0
kvm_exit(WRITE_DR0)           0        0
kvm_exit(WRITE_DR1)           0        0
kvm_exit(WRITE_DR2)           0        0
kvm_exit(WRITE_DR3)           0        0
kvm_exit(WRITE_DR4)           0        0
kvm_exit(WRITE_DR5)           0        0
kvm_exit(WRITE_DR6)           0        0
kvm_exit(WRITE_DR7)           0        0
kvm_entry                 17724       66
kvm_apic                  13935       51
kvm_emulate_insn          13924       51
kvm_mmio                  13897       50
varl-kvm_eoi               3222       12
kvm_inj_virq               3222       12
kvm_apic_accept_irq        3222       12
kvm_pv_eoi                 3184       12
kvm_fpu                     376        2
```

```
kvm_cr                  177     1
kvm_apic_ipi            278     1
kvm_msi_set_irq         295     0
kvm_pio                 79      0
kvm_userspace_exit      52      0
kvm_set_irq             50      0
kvm_pic_set_irq         50      0
kvm_ioapic_set_irq      50      0
kvm_ack_irq             25      0
kvm_cpuid               90      0
kvm_msr                 12      0
```

变量解释：

- **kvm_ack_irq - 中断确认数量(PIC/IOAPIC)中断确认.**
- **kvm_age_page - 内存管理单元(MMU)通知程序页时迭代的数量.**
- **kvm_apic - APIC 寄存器数目。**
- **kvm_apic_accept_irq - 本地 APIC 中可接受的中断数。**
- **kvm_apic_ipi - 处理器中断的数量.**
- **kvm_async_pf_completed - 异步页面错误完成次数.**
- **kvm_async_pf_doublefault - 异步页面错误的数量停止.**
- **kvm_async_pf_not_present - 异步页面错误初始化号.**
- **kvm_async_pf_ready - 异步页面错误完成次数.**
- **kvm_cpuid - 已执行的 CPUID 指令数。**

- *kvm_cr - 陷阱和模拟控制寄存器(CR)访问（CR0、CR3、CR4、CR8）。*
- *kvm_emulate_insn - 模拟指令的数量。*
- *kvm_entry - 模拟指令的数量。*
- *kvm_eoi - 中断(EOI)通知的可编程中断控制器(APIC)的数量.*
- *kvm_exit - VM-exits 数 .*
- *kvm_exit(NAME) - 特定于处理器的单一退出。如需更多信息，请参阅您的处理器文档。*
- *kvm_fpu - KVM 浮点单元(FPU)的数量.*
- *kvm_hv_hypercall - Hyper-V 超calls 数.*
- *kvm_hypercall - 非Hyper-V 超calls 的数量.*
- *kvm_inj_exception - 注入到 guest 中的异常数量.*
- *kvm_inj_virq - 注入到客户机中的中断数。*
- *kvm_invlpga - INVLPGA 指令的数量已截获。*
- *kvm_ioapic_set_irq - 虚拟 IOAPIC 控制器的中断级别数变化。*
- *kvm_mmio - 模拟内存映射的 I/O(MMIO)操作数量.*

- *kvm_msi_set_irq - 消息信号中断(MSI)的数量。*
- *kvm_msr - 特定型号寄存器(MSR)访问的数量.*
- *kvm_nested_intercepts - L1 mvapich L2 嵌套 SVM 交换机的数量。*
- *kvm_nested_vmrun - L1 nested L2 嵌套 SVM 交换机的数量。*
- *kvm_nested_intr_vmexit - 由于中断窗口导致嵌套虚拟机退出注入的数量。*
- *kvm_nested_vmexit - 在执行嵌套(L2)guest 时退出管理程序.*
- *kvm_nested_vmexit_inject - L2 nested L1 嵌套交换机的数量。*
- *kvm_page_fault - 管理程序处理的页面错误数量.*
- *kvm_pic_set_irq - 对虚拟可编程中断控制器(PIC)的更改。*
- *kvm_pio - 模拟程序 I/O(PIO)操作的数量。*
- *kvm_pv_eoi - 输入的半虚拟化结束(EOI)事件的数量.*
- *kvm_set_irq - 通用 IRQ 控制器级别的中断级别更改（counts PIC、IOAPIC 和 MSI）。*
- *kvm_skinit - SVM SKINIT 的数量可退出。*
- *kvm_track_tsc - 时间戳计数器(TSC)写入的数量。*

- kvm_try_async_get_page - 异步页面错误尝试次数.
- kvm_update_master_clock - pvclock masterclock 更新的数量.
- kvm_userspace_exit - 退出给用户空间的数量.
- kvm_write_tsc_offset - TSC offset 写入数.
- vcpu_match_mmio - SPTE 缓存内存映射的 I/O(MMIO)的数量。

kvm_stat 命令的输出信息由 KVM 管理程序导出，该文件位于 /sys/kernel/debug/tracing/events/kvm/ 目录中。

A.5. 使用 SERIAL CONSOLES 的故障排除

Linux 内核可以将信息输出到串行端口。这可用于调试带有视频设备或无外设服务器的内核 panic 和硬件问题。

为 KVM 客户机启用串行控制台输出：

1. 确保 guest 的域 XML 文件包括串行控制台的配置。例如：

```
<console type='pty'>
  <source path='/dev/pts/16'/>
  <target type='virtio' port='1'/>
  <alias name='console1'/>
</console>
```

2. 在客户机上，请遵循 如何为 Red Hat Enterprise Linux 7 启用串行控制台？ 文章。

然后，您可以使用以下命令访问串行控制台，其中 guestname 是客户机虚拟机的名称：

```
# virsh console guestname
```

您还可以使用 virt-manager 显示虚拟文本控制台。在客户机控制台窗口中，从 View 菜单中选择 Serial 1 in Text Consoles。

A.6. 虚拟化日志

以下方法可用于访问关于系统管理程序和您客户机上事件的已记录数据。在对系统中的虚拟化进行故障排除时，这非常有用。

- 每个客户机都有一个日志，保存在 /var/log/libvirt/qemu/ 目录中。日志命名为 GuestName.log，并在达到大小限制时定期压缩。

- 要查看 systemd Journal 中的 libvirt 事件，请使用以下命令：

  ```
  # journalctl _SYSTEMD_UNIT=libvirtd.service
  ```

- auvirt 命令显示与系统管理程序上客户机相关的审核结果。可通过选择特定的客户机、时间范围和信息格式来缩小显示的数据。例如，以下命令提供当前日期的 testguest 虚拟机上的事件摘要。

  ```
  # auvirt --start today --vm testguest --summary
  Range of time for report:       Mon Sep  4 16:44 - Mon Sep  4 17:04
  Number of guest starts:         2
  Number of guest stops:          1
  Number of resource assignments: 14
  Number of related AVCs:         0
  Number of related anomalies:    0
  Number of host shutdowns:       0
  Number of failed operations:    0
  ```

  您还可以配置 auvirt 信息，使其自动包含在 systemd Journal 中。为此，请编辑 /etc/libvirt/libvirtd.conf 文件，并将 audit_logging 参数的值设置为 1。

  如需更多信息，请参阅 auvirt man page。

- 如果在 Virtual Machine Manager 中遇到任何错误，您可以检查 $HOME/.virt-manager/ 目录中的 virt-manager.log 文件中生成的数据。

- 有关虚拟机监控程序系统的审计日志，请查看 /var/log/audit/audit.log 文件。

- 根据客户端操作系统，也可以将各种系统日志文件保存在客户端中。

有关 Red Hat Enterprise Linux 中登录的更多信息，请参阅 系统管理员指南。

A.7. 循环设备错误

如果使用基于文件的客户机镜像，您可能需要增加配置的循环设备数量。默认配置允许最多 8 个活跃的循环设备。如果需要超过八个基于文件的客户机或循环设备，可以在 /etc/modprobe.d/目录中调整配置的循环设备数量。添加以下行：

```
options loop max_loop=64
```

这个示例使用 64，但您可以指定另一个数字来设置最大循环值。您可能还必须在您的系统中实施支持 loop 设备支持的客户机。要将支持 guest 的循环设备用于完全虚拟化的系统，请使用 phy: device 或 file: file: 文件。

A.8. 实时迁移错误

有些情况下，当客户机更改内存太快，实时迁移过程必须再次转移它，且无法完成（聚合）。

当前的实时迁移实施会将默认迁移时间配置为 30ms。这个值决定了迁移结束时客户机暂停时间，以便传输遗留时间。数值越高，实时迁移将被聚合

A.9. 在 BIOS 中启用 INTEL VT-X 和 AMD-V 虚拟化硬件扩展

注意

为扩展您的专业知识，您可能也对红帽虚拟化 (RH318) 培训课程感兴趣。

这部分论述了如何识别硬件虚拟化扩展并在 BIOS 中启用它们（如果被禁用）。

在 BIOS 中可以禁用 Intel VT-x 扩展。某些笔记本电脑供应商的 CPU 中默认禁用了 Intel VT-x 扩展。

AMD-V 的 BIOS 中无法禁用虚拟化扩展。

有关启用禁用虚拟化扩展的说明，请参阅以下部分。

验证 BIOS 中启用了虚拟化扩展。Intel VT 或 AMD-V 的 BIOS 设置通常位于 Chipset 或 Processor 菜单中。菜单名称可能因本指南的不同，虚拟化扩展设置可能会在安全设置或其他 非标准 菜单名称中找到。

过程 A.3. 在 BIOS 中启用虚拟化扩展

1. 重新引导计算机并打开系统的 BIOS 菜单。根据系统，通常可以通过按 删除 键、F1 键或 Alt 和 F4 键来完成此操作。

2. 在 BIOS 中启用虚拟化扩展

注意

以下很多步骤可能会因您的主板、处理器类型、芯片组和 OEM 而异。有关配置系统的正确信息，请参阅您的系统附带的文档。

a. 打开 Processor 子菜单 The processor settings 菜单可以在 Chipset、Advanced CPU Configuration 或 Northbridge 中隐藏。

b. 启用 Intel 虚拟化技术 （也称为 Intel VT-x）。在 BIOS 中无法禁用 AMD-V 扩展，应该已经启用。虚拟化扩展可能被标记为虚拟化扩展、 V anderpool 或其他名称，具体取决于 OEM 和系统 BIOS。

c. 如果选项可用，启用 Intel VT-d 或 AMD IOMMU。Intel VT-d 和 AMD IOMMU 用于 PCI 设备分配。

d. 选择 **Save & Exit**。

3. 重启机器。

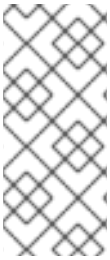
4. 引导计算机后，运行 **grep -E "vmx|svm" /proc/cpuinfo**。指定 **--color** 是可选的，但如果您想要突出显示搜索词，则很有用。如果命令输出，虚拟化扩展现已启用。如果没有输出您的系统可能没有启用虚拟化扩展或正确的 **BIOS** 设置。

A.10. 在 RED HAT ENTERPRISE LINUX 7 主机上关闭 RED HAT ENTERPRISE LINUX 6 虚拟机

使用 **Minimal** 安装选项安装 **Red Hat Enterprise Linux 6** 虚拟机不会安装 **acpid**（**acpi** 守护进程）。**Red Hat Enterprise Linux 7** 不再需要这个软件包，因为它被 **systemd** 接管。但是，在 **Red Hat Enterprise Linux 7** 主机上运行的 **Red Hat Enterprise Linux 6** 客户机虚拟机仍需要它。

如果没有 **acpid** 软件包，在执行 **virsh shutdown** 命令时，**Red Hat Enterprise Linux 6** 客户机虚拟机不会关闭。**virsh shutdown** 命令旨在正常关闭 **guest** 虚拟机。

使用 **virsh shutdown** 命令更容易且更安全地进行系统管理。如果没有通过 **virsh shutdown** 命令正常关闭，系统管理员必须手动登录到客户机虚拟机，或者向每个 **guest** 虚拟机发送 **Ctrl-Alt-Del** 组合键。



注意

其他虚拟化操作系统可能会受到此问题的影响。**virsh shutdown** 命令要求将客户机虚拟机操作系统配置为处理 **ACPI** 关闭请求。许多操作系统需要在客户端虚拟机操作系统中附加配置，以便接受 **ACPI** 关闭请求。

过程 **A.4. Red Hat Enterprise Linux 6** 客户端临时解决方案

1. 安装 **acpid** 软件包

   acpid 服务侦听和处理 **ACPI** 请求。

   登录到客户机虚拟机并在客户机虚拟机上安装 **acpid** 软件包：

```
# yum install acpid
```

2. **在客户端中启用 acpid 服务**

   将 acpid 服务设置为在客户机虚拟机启动序列中启动，并启动该服务：

```
# chkconfig acpid on
# service acpid start
```

3. **准备客户机域 XML**

   编辑域 XML 文件以包含下列元素：将 virtio serial 端口替换为 org.qemu.guest_agent.0，并使用您的客户机的名称而不是显示的端口。在本例中，guest 是 guest1。记得保存文件。

   图 A.1. 虚拟机 XML 替换

```
<channel type='unix'>
  <source mode='bind' path='/var/lib/libvirt/qemu/guest1.agent'/>
  <target type='virtio' name='org.qemu.guest_agent.0'/>
</channel>
```

4. **安装 QEMU 客户机代理**

   安装 QEMU 客户机代理(QEMU-GA)，并以 Red Hat Enterprise Linux 6 虚拟化管理指南 中的指示启动该服务。

5. **关闭客户端**

   a. **列出已知客户机虚拟机，以便可以检索您要关闭的名称。**

```
# virsh list --all
 Id Name            State
----------------------------------
 14 guest1          running
```

   b. **关闭客户机虚拟机。**

```
# virsh shutdown guest1

guest virtual machine guest1 is being shutdown
```

   c. **等待几秒钟，让 guest 虚拟机关闭。验证它是否已关闭。**

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

```
 Id Name                 State
----------------------------------
 14 guest1               shut off
```

d. 使用您编辑的 XML 文件，启动名为 guest1 的客户机虚拟机。

```
# virsh start guest1
```

e. 关闭 guest1 客户机虚拟机中的 acpi。

```
# virsh shutdown --mode acpi guest1
```

f. 再次列出所有 guest 虚拟机，guest1 仍应在列表中，并且应指明它已关闭。

```
#  virsh list --all
  Id Name                 State
----------------------------------
  14 guest1               shut off
```

g. 使用您编辑的 XML 文件，启动名为 guest1 的客户机虚拟机。

```
# virsh start guest1
```

h. 关闭 guest1 客户机虚拟机客户机代理。

```
# virsh shutdown --mode agent guest1
```

i. 列出客户机虚拟机。guest1 应该仍位于列表中，并且应指明它已关闭。

```
# virsh list --all
  Id Name                 State
----------------------------------
  guest1                 shut off
```

客户机虚拟机将使用 virsh shutdown 命令进行连续关闭，而无需使用上述临时解决方案。

除了上述方法外，还可以通过停止 libvirt-guests 服务来自动关闭 guest。有关这个方法的更多信息，请参阅 第 A.11 节 “允许 Graceful Shutdown 的可选临时解决方案”。

A.11. 允许 GRACEFUL SHUTDOWN 的可选临时解决方案

libvirt-guests 服务有参数设置，可以将其配置为确保客户机可以正常关闭。它是作为 libvirt 安装的一部分的软件包，默认安装。当主机关闭时，这个服务会自动将 guest 保存到磁盘，并在主机重启时将其恢复到其预下状态。默认情况下，此设置被设置为暂停客户机。如果您希望 guest 正常关闭，则需要更改 libvirt-guests 配置文件的其中一个参数。

过程 A.5. 更改 libvirt-guests 服务参数，以允许安全关闭 guest

此处描述的步骤允许在主机物理机器卡、关闭或需要重启时安全关闭客户机虚拟机。

1. 打开配置文件

   配置文件位于 /etc/sysconfig/libvirt-guests 中。编辑 文件，删除注释标记(#)，并将 ON_SHUTDOWN=suspend 更改为 ON_SHUTDOWN=shutdown。记得保存更改。

```
$ vi /etc/sysconfig/libvirt-guests

# URIs to check for running guests
# example: URIS='default xen:/// vbox+tcp://host/system lxc:///'
#URIS=default

# action taken on host boot
# - start   all guests which were running on shutdown are started on boot
#           regardless on their autostart settings
# - ignore  libvirt-guests init script won't start any guest on boot, however,
#           guests marked as autostart will still be automatically started by
#           libvirtd
#ON_BOOT=start

# Number of seconds to wait between each guest start. Set to 0 to allow
# parallel startup.
#START_DELAY=0

# action taken on host shutdown
# - suspend   all running guests are suspended using virsh managedsave
# - shutdown  all running guests are asked to shutdown. Please be careful with
#             this settings since there is no way to distinguish between a
#             guest which is stuck or ignores shutdown requests and a guest
#             which just needs a long time to shutdown. When setting
#             ON_SHUTDOWN=shutdown, you must also set SHUTDOWN_TIMEOUT to a
#             value suitable for your guests.
ON_SHUTDOWN=shutdown

# If set to non-zero, shutdown will suspend guests concurrently. Number of
# guests on shutdown at any time will not exceed number set in this variable.
#PARALLEL_SHUTDOWN=0
```

```
# Number of seconds we're willing to wait for a guest to shut down. If parallel
# shutdown is enabled, this timeout applies as a timeout for shutting down all
# guests on a single URI defined in the variable URIS. If this is 0, then there
# is no time out (use with caution, as guests might not respond to a shutdown
# request). The default value is 300 seconds (5 minutes).
#SHUTDOWN_TIMEOUT=300

# If non-zero, try to bypass the file system cache when saving and
# restoring guests, even though this may give slower operation for
# some file systems.
#BYPASS_CACHE=0
```

???

URIS - checks the specified connections for a running guest. The Default setting functions in the same manner as virsh does when no explicit URI is set In addition, one can explicitly set the URI from /etc/libvirt/libvirt.conf. Note that when using the libvirt configuration file default setting, no probing will be used.

???

ON_BOOT - specifies the action to be done to / on the guests when the host boots. The start option starts all guests that were running prior to shutdown regardless on their autostart settings. The ignore option will not start the formally running guest on boot, however, any guest marked as autostart will still be automatically started by libvirtd.

???

The START_DELAY - sets a delay interval in between starting up the guests. This time period is set in seconds. Use the 0 time setting to make sure there is no delay and that all guests are started simultaneously.

???

ON_SHUTDOWN - specifies the action taken when a host shuts down. Options that can be set include: suspend which suspends all running guests using virsh managedsave and shutdown which shuts down all running guests. It is best to be careful with using the shutdown option as there is no way to distinguish between a guest which is stuck or ignores shutdown requests and a guest that just needs a longer time to shutdown. When setting the ON_SHUTDOWN=shutdown, you must also set SHUTDOWN_TIMEOUT to a value suitable for the guests.

???

PARALLEL_SHUTDOWN Dictates that the number of guests on shutdown at any time will not exceed number set in this variable and the guests will be

suspended concurrently. If set to 0, then guests are not shutdown concurrently.

???

Number of seconds to wait for a guest to shut down. If SHUTDOWN_TIMEOUT is enabled, this timeout applies as a timeout for shutting down all guests on a single URI defined in the variable URIS. If SHUTDOWN_TIMEOUT is set to 0, then there is no timeout (use with caution, as guests might not respond to a shutdown request). The default value is 300 seconds (5 minutes).

???

BYPASS_CACHE can have 2 values, 0 to disable and 1 to enable. If enabled it will by-pass the file system cache when guests are restored. Note that setting this may effect performance and may cause slower operation for some file systems.

2. 启动 libvirt-guests 服务

如果您还没有启动该服务，请启动 libvirt-guests 服务。不要重启该服务，因为这将导致所有正在运行的 guest 虚拟机关闭。

A.12. KVM 网络性能

默认情况下，为 KVM 虚拟机分配一个虚拟 Realtek 8139(rtl8139)NIC(network interface controller)。

rtl8139 虚拟化 NIC 在大多数环境中正常工作，但这个设备可能会因某些网络（如 10 Gigabit Ethernet）上的性能下降。

要提高性能，您可以切换到半虚拟网络驱动程序。

注意

请注意，虚拟化 Intel PRO/1000(e1000)驱动程序也支持作为仿真驱动程序选择。要使用 e1000 驱动程序，请将以下流程中的 virtio 替换为 e1000。为了获得最佳性能，建议使用 virtio 驱动程序。

过程 A.6. 切换到 virtio 驱动程序

1. 关闭客户端操作系统。

2. 使用 virsh 命令编辑 guest 的配置文件（其中 GUEST 是 guest 的名称）：

```
# virsh edit GUEST
```

virsh edit 命令使用 $EDITOR shell 变量来确定要使用哪些编辑器。

3. 找到配置的网络接口部分。本节类似以下片段：

```
<interface type='network'>
  [output truncated]
  <model type='rtl8139' />
</interface>
```

4. 将 model 元素的 type 属性从 'rtl8139' 更改为 'virtio'。这会将 rtl8139 驱动程序中的驱动程序改为 virtio 驱动程序。

```
<interface type='network'>
  [output truncated]
  <model type='virtio' />
</interface>
```

5. 保存更改并退出文本编辑器

6. 重启客户端操作系统。

使用其他网络驱动程序创建新客户端

或者，也可以使用其他网络驱动程序创建新 guest。如果您在通过网络连接安装 guest 时，可能需要这样做。这个方法要求您至少有一个 guest 已创建（可能通过 CD 或者 DVD 安装）以用作模板。

1. 从现有 guest（在本示例中，名为 Guest1）创建 XML 模板：

```
# virsh dumpxml Guest1 > /tmp/guest-template.xml
```

2.

复制并编辑 XML 文件并更新唯一字段：虚拟机名称、UUID、磁盘镜像、MAC 地址以及任何其他唯一参数。请注意，您可以删除 UUID 和 MAC 地址行，virsh 将生成一个 UUID 和 MAC 地址。

```
# cp /tmp/guest-template.xml /tmp/new-guest.xml
# vi /tmp/new-guest.xml
```

在网络接口部分添加型号行：

```
<interface type='network'>
 [output truncated]
 <model type='virtio' />
</interface>
```

3. 创建新虚拟机：

```
# virsh define /tmp/new-guest.xml
# virsh start new-guest
```

A.13. 使用 LIBVIRT 创建外部快照的临时解决方案

KVM 客户机有两类快照：

- 内部快照 完全包含在 qcow2 文件中，并且由 libvirt 完全支持，允许创建、删除和恢复快照。这是 libvirt 在创建快照时使用的默认设置，特别是未指定选项时。此文件类型比其他文件创建快照的时间稍长，并且存在需要 qcow2 磁盘的缺陷。

重要

内部快照不会被主动开发，红帽不建议使用它们。

- 外部快照 可用于任何类型的原始磁盘映像，无需客户机停机时间，而且更稳定可靠。因此，建议在 KVM 客户机虚拟机中使用外部快照。但是，目前没有在 Red Hat Enterprise Linux 7 上完全实现外部快照，在使用 virt-manager 时无法使用。

要创建外部快照，请使用 snapshot-create-as 和 --diskspec vda,snapshot=external 选项，或者在快照 XML 文件中使用以下 磁盘 行：

```
<disk name='vda' snapshot='external'>
  <source file='/path/to,new'/>
</disk>
```

目前，外部快照是一个单向操作，因为 libvirt 可以创建它们，但无法做任何操作。在 libvirt 上游页面中 描述了一个临时解决方案。

A.14. 在带有日语键盘的客户机控制台中缺少字符

在 Red Hat Enterprise Linux 7 主机上，将日语键盘连接到计算机，可能会导致在 guest 控制台中正确显示下划线（_ 字符）等字符。这是因为默认没有正确设置所需的 keymap。

由于红帽企业 Linux 6 和红帽企业 Linux 7 虚拟机，按相关密钥通常不会产生错误消息。但是，Red Hat Enterprise Linux 4 和 Red Hat Enterprise Linux 5 客户端可能会显示类似如下的错误：

```
atkdb.c: Unknown key pressed (translated set 2, code 0x0 on isa0060/serio0).
atkbd.c: Use 'setkeycodes 00 <keycode>' to make it known.
```

要在 virt-manager 中修复这个问题，请执行以下步骤：

- 在 virt-manager 中打开受影响的 guest。
- 单击 View → Details。
- 从列表中选择 Display VNC。
- 在" 密钥 映射 "下拉菜单中将 Auto 更改为 ja。
- 点 Apply 按钮。

或者，在目标客户端中使用 virsh edit 命令解决了这个问题：

- 运行 virsh edit guestname

- 将以下属性添加到 <graphics> tag: keymap='ja' 中。例如：

```
<graphics type='vnc' port='-1' autoport='yes' keymap='ja'/>
```

A.15. 虚拟机故障切换至关闭

通常，执行 virsh shutdown 命令会导致发送电源按钮 ACPI 事件，因此像在物理机上按电源按钮时复制相同的操作。在每个物理机器中，它取决于操作系统来处理此事件。在过去的操作系统中，只需静默地关闭。今天，最常见的操作是显示一个对话框，询问应该执行的操作。有些操作系统甚至完全忽略此事件，特别是在没有用户登录时。当在客户机虚拟机中安装此类操作系统时，运行 virsh shutdown just no work（它会被忽略，或者在虚拟显示中显示对话框）。但是，如果将 qemu-guest-agent 频道添加到客户机虚拟机虚拟机，且此代理在客户机虚拟机操作系统中运行，virsh shutdown 命令将要求代理关闭客户端操作系统，而不是发送 ACPI 事件。该代理将从客户机虚拟机操作系统内部调用关机，一切都可以正常工作。

过程 A.7. 在客户机虚拟机中配置客户机代理频道

1. 停止 guest 虚拟机。

2. 为客户机虚拟机打开 Domain XML，并添加以下内容：

图 A.2. 配置客户机代理频道

```
<channel type='unix'>
   <source mode='bind'/>
   <target type='virtio' name='org.qemu.guest_agent.0'/>
</channel>
```

3. 通过运行 virsh start [domain] 启动 guest 虚拟机。

4. 在客户机虚拟机(yum install qemu-guest-agent)上安装 qemu-guest-agent，并使其在每次引导时自动运行(qemu-guest-agent.service)。

A.16. 为客户机虚拟机禁用 SMART DISK MONITORING

SMART 磁盘监控可以作为虚拟磁盘安全禁用，物理存储设备则由主机物理计算机管理。

```
# service smartd stop
# systemctl --del smartd
```

A.17. LIBGUESTFS 故障排除

有测试工具可用于检查 libguestfs 是否正常工作。在安装 libguestfs（不需要 root 访问权限）后输入以下命令来测试正常操作：

```
$ libguestfs-test-tool
```

此工具打印大量文本，用于测试 libguestfs 的操作。如果测试成功，则输出末尾附近会出现以下文本：

```
===== TEST FINISHED OK =====
```

A.18. SR-IOV 故障排除

本节包含可能影响 SR-IOV 的问题的解决方案。如果您需要其他帮助，请参阅 第 16.2.4 节 “从 SR-IOV 虚拟功能池设置 PCI 设备分配”。

启动客户机时出错

启动配置的虚拟机时，会出错：

```
# virsh start test
error: Failed to start domain test
error: Requested operation is not valid: PCI device 0000:03:10.1 is in use by domain rhel7
```

此错误通常是由已经分配给另一个虚拟客户机或主机本身的设备所致。

迁移、保存或转储客户机时出错

尝试迁移并转储虚拟机会导致类似如下的错误：

```
# virsh dump rhel7/tmp/rhel7.dump

error: Failed to core dump domain rhel7 to /tmp/rhel7.dump
error: internal error: unable to execute QEMU command 'migrate': State blocked by non-migratable device '0000:00:03.0/vfio-pci'
```

由于设备分配使用虚拟机启动的特定主机上的硬件，因此在使用设备分配时，不支持客户机迁移和保存。目前，同样的限制也适用于 core-dumping a guest；这可能会在以后有所变化。请注意，除非指定了 --memory-only 选项，否则 QEMU 目前不支持在附加了 PCI 设备的虚拟客户机上迁移、保存和转储操作。目前，它只支持使用 USB 设备的这些操作。目前正在做的工作，以便在未来进行这个改进。

A.19. 常见 LIBVIRT 错误和故障排除

本附录记录了常见的 libvirt- 相关问题和错误，以及处理它们的说明。

在下面的表格中找到错误，并按照答案中的相应链接以获取详细故障排除信息。

表 A.1. 常见 libvirt 错误

Error	问题描述	解决方案
libvirtd 无法启动	libvirt 守护进程无法启动。但是，**/var/log/messages** 中没有有关此错误的信息。	第 A.19.1 节 “**libvirtd** 无法启动”
无法读取 CA 证书	这是 URI 无法连接到管理程序时出现的几个错误之一。	第 A.19.2 节 “到虚拟机监控程序的 URI 失败”
其他连接错误	这些是 URI 无法连接管理程序时出现的其他错误。	第 A.19.2 节 “到虚拟机监控程序的 URI 失败”
客户端中的 PXE 引导（或 DHCP）失败	客户机虚拟机可以成功启动，但无法从 DHCP 获取 IP 地址，或使用 PXE 协议引导。这通常是由于为网桥设置的较长的转发延迟时间，或者当 iptables 软件包和内核不支持 checksum mangling 规则时。	第 A.19.3 节 “客户机失败的 PXE 引导（或 DHCP）”
客户机可以访问外部网络，但在使用 macvtap 接口时无法访问主机	客户机可以与其他虚拟客户机通信，但在配置为使用 macvtap（或 ***type='direct'***）网络接口后无法连接到主机。 这实际上不是错误 - 它是 macvtap 的定义的行为。	第 A.19.4 节 “Guest Can Reach Outside Network，但在使用 macvtap 界面时可能会重新访问主机”
无法添加规则来修复网络 *"默认"*上的 DHCP 响应校验和	此警告消息几乎总是没有损害，但通常被误认为是问题证。	第 A.19.5 节 “无法添加规则来修复网络 *"默认"*上的 DHCP 响应校验和”

Error	问题描述	解决方案
无法添加网桥 br0 端口 vnet0：没有这样的设备	此错误消息或类似的 **Failed 将 tap 接口添加到网桥 *'br0'*：没有这样的设备** 显示客户机（或域） **<interface>** 定义中指定的网桥设备不存在。	第 A.19.6 节 “无法添加网桥 br0 端口 vnet0：没有这样的设备”
无法解析地址 *name_of_host* 服务 '49155':未知的名称或服务	QEMU 虚拟客户机迁移失败，且显示此错误消息并显示不熟悉的主机名。	第 A.19.7 节 “migration Fails with **error: unables to address**”
无法允许访问磁盘路径 /var/lib/libvirt/images/qemu.img：没有这些文件或目录	无法迁移客户机虚拟机，因为 libvirt 无法访问磁盘映像。	第 A.19.8 节 “带有 **Unables 的 migration Fails 以允许访问磁盘路径：没有这些文件或目录**”
当 libvirtd 启动时没有客户机虚拟机	**libvirt** 守护进程成功启动，但在运行 **virsh list --all** 时不会显示客户机虚拟机。	第 A.19.9 节 “当 **libvirtd** 已启动时，没有演示客户机虚拟机”
常见 XML 错误	**libvirt** 使用 XML 文档来存储结构化的数据。当 XML 文档通过 API 传递至 **libvirt** 时，会会出现几个常见错误。这个条目提供了编辑客户机 XML 定义的说明，以及 XML 语法和配置中的常见错误。	第 A.19.10 节 “常见 XML 错误”

A.19.1. libvirtd 无法启动

症状

libvirt 守护进程不会自动启动。手动启动 libvirt 守护进程也失败：

```
# systemctl start libvirtd.service
* Caching service dependencies ...                                [ ok ]
* Starting libvirtd ...
/usr/sbin/libvirtd: error: Unable to initialize network sockets. Check /var/log/messages or run
without --daemon for more info.
* start-stop-daemon: failed to start `/usr/sbin/libvirtd'        [ !! ]
* ERROR: libvirtd failed to start
```

此外，/var/log/messages 中没有有关此错误的更多信息。

正在调查

通过启用下面的行，更改 libvirt 在 /etc/libvirt/libvirtd.conf 中的登录。要启用设置行，请在文本编辑器中打开 /etc/libvirt/libvirtd.conf 文件，从以下行的开头删除 hash（或 #）符号，并保存更

改：

```
log_outputs="3:syslog:libvirtd"
```

注意

默认情况下，该行会被注释掉，以防止 libvirt 生成过量日志消息。诊断问题后，建议在 /etc/libvirt/libvirtd.conf 文件中再次为该行添加注释。

重新启动 libvirt 以确定是否解决了该问题。

如果 libvirtd 仍然没有成功启动，则会输出类似以下内容的错误：

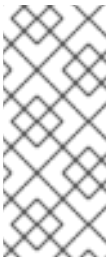
```
# systemctl restart libvirtd
Job for libvirtd.service failed because the control process exited with error code. See "systemctl
status libvirtd.service" and "journalctl -xe" for details.

Sep 19 16:06:02 jsrh libvirtd[30708]: 2017-09-19 14:06:02.097+0000: 30708: info : libvirt version:
3.7.0, package: 1.el7 (Unknown, 2017-09-06-09:01:55, js
Sep 19 16:06:02 jsrh libvirtd[30708]: 2017-09-19 14:06:02.097+0000: 30708: info : hostname: jsrh
Sep 19 16:06:02 jsrh libvirtd[30708]: 2017-09-19 14:06:02.097+0000: 30708: error :
daemonSetupNetworking:502 : unsupported configuration: No server certif
Sep 19 16:06:02 jsrh systemd[1]: libvirtd.service: main process exited, code=exited,
status=6/NOTCONFIGURED
Sep 19 16:06:02 jsrh systemd[1]: Failed to start Virtualization daemon.

-- Subject: Unit libvirtd.service has failed
-- Defined-By: systemd
-- Support: http://lists.freedesktop.org/mailman/listinfo/systemd-devel
--
-- Unit libvirtd.service has failed.
--
-- The result is failed.
```

libvirtd man page 显示，当 libvirt 在 TCP/IP 连接 模式下运行时，缺少的 cacert.pem 文件将用作 TLS 颁发机构。这表示将传递 --listen 参数。

解决方案

使用以下方法之一配置 libvirt 守护进程的设置：

- 安装 CA 证书。



注意

有关 CA 证书和配置系统身份验证的更多信息，请参阅 Red Hat Enterprise Linux 7 域身份、身份验证和策略指南中的管理证书和证书颁发机构章节。

- 不要使用 TLS；改为使用裸机 TCP。在 /etc/libvirt/libvirtd.conf 中设置 listen_tls = 0 和 listen_tcp = 1。默认值为 listen_tls = 1，listen_tcp = 0。

- 不要传递 --listen 参数。在 /etc/sysconfig/libvirtd.conf 中，更改 LIBVIRTD_ARGS 变量。

A.19.2. 到虚拟机监控程序的 URI 失败

连接到服务器时可能会出现几个不同的错误（例如，在运行 virsh 时）。

A.19.2.1. 无法读取 CA 证书

症状

在运行命令时，会出现以下错误（或类似）：

```
$ virsh -c qemu://$hostname/system_list
error: failed to connect to the hypervisor
error: Cannot read CA certificate '/etc/pki/CA/cacert.pem': No such file or directory
```

正在调查

错误消息是关于实际原因的误导。此错误可能是由各种因素造成的，如指定的 URI 或未配置的连接。

解决方案

不正确的指定 URI

当将 qemu://system 或 qemu://session 指定为连接 URI 时，virsh 尝试分别连接到主机名的 系统或 会话。这是因为 virsh 识别第二个正斜杠之后的文本作为主机。

使用三个正斜杠连接到本地主机。例如，指定 qemu:///system 会指示 virsh 连接到本地主机上 libvirtd 的系统 实例。

当指定主机名时，QEMU 传输默认为 TLS。这会导致证书。

没有配置连接

URI 正确（例如 qemu[+tls]://server/system），但不会在您的机器上正确设置证书。有关配置 TLS 的详情，请查看 上游 libvirt 网站。

A.19.2.2. 无法通过 'host:16509' 连接到服务器：连接已拒绝

症状

虽然 libvirtd 应该侦听连接的 TCP 端口，但连接会失败：

```
# virsh -c qemu+tcp://host/system
error: failed to connect to the hypervisor
error: unable to connect to server at 'host:16509': Connection refused
```

即使在更改 /etc/ libvirt /libvirtd.conf 中的配置后，libvirt 守护进程不会在 TCP 端口上侦听 TCP 端口：

```
# grep listen_ /etc/libvirt/libvirtd.conf
listen_tls = 1
listen_tcp = 1
listen_addr = "0.0.0.0"
```

但是，在更改配置后，libvirt 的 TCP 端口仍会打开：

```
# netstat -lntp | grep libvirtd
#
```

正在调查

libvirt 守护进程在没有 --listen 选项的情况下启动。运行以下命令验证：

```
# ps aux | grep libvirtd
root     10749  0.1  0.2 558276 18280 ?        Ssl  23:21   0:00 /usr/sbin/libvirtd
```

输出中不包含 --listen 选项。

解决方案

使用 **--listen** 选项启动守护进程。

要做到这一点，修改 **/etc/sysconfig/libvirtd** 文件并取消注释以下行：

```
# LIBVIRTD_ARGS="--listen"
```

然后，使用以下命令重启 **libvirtd** 服务：

```
# /bin/systemctl restart libvirtd.service
```

A.19.2.3. 身份验证失败

症状

在运行命令时，会出现以下错误（或类似）：

```
$ virsh -c qemu://$hostname/system_list
error: failed to connect to the hypervisor
error: authentication failed: authentication failed
```

正在调查

如果即使在使用了正确的凭证时也无法身份验证，则可能不会配置 **SASL** 身份验证。

解决方案

1. 编辑 **/etc/libvirt/libvirtd.conf** 文件，并将 **auth_tcp** 参数的值设置为 **sasl**。验证：

```
# cat /etc/libvirt/libvirtd.conf | grep auth_tcp
auth_tcp = "sasl"
```

2. 编辑 **/etc/sasl2/libvirt.conf** 文件，并将以下几行添加到文件中：

```
mech_list: digest-md5
sasldb_path: /etc/libvirt/passwd.db
```

3. 确定安装了 cyrus-sasl-md5 软件包：

```
# yum install cyrus-sasl-md5
```

4. 重启 libvirtd 服务：

```
# systemctl restart libvirtd
```

5. 为 libvirt SASL 设置用户名和密码：

```
# saslpasswd2 -a libvirt 1
```

A.19.2.4. 权限已拒绝

症状

当以非 root 用户身份运行 virsh 命令时，会出现以下错误（或类似）：

```
$ virsh -c qemu://$hostname/system_list
error: Failed to connect socket to '/var/run/libvirt/libvirt-sock': Permission denied
error: failed to connect to the hypervisor
```

解决方案

1. 编辑 /etc/libvirt/libvirt.conf 文件，并将以下几行添加到文件中：

```
#unix_sock_group = "libvirt"
#unix_sock_ro_perms = "0777"
#unix_sock_rw_perms = "0770"
```

2. 重启 libvirtd 服务：

```
# systemctl restart libvirtd
```

A.19.3. 客户机失败的 PXE 引导（或 DHCP）

症状

客户机虚拟机可以成功启动，但随后无法从 DHCP 获取 IP 地址或使用 PXE 协议引导。此错误有两个常见原因：为网桥设置了很长时间，当 iptables 软件包和内核不支持 checksum mangling 规则时。

网桥上的 forward 延迟时间

正在调查

这是此错误的最常见原因。如果客户机网络接口连接到启用了 STP（Spanning Tree 协议）的桥接设备，并且设置了长的延时，则网桥将不会将网络数据包从客户机虚拟机转发到网桥，直到在客户机连接到网桥以来至少经过转发延迟秒数。此延迟允许网桥时间监控来自接口的流量，并确定其后面的 MAC 地址，并阻止网络拓扑中的转发循环。

如果转发延迟比客户机的 PXE 或 DHCP 客户端的超时时间更长，客户端的操作将失败，并且客户机将无法引导（如果是 PXE），或者无法获取 IP 地址（如果是 DHCP）。

解决方案

如果是这种情况，请将网桥的转发延迟更改为 0，在网桥中禁用 STP，或者同时更改这两者。

注意

只有网桥不使用多个网络来连接多个网络，但只是将多个端点连接到一个网络（由 libvirt使用的网桥，最常见的用例）才适用。

如果客户机具有连接到 libvirt- 管理的虚拟网络的接口，请编辑网络的定义，然后重新启动它。例如，使用以下命令编辑默认网络：

```
# virsh net-edit default
```

在 <bridge> 元素中添加以下属性：

```
<name_of_bridge='virbr0' delay='0' stp='on'/>
```

注意

delay='0' 和 stp='on' 是虚拟网络的默认设置，因此仅当从默认修改配置时才需要这个步骤。

如果客户机接口连接到在 libvirt 之外配置的主机网桥，请更改延迟设置。

在 /etc/sysconfig/network-scripts/ifcfg-name_of_bridge 文件中添加或编辑以下行，以使用 0 秒打开 STP：

```
STP=on DELAY=0
```

更改配置文件后，重启网桥设备：

```
/usr/sbin/ifdown name_of_bridge
/usr/sbin/ifup name_of_bridge
```

注意

如果 name_of_bridge 不是网络中的 root 网桥，则该网桥的延迟最终会重置为 root 网桥的延时。要防止这种情况的发生，请在 name_of_bridge 上禁用 STP。

iptables 软件包和内核不支持 checksum mangling 规则

正在调查

只有当所有四个条件都为 true 时，这个信息才会出现问题：

- 客户机使用 virtio 网络设备。

  如果是这样，配置文件将包含 model type='virtio'

- 主机已加载 vhost-net 模块。

  如果 ls /dev/vhost-net 没有返回空结果，则会出现这种情况。

- 客户机尝试从主机上直接运行的 DHCP 服务器获取 IP 地址。

- 主机上的 iptables 版本早于 1.4.10。

  iptables 1.4.10 是第一个版本，用来添加 libxt_CHECKSUM 扩展。如果 libvirtd 日志中出现以下信息，会出现这种情况：

  ```
  warning: Could not add rule to fixup DHCP response checksums on network default
  warning: May need to update iptables package and kernel to support CHECKSUM rule.
  ```

  重要

  除非此列表中的所有其他条件也是 true，否则上述警告消息可以忽略，并且不是任何其他问题的指示符。

当出现这些条件时，从主机发送到客户机的 UDP 数据包具有取消计算的校验和。这使得主机的 UDP 数据包在客户机的网络堆栈看起来无效。

解决方案

要解决这个问题，使上述任何四点无效。最佳解决方案是将主机 iptables 和内核更新至 iptables-1.4.10 或更新版本。否则，最具体的修复是禁用这个特定虚拟机的 vhost-net 驱动程序。要做到这一点，使用以下命令编辑客户机配置：

```
virsh edit name_of_guest
```

在 <driver> 部分更改或添加 <interface> 行：

```
<interface type='network'>
  <model type='virtio'/>
  <driver name='qemu'/>
  ...
</interface>
```

保存更改，关闭 guest，然后重新启动它。

如果问题仍未解决，则问题可能是 firewalld 和默认的 libvirt 网络之间的冲突。

要解决这个问题，使用 service firewalld stop 命令停止 firewalld，然后使用 服务 libvirtd restart 命令重启 libvirt。

注意

另外，如果正确配置了 /etc/sysconfig/network-scripts/ifcfg-network_name 文件，您可以确保客户机使用 dhclient 命令作为 root 用户获取 IP 地址。

A.19.4. Guest Can Reach Outside Network，但在使用 macvtap 界面时可能会重新访问主机

症状

客户机虚拟机可以与其他虚拟客户机通信，但在配置为使用 macvtap（也称为 type='direct 的）网络接口后无法连接到主机。

正在调查

即使没有连接到虚拟以太网端口聚合器(VEPA)或 VN-Link 功能切换，但 macvtap 接口也很有用。将此类接口的模式设置为 网桥 可让客户机以非常简单的方式直接连接到物理网络，而无需设置问题（或 NetworkManager 不兼容），这些模式可以同时使用传统主机桥接设备。

但是，当 guest 虚拟机配置为使用 type='direct' 网络接口（如 macvtap）时，尽管能够与网络上的其他客户机和其他外部主机通信，但客户机无法与其自己的主机通信。

这种情况实际上不是 error - 它是 macvtap 的定义行为。由于主机物理以太网连接到 macvtap 网桥的方式，从客户机的流量传输到物理接口，无法退回到主机的 IP 堆栈。另外，发送到物理接口的主机 IP 堆栈的流量无法退回到 macvtap 网桥，以转发到客户机。

解决方案

使用 libvirt 创建隔离网络，并为连接到此网络的每个客户机虚拟机创建第二个接口。然后，主机和客户机可以通过这个隔离网络直接进行通信，同时还可保持与 NetworkManager 的兼容性。

过程 A.8. 使用 libvirt创建隔离网络

1. 在 /tmp/isolated.xml 文件中添加并保存以下 XML。如果 192.168.254.0/24 网络已在您的网络其他位置使用，您可以选择不同的网络。

图 A.3. 隔离网络 XML

```
...
<network>
  <name>isolated</name>
  <ip address='192.168.254.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.254.2' end='192.168.254.254'/>
    </dhcp>
  </ip>
</network>
...
```

2. 使用此选项创建网络： **virsh net-define /tmp/isolated.xml**

3. 使用 **virsh net-autostart 隔离** 命令将网络设置为 **autostart**。

4. 使用 **virsh net-start isolated** 命令启动网络。

5. 使用 **virsh edit name_of_guest**，编辑每个使用 **macvtap** 进行网络连接的客户机配置，并在 **<devices>** 部分中添加新接口，类似于以下内容（请注意 **<模型 type='virtio'/>** 行为 **include** 可选）： **<>**

图 A.4. 接口设备 XML

```
...
<interface type='network' trustGuestRxFilters='yes'>
  <source network='isolated'/>
  <model type='virtio'/>
</interface>
```

6. 关闭，然后重启每个虚拟机。

现在，客户机可以通过地址 **192.168.254.1** 访问主机，主机则能够通过从 **DHCP** 获取的 **IP** 地址（此外，您可以手动配置客户机的 **IP** 地址）。由于此新网络仅隔离到主机和客户机，来自客户机的所有其他通信都将使用 **macvtap** 界面。如需更多信息，请参阅 第 23.17.8 节 “网络接口”。

A.19.5. 无法添加规则来修复网络 "默认"上的 DHCP 响应校验和

症状

此时会出现这个信息：

> Could not add rule to fixup DHCP response checksums on network 'default'

正在调查

尽管此消息似乎是错误的证据，但几乎总是有危害。

解决方案

除非遇到的问题是客户机虚拟机无法通过 DHCP 获取 IP 地址，否则可以忽略此消息。

如果是这种情况，请参阅 第 A.19.3 节 “客户机失败的 PXE 引导（或 DHCP）” 以了解这种情况的详情。

A.19.6. 无法添加网桥 br0 端口 vnet0：没有这样的设备

症状

此时会出现以下出错信息：

> Unable to add bridge name_of_bridge port vnet0: No such device

例如，如果网桥名称是 br0，则错误消息会显示如下：

> Unable to add bridge br0 port vnet0: No such device

在 libvirt 版本 0.9.6 及更早版本中，会出现相同的错误：

> Failed to add tap interface to bridge name_of_bridge: No such device

或者，如果网桥命名为 br0 ：

> Failed to add tap interface to bridge 'br0': No such device

正在调查

两个错误消息都显示客户机（或域） <interface> 定义中指定的网桥设备不存在。

要验证错误消息中列出的网桥设备没有被存在，请使用 ip addr show br0。

这条消息通过该名称确认主机没有网桥：

```
br0: error fetching interface information: Device not found
```

如果是这样，请继续解决方案。

但是，如果生成的信息类似如下，这个问题会在其他位置存在：

```
br0       Link encap:Ethernet  HWaddr 00:00:5A:11:70:48
          inet addr:10.22.1.5  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:249841 errors:0 dropped:0 overruns:0 frame:0
          TX packets:281948 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
    RX bytes:106327234 (101.4 MiB)  TX bytes:21182634 (20.2 MiB)
```

解决方案

编辑现有网桥或使用 virsh 创建新桥接

使用 virsh 编辑现有网桥或网络的设置，或者将网桥设备添加到主机系统配置中。

使用 virsh 编辑现有网桥设置

使用 virsh edit name_of_guest 将 <interface> 定义更改为使用已存在的桥接或网络。

例如，将 type='bridge' 改为 type='network'，<source bridge='br0'/> 改为 <source network='default'/>。

使用 virsh 创建主机桥接

对于 libvirt 版本 0.9.8 及更高版本，可使用 virsh iface-bridge 命令创建网桥设备。这会创建一个使用 eth0 的桥接设备 br0，它是作为附加桥接的一部分设置的物理网络接口：

```
virsh iface-bridge eth0 br0
```

- 可选：如果需要，请删除此网桥，并使用这个命令恢复原始 eth0 配置：

```
virsh iface-unbridge br0
```

手动创建主机桥接

对于较旧版本的 libvirt，可以在主机上手动创建网桥设备。具体说明请查看 第 6.4.3 节 “使用 libvirt 进行桥接网络”。

A.19.7. migration Fails with error: unables to address

症状

QEMU 客户机代理失败，并显示这个错误消息：

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system
  error: Unable to resolve address name_of_host service '49155': Name or service not known
```

例如，如果目标主机名是 newyork，则错误消息会显示如下：

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system
error: Unable to resolve address 'newyork' service '49155': Name or service not known
```

但是，这个错误看上去有些奇怪，因为我们没有在任何位置使用 newyork 主机名。

正在调查

在迁移过程中，在目标主机上运行的 libvirtd 从地址和端口创建一个 URI，用于接收迁移数据并将其发回源主机上运行的 libvirtd。

在这种情况下，目标主机(192.168.122.12)的名称被设置为 'newyork'。出于某种原因，在该主机上运行的虚拟机的 libvirtd 无法将名称解析为可以返回并仍然有用的 IP 地址。因此，它会在解析名称时返回 "newyork "主机名希望源 libvirtd 更加成功。如果 DNS 没有被正确配置，或者 /etc/hosts 具有与本地回环地址(127.0.0.1)关联的主机名，则可能会出现此情况。

请注意，无法从连接到目标 libvirtd 的地址自动决定用于迁移数据的地址（例如，来自 qemu+tcp://192.168.122.12/system）。这是因为，要与目标 libvirtd 通信，源 libvirtd 可能需要使

用与 virsh （可能在独立计算机上运行）类型不同的网络基础架构。

解决方案

最佳解决方案是正确配置 DNS，以便涉及迁移的所有主机都可以解析所有主机名。

如果 DNS 无法配置为执行此操作，则可以手动将用于迁移的每个主机添加到每个主机上的 /etc/hosts 文件中。但是，在动态环境中让这类列表保持一致非常困难。

如果主机名不能被任何途径解析，virsh migrate 支持指定迁移主机：

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system tcp://192.168.122.12
```

目的地 libvirtd 将采用 tcp://192.168.122.12 URI，并附加自动生成的端口号。如果这不是理想的情况（例如，由于防火墙配置），可在此命令中指定端口号：

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system tcp://192.168.122.12:12345
```

另一种选择是使用隧道迁移。隧道的迁移不会为迁移数据创建单独的连接，而是通过用于连接目标 libvirtd 的连接（例如 qemu+tcp://192.168.122.12/system）：

```
# virsh migrate qemu qemu+tcp://192.168.122.12/system --p2p --tunnelled
```

A.19.8. 带有 Unables 的 migration Fails 以允许访问磁盘路径：没有这些文件或目录

症状

无法迁移客户机虚拟机（或域），因为 libvirt 无法访问磁盘映像：

```
# virsh migrate qemu qemu+tcp://name_of_host/system
error: Unable to allow access for disk path /var/lib/libvirt/images/qemu.img: No such file or
directory
```

例如，如果目标主机名是 newyork，则错误消息会显示如下：

```
# virsh migrate qemu qemu+tcp://newyork/system
error: Unable to allow access for disk path /var/lib/libvirt/images/qemu.img: No such file or
directory
```

正在调查

默认情况下，迁移仅传输正在运行的客户机的内存中状态（如内存或 CPU 状态）。虽然磁盘镜像不会在迁移期间传输，但它们都需要在这两个主机在同一个路径上保持访问。

解决方案

在两个主机上的同一位置设置并挂载共享存储。执行此操作的最简单方法是使用 NFS:

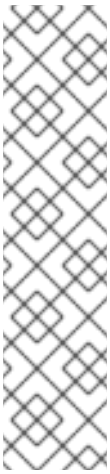
过程 A.9. 设置共享存储

1. 在主机上设置 NFS 服务器作为共享存储。NFS 服务器可以是迁移中涉及的主机之一，只要涉及的主机都通过 NFS 访问共享存储。

```
# mkdir -p /exports/images
# cat >>/etc/exports <<EOF
/exports/images    192.168.122.0/24(rw,no_root_squash)
EOF
```

2. 在运行 libvirt 的所有主机上，将导出的目录挂载到一个通用位置。例如，如果 NFS 服务器的 IP 地址为 192.168.122.1，使用以下命令挂载该目录：

```
# cat >>/etc/fstab <<EOF
192.168.122.1:/exports/images  /var/lib/libvirt/images  nfs  auto  0 0
EOF
# mount /var/lib/libvirt/images
```



注意

使用 NFS 从一个主机导出本地目录，并将它挂载到另一主机上的相同路径 - 用于存储磁盘镜像的目录必须从两个主机上的共享存储挂载。如果没有正确配置，客户机虚拟机在迁移过程中可能会丢失对磁盘映像的访问，因为源主机的 libvirt 守护进程可能会在成功将客户机成功迁移到其目的地后更改磁盘映像的所有者、权限和 SELinux 标签。

如果 libvirt 检测到磁盘镜像是从共享存储位置挂载的，则不会进行这些更改。

A.19.9. 当 libvirtd 已启动时，没有演示客户机虚拟机

症状

libvirt 守护进程成功启动，但没有显示客户机虚拟机。

```
# virsh list --all
 Id    Name                           State
----------------------------------------------------
```

正在调查

此问题有多种可能的原因。执行这些测试有助于确定此情况的原因：

验证 KVM 内核模块

确定在内核中插入 KVM 内核模块：

```
# lsmod | grep kvm
kvm_intel             121346  0
kvm                   328927  1 kvm_intel
```

如果您使用 AMD 机器，请验证在内核中使用类似命令 lsmod | grep kvm_amd 内核模块是否已插入到内核中。

如果模块不存在，请使用 modprobe <modulename> 命令插入它们。

注意

虽然这比较常见，但 KVM 虚拟化支持可能会被编译到内核中。在这种情况下，不需要模块。

验证虚拟化扩展

验证主机上是否支持并启用虚拟化扩展：

```
# egrep "(vmx|svm)" /proc/cpuinfo
flags  : fpu vme de pse tsc ... svm ... skinit wdt npt lbrv svm_lock nrip_save
flags  : fpu vme de pse tsc ... svm ... skinit wdt npt lbrv svm_lock nrip_save
```

在 BIOS 设置中启用硬件固件配置中的虚拟化扩展。详情请查看您的硬件文档。

验证客户端 URI 配置

验证客户端的 URI 是否已按预期配置：

```
# virsh uri
vbox:///system
```

例如，此消息显示 URI 已连接到 VirtualBox 管理程序，而不是 QEMU，并显示了设置为连接到 QEMU 管理程序的 URI 的配置错误。如果 URI 正确连接到 QEMU，则会显示相同的信息：

```
# virsh uri
qemu:///system
```

当存在其他虚拟机监控程序时，默认情况下 libvirt 可能进行通信时会出现这种情况。

解决方案

执行这些测试后，使用以下命令查看客户端虚拟机列表：

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

A.19.10. 常见 XML 错误

libvirt 工具使用 XML 文档来存储结构化的数据。当 XML 文档通过 API 传递至 libvirt 时，会出现各种常见错误。有几个常见 XML 错误 - 包括不正确的 XML 标签、不当的值以及缺失的元素 - 如下所示。

A.19.10.1. 编辑域定义

虽然不建议您这样做，但有时需要手动编辑 guest 虚拟机（或域） XML 文件。要访问客户机的 XML 进行编辑，请使用以下命令：

```
# virsh edit name_of_guest.xml
```

此命令会在文本编辑器中打开文件，其中包含 guest 虚拟机的当前定义。完成编辑并保存更改后，libvirt 会重新加载并解析 XML。如果 XML 正确，则会显示以下消息：

```
# virsh edit name_of_guest.xml

Domain name_of_guest.xml XML configuration edited.
```

重要

在 virsh 中使用 edit 命令来编辑 XML 文档时，请在退出编辑器前保存所有更改。

保存 XML 文件后，使用 xmllint 命令验证 XML 的格式是否正确，或者 virt-xml-validate 命令检查用量问题：

```
# xmllint --noout config.xml
```

```
# virt-xml-validate config.xml
```

如果没有返回错误，XML 描述将正确格式，并与 libvirt 模式匹配。虽然架构没有捕获所有限制，但修复报告的所有错误将会进一步故障排除。

libvirt存储的 XML 文档

这些文档包含客户机的状态和配置定义。这些文档会自动生成，不应手动编辑。这些文档中的错误包含损坏的文档的文件名。文件名仅在 URI 定义的主机上有效，这可能看到运行 命令的机器。

libvirt 创建的文件中的错误很少见。但是，这些错误的一个可能源是 libvirt 的降级问题，而较新版本的 libvirt 总是可以读取由较旧版本生成的 XML，而较旧版本的 libvirt 可能会与新版本中添加的 XML 元素混淆。

A.19.10.2. XML 语法错误

XML 解析器会发现语法错误。错误消息包含可识别问题的信息。

XML 解析器的错误消息包含三行 - 第一行表示错误消息，而两行则包含包含该错误的 XML 代码的上下文和位置。第三行包含一个指示器，显示它上面的错误：

```
error: (name_of_guest.xml):6: StartTag: invalid element name
<vcpu>2</vcpu><
-----------------^
```

此消息中包含的信息：

(name_of_guest.xml)

这是包含该错误的文档的文件名。括号中的文件名是用来描述从内存中解析的 XML 文档的符号链接，不直接对应于磁盘上的文件。不在括号中包含的文件名是驻留在连接目标的本地文件。

6

这是包含错误的 XML 文件中的行号。

StartTag: 无效的元素名称

这是 libxml2 parser 的错误消息，它描述了特定的 XML 错误。

A.19.10.2.1. 文档中的位置 <

症状

发生以下错误：

```
error: (name_of_guest.xml):6: StartTag: invalid element name
<vcpu>2</vcpu><
-----------------^
```

正在调查

这个错误消息显示，解析器需要在客户机 XML 文件的 6 行上 < 符号后有一个新的元素名称。

确保在文本编辑器中启用了行号显示。打开 XML 文件，并在第 6 行中找到文本：

```
<domain type='kvm'>
  <name>name_of_guest</name>
<memory>524288</memory>
<vcpu>2</vcpu><
```

此客户机 XML 文件的片段在文档中包含一个额外的 < ：

解决方案

删除额外的 < 或完成新元素。

A.19.10.2.2. Unterminated 属性

症状

发生以下错误：

```
error: (name_of_guest.xml):2: Unescaped '<' not allowed in attributes values
<name>name_of_guest</name>
--^
```

正在调查

该客户机 XML 文件中的这个片段包含一个未确定的元素属性值：

```
<domain type='kvm>
<name>name_of_guest</name>
```

在本例中，"kvm" 缺少第二个引号。属性值必须使用引号或 postrophes 打开和关闭，类似于 XML start 和 end 标签。

解决方案

正确打开和关闭所有属性值字符串。

A.19.10.2.3. 打开和结束的标签不匹配

症状

发生以下错误：

```
error: (name_of_guest.xml):61: Opening and ending tag mismatch: clock line 16 and domain
</domain>
---------^
```

正在调查

以上错误消息包含三个附加标签：

最后一个冒号（时钟行 16 和域）后面的消息显示 <clock> 在文档的第 16 行上包含一个不匹配的标签。最后一个提示是消息的上下文部分中的指针，用于标识第二个出错标签。

未对的标签必须使用 /> 关闭。以下片段没有遵循这个规则，并生成上述错误消息：

```
<domain type='kvm'>
  ...
    <clock offset='utc'>
```

这个错误是由文件中不匹配的 XML 标签造成的。每个 XML 标签都必须具有匹配的 start 和 end 标签。

不匹配 XML 标签的其他示例

以下示例生成类似的错误消息，并显示不匹配 XML 标签的变体。

此片段包含 <features> 不匹配的错误，因为没有结束标签(</name>)：

```
<domain type='kvm'>
 ...
 <features>
   <acpi/>
   <pae/>
 ...
 </domain>
```

此片段包含一个结尾标签(</name>)，没有对应的 start tag：

```
<domain type='kvm'>
  </name>
  ...
</domain>
```

解决方案

确保所有 XML 标签都正确启动和结束。

A.19.10.2.4. 标记中的排字错误

症状

此时会出现以下出错信息：

```
error: (name_of_guest.xml):1: Specification mandate value for attribute ty
<domain ty pe='kvm'>
-----------^
```

正在调查

XML 错误很容易由一个简单的拼写错误导致。此错误消息突出显示 XML 错误 - 在这种情况下，单词 type - 额外空格，带有指针。

```
<domain ty pe='kvm'>
```

由于拼写错误（如缺少特殊字符）或额外字符，这些 XML 示例无法正确解析：

```
<domain type 'kvm'>
```

```
<dom#ain type='kvm'>
```

解决方案

要识别有问题的标签，请读取文件上下文的错误消息，并找到与指针相关的错误。更正 XML 并保存更改。

A.19.10.3. 逻辑和配置错误

格式良好的 XML 文档可以包含语法正确但 libvirt 无法解析的错误。其中有很多错误，其中两个最常见的情况如下所示。

A.19.10.3.1. Vanishing 部分

症状

您所做的部分更改在编辑或定义域后不会显示任何影响。定义 或编辑命令 可以正常工作，但在再次转储 XML 时，更改将消失。

正在调查

libvirt 不解析的构造或语法中的这个错误可能产生。libvirt 工具通常仅查找其知道的构造内容，但忽略了其他内容，从而导致 libvirt 解析输入后的一些 XML 更改传播。

解决方案

在传递 编辑或 定义 命令之前验证 XML 输入。libvirt 开发人员维护与 libvirt 捆绑的一组 XML

模式，用于定义 libvirt 使用的 XML 文档中允许的大部分构造。

使用以下命令验证 libvirt XML 文件：

```
# virt-xml-validate libvirt.xml
```

如果这个命令通过，则 libvirt 可能会了解 XML 中的所有结构，除非架构无法检测仅对给定管理程序有效的选项。例如，libvirt 生成的任何 XML 作为 virsh dump 命令生成的任何 XML 应该验证且无错误。

A.19.10.3.2. 不正确的驱动器设备类型

症状

CD-ROM 虚拟驱动器的源镜像定义不存在，尽管添加了：

```
# virsh dumpxml domain
<domain type='kvm'>
  ...
  <disk type='block' device='cdrom'>
    <driver name='qemu' type='raw'/>
    <target dev='hdc' bus='ide'/>
    <readonly/>
  </disk>
  ...
</domain>
```

解决方案

通过添加缺少的 <source> 参数更正 XML：

```
<disk type='block' device='cdrom'>
  <driver name='qemu' type='raw'/>
  <source file='/path/to/image.iso'/>
  <target dev='hdc' bus='ide'/>
  <readonly/>
</disk>
```

type='block' 磁盘设备预期源是一个物理设备。要将磁盘与镜像文件搭配使用，使用 type='file' 替代。

附录 B. 在多个构架中使用 KVM 虚拟化

默认情况下，Red Hat Enterprise Linux 7 中的 KVM 虚拟化与 AMD64 和 Intel 64 构架兼容。但是，从 Red Hat Enterprise Linux 7.5 开始，KVM 虚拟化还在以下构架中被支持，因为引进了 kernel-alt 软件包：

- IBM POWER
- IBM Z
- ARM 系统 （不支持）

请注意，在这些构架中使用虚拟化时，安装、使用和功能支持因某些方面与 AMD64 和 Intel 64 不同。如需更多信息，请参阅以下部分：

B.1. 在 IBM POWER 系统中使用 KVM 虚拟化

从 Red Hat Enterprise Linux 7.5 开始，IBM POWER8 系统和 IBM POWER9 系统支持 KVM 虚拟化。但是 IBM POWER8 不使用 kernel-alt，这意味着这两个架构在某些方面有所不同。

安装

要在用于 IBM POWER 8 和 POWER9 系统的 Red Hat Enterprise Linux 7 中安装 KVM 虚拟化：

1. 从客户门户网站中的可引导镜像安装主机系统：
   - IBM POWER8
   - IBM POWER9

具体步骤请查看 Red Hat Enterprise Linux 7 安装指南。

2. 确定您的主机系统满足管理程序要求：

- 验证您具有正确的机器类型：

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

这个命令的输出必须包含 PowerNV 条目，这代表您在受支持的 PowerNV 机器类型中运行：

```
platform        : PowerNV
```

- 载入 KVM-HV 内核模块：

```
# modprobe kvm_hv
```

- 验证是否载入了 KVM-HV 内核模块：

```
# lsmod | grep kvm
```

如果成功载入了 KVM-HV，这个命令的输出会包括 kvm_hv。

3. 除了 第 2 章 安装虚拟化软件包 中描述的其他虚拟化软件包外，安装 qemu-kvm-ma 软件包。

架构特定

Red Hat Enterprise Linux 7.5 for IBM POWER 中的 KVM 虚拟化与 AMD64 和 Intel 64 系统的 KVM 不同：

- IBM POWER 主机上客户机的建议 最小内存分配 是 2GB RAM。

- IBM POWER 系统不支持 SPICE 协议。要显示客户机的图形输出，请使用 VNC 协议。另外，只支持以下虚拟 图形卡设备 ：

  - VGA - 仅在 -vga std 模式中支持，且不支持 -vga cirrus 模式
  - virtio-vga
  - virtio-gpu

- AMD64 和 Intel 64 主机上禁用了以下虚拟化功能，但可在 IBM POWER 上工作。但是，红帽不支持它们，因此不推荐使用：

  - I/O 线程

- SMBIOS 配置不可用。

- POWER8 虚拟机，包括兼容模式客户机，可能无法启动类似如下的错误：

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

  这在使用 Red Hat Enterprise Linux 7.3 或更早版本的客户端发生的可能性更大。

  要解决这个问题，通过将 kvm_cma_resv_ratio= 内存添加到主机的内核命令行（其中 memory 是 CMA 池保留的主机内存的百分比）来增加 CMA 内存池（默认为 5）。

- 透明巨页(THP)目前在 IBM POWER8 客户端中不提供任何显著的性能优势

  另请注意，IBM POWER8 系统中静态 巨页 的大小是 16MiB 和 16GiB，而 AMD64 和 Intel 64 和 Intel 64 和 IBM POWER9 上的 2MiB 和 1GiB 不同。因此，如果客户机配置了静态巨页，将客户机从 IBM POWER8 主机迁移到 IBM POWER9 主机会失败。

  另外，为了能够在 IBM POWER8 客户端中 使用静态巨页或 THP，您必须首先在 主机 中设置巨页。

- IBM POWER 系统中不支持在 AMD64 和 Intel 64 系统中支持很多虚拟 外围设备，或者支持不同的设备替换：

  - 不支持用于 PCI-E 层次结构的设备，包括 ioh3420 和 xio3130-downstream 设备。这个功能由 spapr-pci-host-bridge 设备提供的多个独立 PCI 根网桥替代。
  - 不支持 UHCI 和 EHCI PCI 控制器。使用 OHCI 和 XHCI 控制器。
  - 不支持 IDE 设备，包括虚拟 IDE CD-ROM(ide-cd)和虚拟 IDE 磁盘(ide-hd)。改为使用 virtio-scsi 和 virtio-blk 设备。
  - 不支持模拟 PCI NIC(rtl8139)。改为使用 virtio-net 设备。
  - 不支持声音设备，包括 intel-hda、hda-output 和 AC97。
  - 不支持 USB 重定向设备，包括 usb-redir 和 usb-tablet。

- kvm-clock 服务 不必为 IBM POWER 系统上的 时间管理 配置。

- IBM POWER 系统不支持 pvpanic 设备。但是，默认在该架构中提供了等同的功能并激活。要在客户端中启用它，请使用带有 保留 值的 <on_crash> 配置元素。另外，确保从

<devices> 部分删除 <panic> 元素，因为它存在可能会导致客户机无法在 IBM POWER 系统中引导。

- *在 IBM POWER8 系统中，主机机器必须以单线程模式运行，才能支持客户机。如果安装了 qemu-kvm-ma 软件包，则会自动进行配置。但是，在单线程主机上运行的虚拟机仍然可以使用多个线程。*

- *当在 RHEL 7 主机中运行的 IBM POWER 虚拟机时，配置了使用零内存(memory='0')的 NUMA 节点，虚拟机无法正常工作。因此，红帽不支持在 RHEL 7 中带有零内存 NUMA 节点的 IBM POWER 虚拟机*

B.2. 在 IBM Z 中使用 KVM 虚拟化

安装

在 IBM Z 主机上，KVM 管理程序需要安装在专用的逻辑分区(LPAR)中。不支持在 z/VM 操作系统中运行 KVM。LPAR 还必须支持 开始执行(SIE)虚拟化扩展。

在 Red Hat Enterprise Linux 7 for IBM Z 中安装 KVM 虚拟化：

1. *从 客户门户网站中的可引导镜像安装 主机系统 - 了解更多详细信息，请参阅 安装指南。*

2. *确定您的系统满足管理程序要求：*

   - *验证 CPU 虚拟化扩展是否可用：*

     ```
     # grep sie /proc/cpuinfo
     ```

     此命令的输出必须包含 sie 条目，这表示您的处理器具有所需的虚拟化扩展。

     ```
     features       : esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs te sie
     ```

   - *载入 KVM 内核模块：*

```
# modprobe kvm
```

- 验证是否载入了 KVM 内核模块：

```
# lsmod | grep kvm
```

如果成功载入 KVM，此命令的输出包括 kvm。如果没有，请确定您正在为 Red Hat Enterprise Linux 7 使用 kernel-alt 版本。

3. 除了 第 2 章 安装虚拟化软件包 中描述的其他虚拟化软件包外，安装 qemu-kvm-ma 软件包。

4. 在设置客户机时，建议以 以下一种方式配置其 CPU，以保护客户机不受 "Spectre" 漏洞的影响：

- 使用主机 CPU 模型，例如：

```
<cpu mode='host-model' check='partial'>
  <model fallback='allow'/>
</cpu>
```

这样，如果主机支持，则客户端可使用 ppa15 和 bpb 功能。

- 如果使用特定的主机模型，请添加 ppa15 和 bpb 功能。以下示例使用 zEC12 CPU 模型：

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

注意

使用带有 z114 和 z196 CPU 型号的 ppa15 功能时，请确保使用最新的 microcode 级别（bundle 95 或更高版本）。

架构特定

Red Hat Enterprise Linux 7.5 for IBM Z 上的 KVM 虚拟化与 AMD64 和 Intel 64 系统的 KVM 不同：

- *IBM Z 不支持 SPICE 和 VNC 协议，因此无法显示客户机图形输出。???*
- *IBM Z 不支持虚拟 PCI 和 USB 设备。这也意味着 virtio-*-pci 设备不受支持，应当使用 virtio-*-ccw 设备。例如，使用 virtio-net-ccw 而不是 virtio-net-pci。*
- *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 <devices> 部分中的 <boot order='number'/> 元素。例如，请参阅 上游 libvirt 文档。*

注意

在 AMD64 和 Intel 64 主机上使用 <boot order='number'/> 进行引导顺序管理。

- *SMBIOS 配置不可用。*
- *IBM Z 中使用的 watchdog 设备模型 应为 diag288。*
- *要在 IBM Z 上启用 嵌套虚拟化 功能，请执行以下操作：请注意，与 AMD64 和 Intel 64 系统中一样，嵌套虚拟化在 IBM Z 中作为技术预览提供，因此不建议在生产环境中使用。*

1.
检查是否在您的系统中启用了嵌套虚拟化：

```
$ cat /sys/module/kvm/parameters/nested
```

如果这个命令返回 1，则代表该功能已经启用。

如果命令返回 0， 请使用以下步骤启用它。

2.
卸载 kvm 模块：

```
# modprobe -r kvm
```

3.
激活嵌套功能：

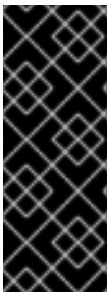
```
# modprobe kvm nested=1
```

4.
现在，嵌套功能只有在下次重启主机时才会启用。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
```

- kvm-clock 服务专用于 AMD64 和 Intel 64 系统，且不必为 IBM Z 上的时间管理 进行配置。

B.3. 在 ARM 系统上使用 KVM 虚拟化



重要

Red Hat Enterprise Linux 7.5 及更新的版本中提供了 KVM 虚拟化，用于 64 位 ARM 架构。因此，红帽不支持 ARM 系统上的 KVM 虚拟化，不适用于在生产环境中使用，且可能无法解决已知的安全漏洞。另外，由于 ARM 上的 KVM 虚拟化仍在快速开发中，因此以下信息无法保证准确或完整。

安装

在 Red Hat Enterprise Linux 7.5 for ARM 上使用安装虚拟化：

1. 从客户门户网站中的可引导镜像安装 主机系统。

2. 安装该系统后，使用以下命令在系统中安装虚拟化堆栈：

```
# yum install qemu-kvm-ma libvirt libvirt-client virt-install AAVMF
```

确保启用了 Optional 频道以便安装成功。

架构特定

对于 64 位 ARM 架构的 Red Hat Enterprise Linux 7.5 中的 KVM 虚拟化与 AMD64 和 Intel 64 系统的 KVM 不同：

-

PXE 引导只支持 virtio-net-device 和 virtio-net-pci 网络接口控制器(NIC)。另外，ARM 架构虚拟机固件(AAVMF)的内置 VirtioNetDxe 驱动程序需要用于 PXE 引导。请注意，不支持 iPXE 选项 ROM。

- *只能分配给一个虚拟机，最多 123 个虚拟 CPU(vCPU)。*

附录 C. 虚拟化限制

本附录涵盖了 Red Hat Enterprise Linux 7 中虚拟化软件包的额外支持和产品限制。

C.1. 系统限制

主机系统

只有在以下主机构架中才支持带 KVM 的 Red Hat Enterprise Linux：

- AMD64 和 Intel 64
- IBM Z
- IBM POWER8
- IBM POWER9

本文档主要论述 AMD64 和 Intel 64 功能和功能，但其他支持的构架非常相似。详情请查看 附录 B, 在多个构架中使用 KVM 虚拟化。

客户端系统

在 Red Hat Enterprise Linux 7 中，只有特定订阅程序（如 Advanced mission Critical）支持 Microsoft Windows 客户机虚拟机。如果您不确定您的订阅模式是否包括 Windows 客户机支持，请联系客户服务。

有关 Red Hat Enterprise Linux 7 中 Windows 客户机虚拟机的更多信息，请参阅 Red Hat Enterprise Linux 7 的 Windows 客户机虚拟机文章。

C.2. 功能限制

Red Hat Enterprise Linux 中包含的虚拟机监控程序软件包是 qemu-kvm。这与 Red Hat Virtualization(RHV)和 Red Hat OpenStack(RHOS)产品中包含的 qemu-kvm-rhev 软件包不同。应用到 qemu-kvm 的许多限制都不适用于 qemu-kvm-rhev。

有关 qemu-kvm 和 qemu-kvm -rhev 软件包之间的区别的更多信息，请参阅 qemu-kvm 和 qemu-kvm-rhev 和所有子包之间的区别？

以下限制适用于 Red Hat Enterprise Linux 中包含的 KVM 管理程序：

每个虚拟机的最大 vCPU

Red Hat Enterprise Linux 7.2 及更高版本支持每个客户机的 240 个 vCPU，Red Hat Enterprise Linux 7.0 中最多支持 160 个 vCPU。

嵌套虚拟化

在 Red Hat Enterprise Linux 7.2 及更新的版本中，嵌套虚拟化作为技术预览提供。这个功能可让 KVM 启动作为虚拟机监控程序并创建自己的客户机的客户机。

TCG 支持

QEMU 和 libvirt 包括使用 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。

当使用 qemu-kvm 软件包在虚拟机中创建嵌套虚拟客户机时，它使用 TCG，除非父虚拟机上启用了嵌套虚拟化。请注意，嵌套虚拟化目前还是一个技术预览功能。如需更多信息，请参阅 第 12 章 嵌套虚拟化。

可使用以下方法识别基于 TCG 的客户机：

- 客户机的域 XML 文件包含 <domain type='qemu'> 行，而 KVM 客户机包含 <domain type='kvm'>。
- 在 Virtual 硬件详情 视图的 Overview 窗格中，virt-manager 将虚拟机的类型显示为 QEMU TCG，而不是 KVM。

持续 TSC 位

没有 Constant 时间戳计数器(TSC)的系统需要额外的配置。有关确定您是否具有 Constant 时间戳计数器和配置步骤以修复相关问题的详细信息，请参阅 第 8 章 KVM 客户机计时管理。

模拟 SCSI 适配器

SCSI 设备模拟只支持 virtio-scsi 半虚拟化主机总线适配器(HBA)。Red Hat Enterprise Linux 中的 KVM 不支持模拟 SCSI HBA。

模拟 IDE 设备

KVM 限制为每个虚拟机最多四个虚拟化（模拟）IDE 设备。

半虚拟设备

半虚拟设备也称为 VirtIO 设备。它们是纯粹的虚拟设备，设计为在虚拟机中最佳工作。

Red Hat Enterprise Linux 7 每个虚拟机总线支持 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并且使用 PCI 网桥时，每个总线最多提供了 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，一些总线不会为用户提供所有 256 个设备地址；例如，根总线有几个内置设备占用的插槽。

有关 PCI 网桥的更多信息，请参阅 第 16 章 虚拟机设备配置 和 第 16.1.5 节 “PCI Bridges”。

迁移限制

设备分配指的是已公开给虚拟机的物理设备，以独占地使用该虚拟机。由于设备分配使用虚拟机运行的特定主机上的硬件，所以使用设备分配时不支持迁移和保存/恢复。如果客户机操作系统支持热插拔，可以在迁移或保存/恢复操作之前删除分配的设备，以启用此功能。

实时迁移只能在具有相同 CPU 类型的主机之间实现（即 Intel 到 Intel 或 AMD）。

对于实时迁移，两个主机都必须为 No eXecution(NX)位设置相同的值，可在 或 关闭。

要使迁移正常工作，必须为在写入模式下打开的所有块设备指定 cache=none。

警告

未能包括 cache=none 选项可能会导致磁盘崩溃。

存储限制

向 guest 虚拟机授予对整个磁盘或块设备（如 /dev/sdb）的写入权限存在风险。如果客户机虚拟机可以访问整个块设备，它可以与主机共享任何卷标签或分区表。如果主机系统的分区识别代码中存在错误，这可能会造成安全隐患。通过将主机机器配置为忽略分配给客户机虚拟机的设备，避免出现此风险。

警告

未能遵守存储限制会导致出现安全性风险。

实时快照

Red Hat Enterprise Linux 中的 KVM 中的备份和恢复 API 不支持实时快照。

流、镜像(mirror)和实时更新

不支持流、镜像(mirror)和 live-merge。这可防止 block-jobs。

I/O 节流

Red Hat Enterprise Linux 不支持为虚拟磁盘上的操作配置最大输入和输出等级。

I/O 线程

Red Hat Enterprise Linux 不支持为使用 VirtIO 接口的磁盘上输入和输出操作创建独立线程。

内存热插拔和热拔

Red Hat Enterprise Linux 不支持热插拔或热拔虚拟机的内存。

vhost-user

Red Hat Enterprise Linux 不支持实现用户空间 vhost 接口。

CPU 热拔

Red Hat Enterprise Linux 不支持从虚拟机热拔 CPU。

适用于 PCIe 的 NUMA 客户机位置

Red Hat Enterprise Linux 不支持将虚拟 PCIe 设备绑定到特定的 NUMA 节点。

内核转储限制

因为当前在迁移之上实施内核转储，所以使用设备分配时不支持它。

实时内核

KVM 目前不支持实时内核，因此无法在 Red Hat Enterprise Linux for Real Time 中使用。

C.3. 应用程序限制

虚拟化方面对某些类型的应用程序变得非常明显。

具有高 I/O 吞吐量要求的应用程序应该将 KVM 的泛虚拟化驱动程序（virtio 驱动程序）用于完全虚拟化的虚拟机。如果没有 virtio 驱动程序，某些应用程序可能会在重度 I/O 负载下预计。

由于 I/O 要求高，应避免以下应用程序：

- *Kdump 服务器*
- *netdump 服务器*

您应该仔细评估大量利用 I/O 或需要实时性能的应用程序和工具。考虑 virtio 驱动程序或 PCI 设备分配来提高 I/O 性能。有关完全虚拟化的客户机的 virtio 驱动程序的详情，请参考 第 5 章 KVM 半虚拟化 (virtio)驱动程序。有关 PCI 设备分配的详情请参考 第 16 章 虚拟机设备配置。

应用程序对虚拟环境中的运行性能较小。应根据使用虚拟化的相关潜在应用程序性能问题评估虚拟化到更新、更快的硬件的性能优势。

C.4. 其他限制

有关影响虚拟化的其他限制和问题的列表，请参阅 Red Hat Enterprise Linux 7 发行注记。Red Hat Enterprise Linux 7 发行注记 涵盖了存在新功能、已知问题和限制，因为它们被更新或发现。

C.5. 存储支持

虚拟机支持以下 存储方法 ：

- 本地存储中的文件
- 物理磁盘分区
- 本地连接的物理 LUN
- LVM 分区
- NFS 共享文件系统
- iSCSI
- GFS2 集群文件系统
- 基于 Fibre Channel 的 LUN

- 通过以太网的光纤(FCoE)

C.6. USB 3 / XHCI 支持

Red Hat Enterprise Linux 7.2 及更高版本中支持 USB 3(xHCI)USB 主机适配器模拟。所有 USB 速度都可用，这意味着任何生成 USB 设备都可以插入到 xHCI 总线中。另外，不需要相应的控制器（对于 USB 1 设备）。但请注意，不支持 USB 3 批量流。

xHCI 的优点：

- 由于轮询开销减少了，虚拟化兼容硬件设计，这意味着 xHCI 模拟需要比之前的版本要少。
- USB 透传 USB 3 设备可用。

xHCI 的限制：

- 不支持 Red Hat Enterprise Linux 5 guest。

有关 xHCI 设备的域 XML 设备示例，请参阅 图 16.19 “USB3/xHCI 设备的域 XML 示例”。

附录 D. 其它资源

要了解更多有关虚拟化和 Red Hat Enterprise Linux 的信息，请查看以下资源。

D.1. 在线资源

- http://www.libvirt.org/ 是 libvirt 虚拟化 API 的官方上游网站。
- https://virt-manager.org/ 是 虚拟机管理器 (virt-manager)的上游项目网站，用于管理虚拟机的图形应用程序。
- Red Hat Virtualization - http://www.redhat.com/products/cloud-computing/virtualization/
- 红帽产品文档 - https://access.redhat.com/documentation/en/
- 虚拟化技术概述 - http://virt.kernelnewbies.org

D.2. 安装的文档

- man virsh 和 /usr/share/doc/libvirt-version-number - 包含 virsh 虚拟机管理实用程序的子命令和选项以及 libvirt 虚拟化库 API 的综合信息。
- /usr/share/doc/gnome-applet-vm-version-number - 监控和管理本地运行中虚拟机的 GNOME 图形面板小程序的文档。
- /usr/share/doc/libvirt-python-version-number - 提供 libvirt 库的 Python 绑定详情。libvirt-python 软件包允许 python 开发人员创建与 libvirt 虚拟化管理库交互的程序。
- /usr/share/doc/virt-install-version-number - 提供 virt-install 命令的文档，该文档可帮助开始在虚拟机内部安装 Fedora 和 Red Hat Enterprise Linux 相关发行版本。
- /usr/share/doc/virt-manager-version-number - 提供虚拟机管理器的文档，它提供了一个用于管理虚拟机的图形工具。

- 

  注意

  有关其他 Red Hat Enterprise Linux 组件的详情，请查看 usr/share/doc/ 中的相应 man page 或文件。

附录 E. 使用 IOMMU 组[1]

Red Hat Enterprise Linux 7 中引入了虚拟功能 I/O(VFIO)是一组提供用户空间驱动程序框架的 Linux 内核模块。这个框架使用输入输出内存管理单元(IOMMU)保护来启用用户空间驱动程序的安全设备访问。VFIO 启用用户空间驱动程序，如 Data Plane Development Kit(DPDK) 以及更常见的 PCI 设备分配。

VFIO 使用 IOMMU 组隔离设备，并防止在同一主机物理机器上运行的两个设备之间进行无意 直接内存访问(DMA)，这会影响主机和客户机功能。Red Hat Enterprise Linux 7 提供了 IOMMU 组，它比 Red Hat Enterprise Linux 6 中的旧 KVM 设备分配有显著提高。本附录突出显示了以下内容：

- IOMMU 组概述
- 设备隔离的重要性
- VFIO 的好处

E.1. IOMMU 概述

IOMMU 为设备创建一个虚拟地址空间，其中每个 I/O 虚拟地址(IOVA)可能会转换为物理系统内存中的不同地址。转换完成后，设备会连接到物理系统内存中的不同地址。如果没有 IOMMU，所有设备都具有物理内存的共享平面视图，因为它们缺少内存地址转换。使用 IOMMU 时，设备会接收 IOVA 空间作为一个新的地址空间，对于设备分配非常有用。

不同的 IOMMU 具有不同的功能级别。过去，IOMMUs 已有限，仅提供翻译，并且通常仅针对地址空间的小窗口提供转换。例如，IOMMU 只保留在内存中的 IOVA 空间小窗口（1 GB 或更少），此空间由多个设备共享。AMD 图形地址重新映射表(GART)用作通用 IOMMU，就是此模型的一个示例。这些典型 IOMMU 主要提供两种功能： 退回缓冲区和 地址合起来。

- 当设备的寻址功能低于平台中的功能时，需要 退回缓冲区。例如，如果设备的地址空间限制为 4GB（32 位），并且驱动程序被分配给超过 4 GB 的缓冲区，则该设备将无法直接访问缓冲区。这种情况需要使用反转缓冲区；一个位于较低的内存中的缓冲空间，其中设备可以执行 DMA 操作。在完成该操作时，缓冲区中的数据仅复制到驱动程序分配的缓冲区中。换句话说，缓冲区将从较低的内存地址退回到更高的内存地址。IOMMU 可避免通过在设备的地址空间内提供 IOVA 转换来避免反转缓冲。这允许设备直接执行 DMA 操作到缓冲区，即使缓冲区已超过设备的物理地址空间。过去，IOMMU 功能通常是 IOMMU 的独占用例，但在采用 PCI-Express (PCIe)时，所有非传统端点都需要支持上述 4GB 的能力。
- 在传统内存分配中，根据应用程序的需求分配和释放的内存块。使用此方法可造成在物理地

址空间中分散的内存空白。如果内存差距相结合，最好使用，以便更高效地使用，如果收集了内存差距，则最好在基本术语中使用。IOMMU 将这些分散内存列表与 IOVA 空间一致，有时被称为 scatter-gather 列表。这样，IOMMU 可创建连续的 DMA 操作，并最终提高 I/O 性能的效率。在最简单的示例中，驱动程序可以分配在物理内存空间中不连续的两个 4KB 缓冲区。IOMMU 可以为这些缓冲区分配连续范围，以便 I/O 设备能够执行单个 8KB DMA，而不是两个独立的 4KB DMA。

虽然内存耦合和退回的缓冲对于主机上的高性能 I/O 来说非常重要，但虚拟化环境的 IOMMU 功能是现代 IOMMU 的隔离功能。在 PCI-Express 前面无法进行隔离，因为传统 PCI 不用请求设备的 ID 标记事务（请求 ID）。虽然 PCI-X 包括请求者 ID 的一些程度，但具有事务所有权的互连设备规则也不提供对设备隔离的全面支持。

借助 PCIe，每个设备的事务都会使用对设备的唯一请求者 ID 进行标记（PCI 总线/设备/功能号，通常简写为 BDF），用于引用该设备的唯一 IOVA 表。现在，可以使用隔离功能，IOVA 空间只能用于转换操作，如卸载无法访问内存和 coalesing 内存，但也可以用于限制该设备中的 DMA 访问。这允许将设备相互隔离，防止重复分配内存空间，这对于适当的客户机虚拟机设备管理来说至关重要。在客户机虚拟机中使用这些功能，涉及为虚拟机使用 guest-physical-to-host-physical memory 映射为分配的设备填充 IOVA 空间。完成后，该设备会在客户机虚拟机地址空间中透明执行 DMA 操作。

E.2. 修正到 IOMMU 组

IOMMU 组被定义为可从 IOMMU 的角度来隔离的最小设备集合。实现隔离的第一步是进行粒度。如果 IOMMU 无法将设备区分到单独的 IOVA 空间，它们不会被隔离。例如，如果多个设备试图别名到同一 IOVA 空间，IOMMU 将无法区分它们。这就是为什么典型的 x86 PC 将将所有传统-PCI 设备分组到一起，它们的所有别名都别名化到同一请求者 ID（ PCIe-to-PCI 网桥）。旧的 KVM 设备分配允许用户单独分配这些传统-PCI 设备，但配置会失败，因为 IOMMU 无法区分设备。因为 VFIO 受 IOMMU 组管理，它可防止任何违反了此最基本的粒度的配置。

下一步是确定该设备的事务是否实际到达 IOMMU。PCIe 规范允许在互连结构内重新路由事务。PCIe downstream 端口可以从一个下游设备重新路由到另一个事务。PCIe 交换机的下游端口可以互连，以允许从一个端口重新路由到另一个端口。即使在多功能端点设备内，一个功能的事务可以直接传送到另一个功能。这些从一个设备到另一个设备的进程被称为 peer-to-peer 事务，可以销毁在单独的 IOVA 空间中运行的设备的隔离。例如，如果分配给客户机虚拟机的网卡，DMA 将操作尝试写入到其自身 IOVA 空间内的虚拟地址。但是在物理空间中，同一地址属于主机拥有的对等磁盘控制器。由于 IOVA 到设备的物理转换仅在 IOMMU 执行，尝试优化该事务的数据路径的任何互连都会错误地将 DMA 写入操作重定向到磁盘控制器，然后再进入转换的 IOMMU。

为解决这个问题，PCI Express 规范包括支持 PCIe Access Control Services(ACS)，它提供对这些重定向的可见性和控制力。这是将设备相互隔离的重要组件，通常在互连和多功能端点中缺失。在设备的每个级别上没有 ACS 支持到 IOMMU，则必须假定可以重定向。因此，这将打破 PCI 拓扑中缺少 ACS 支

持的所有设备的隔离。PCI 环境中的 IOMMU 组将这一隔离取到一起，将设备组合在一起，这些设备能够无法转换的对等点对手。

总之，IOMMU 组代表 IOMMU 可见且与其他组隔离的最小设备集合。VFIO 使用这些信息为用户空间强制实施安全设备的所有权。除网桥、根端口和交换机外（所有互连结构示例），IOMMU 组中的所有设备必须绑定到 VFIO 设备驱动程序或已知的安全 stub 驱动程序。对于 PCI，这些驱动程序是 vfio-pci 和 pci-stub。仅允许 pci-stub，因为主机不知道主机不会通过这个驱动程序与设备交互[2].如果发生错误表示组在使用 VFIO 时是不可行的，这表示该组中的所有设备都需要绑定到适当的主机驱动程序。使用 virsh nodedev-dumpxml 探索 IOMMU 组的组成和 virsh nodedev-detach 将设备绑定到 VFIO 兼容驱动程序，将有助于解决这些问题。

E.3. 如何识别和分配 IOMMU 组

本例演示如何识别和分配目标系统上存在的 PCI 设备。有关更多示例和信息，请参阅 第 16.7 节 “分配 GPU 设备”。

过程 E.1. IOMMU 组

1. 列出设备

   通过运行 virsh nodev-list device-type 命令确定系统中的设备。本例演示如何定位 PCI 设备。为简洁起见，输出已被截断。

```
# virsh nodedev-list pci

pci_0000_00_00_0
pci_0000_00_01_0
pci_0000_00_03_0
pci_0000_00_07_0
[...]
pci_0000_00_1c_0
pci_0000_00_1c_4
[...]
pci_0000_01_00_0
pci_0000_01_00_1
[...]
pci_0000_03_00_0
pci_0000_03_00_1
pci_0000_04_00_0
pci_0000_05_00_0
pci_0000_06_0d_0
```

2. 找到设备的 IOMMU 分组

   对于列出的每个设备，可以使用 virsh nodedev-dumpxml name-of-device 命令找到有关设备（包括 IOMMU 分组）的每个设备的信息。例如，要查找名为 pci_0000_04_00_0（PCI 地址 0000:04:00.0）的 PCI 设备的 IOMMU 分组，请使用以下命令：

```
# virsh nodedev-dumpxml pci_0000_04_00_0
```

这个命令会生成类似于显示的 **XML** 转储。

图 E.1. IOMMU 组 XML

```
<device>
  <name>pci_0000_04_00_0</name>
  <path>/sys/devices/pci0000:00/0000:00:1c.0/0000:04:00.0</path>
  <parent>pci_0000_00_1c_0</parent>
  <capability type='pci'>
    <domain>0</domain>
    <bus>4</bus>
    <slot>0</slot>
    <function>0</function>
    <product id='0x10d3'>82574L Gigabit Network Connection</product>
    <vendor id='0x8086'>Intel Corporation</vendor>
    <iommuGroup number='8'>  <!--This is the element block you will need to use-->
      <address domain='0x0000' bus='0x00' slot='0x1c' function='0x0'/>
      <address domain='0x0000' bus='0x00' slot='0x1c' function='0x4'/>
      <address domain='0x0000' bus='0x04' slot='0x00' function='0x0'/>
      <address domain='0x0000' bus='0x05' slot='0x00' function='0x0'/>
    </iommuGroup>
    <pci-express>
      <link validity='cap' port='0' speed='2.5' width='1'/>
      <link validity='sta' speed='2.5' width='1'/>
    </pci-express>
  </capability>
</device>
```

3. 查看 **PCI** 数据

在以上输出中，有一个有 **4** 个设备的 **IOMMU** 组。这是在没有 **ACS** 支持的多功能 **PCIe root** 端口的示例。插槽 **0x1c** 中的两个功能是 **PCIe root** 端口，可以通过运行 **lspci** 命令（来自 **pciutils** 软件包）来标识：

```
# lspci -s 1c

00:1c.0 PCI bridge: Intel Corporation 82801JI (ICH10 Family) PCI Express Root Port 1
00:1c.4 PCI bridge: Intel Corporation 82801JI (ICH10 Family) PCI Express Root Port 5
```

在总线 **0x04** 和 **0x05** 上为两个 **PCIe** 设备重复此步骤，它们是端点设备。

```
# lspci -s 4
04:00.0 Ethernet controller: Intel Corporation 82574L Gigabit Network Connection This is
used in the next step and is called 04:00.0
```

```
# lspci -s 5 This is used in the next step and is called 05:00.0
05:00.0 Ethernet controller: Broadcom Corporation NetXtreme BCM5755 Gigabit Ethernet
PCI Express (rev 02)
```

4. 为客户机虚拟机分配端点

要为虚拟机分配其中一个端点（您目前没有分配的端点）必须与 VFIO 兼容驱动程序绑定，以便 IOMMU 组不在用户和主机驱动程序之间分割。例如，使用上面收到的输出时，您要使用 04:00.0 配置虚拟机，除非 05:00.0 断开与主机驱动程序分离，否则虚拟机将无法启动。要分离 05:00.0，以 root 用户身份运行 virsh nodedev-detach 命令：

```
# virsh nodedev-detach pci_0000_05_00_0
Device pci_0000_05_00_0 detached
```

为虚拟机分配两个端点是解决此问题的另一种选择。请注意，当为 <hostdev> 元素内的 managed 属性使用 yes 值时，libvirt 将自动为附加设备执行此操作。例如：<hostdev mode='subsystem' type='pci' managed='yes'>。如需更多信息，请参阅 注意。

注意

libvirt 有两种处理 PCI 设备的方法。它们可以是受管或非受管。这由提供给 <hostdev> 元素中的 managed 属性的值决定。当设备被管理时，libvirt 会自动从现有驱动程序中分离该设备，然后将其绑定到 vfio-pci on boot（用于虚拟机）来将其分配给虚拟机。当虚拟机关闭或删除时，或者 PCI 设备与虚拟机分离时，libvirt 不会绑定来自 vfio-pci 的设备，并将其重新绑定到原始驱动程序。如果设备是非受管设备，则 libvirt 不会自动这个过程，且必须在将设备分配给虚拟机之前确保所有这些管理方面完成，且在该设备不再被虚拟机使用后，您必须重新分配设备。在非受管设备中无法执行这些操作将导致虚拟机失败。因此，确保 libvirt 管理该设备可能更简单。

E.4. IOMMU 策略和用例

有很多方法可以处理包含更多设备的 IOMMU 组。对于插件卡，第一个选项是确定是否将卡安装到不同的插槽中是否会产生预期的分组。在典型的 Intel 芯片组上，PCIe 根端口通过处理器和平台控制器 Hub(PCH)提供。这些根端口的功能可能非常不同。Red Hat Enterprise Linux 7 支持公开大量 PCH 根端口的隔离，即使其中很多端口没有原生 PCIe ACS 支持。因此，这些根端口是创建较小的 IOMMU 组的良好目标。通过 Intel® Xeon® 类处理器（E5 系列和以上）和"High End Desktop Processors"，基于处理器的 PCIe 根端口通常提供对 PCIe ACS 的原生支持，但低延迟客户端处理器（如 Core™ i3、i5 和 i7 和 Xeon E3 处理器不提供）。对于这些系统，PCH 根端口通常提供最灵活的隔离配置。

另一种选择是与硬件供应商合作，以确定是否存在隔离，并要求内核识别此隔离。这通常是决定在功能间是否可以进行内部对等到工作程序（如果是下游端口）以及是否有可能确定重定向是可行的。Red Hat Enterprise Linux 7 内核包括很多设备 quirks 和红帽客户支持可帮助您与硬件供应商合作，以确定 ACS-quivalent 隔离是否可用，以及最好将类似的 quirk 纳入内核以公开此隔离。对于硬件供应商，请注意，不支持 peer-to-peer 的多功能端点可以使用配置空间中的单个静态 ACS 表来公开这一点。将这一功能添加到硬件可让内核自动检测到隔离的功能，并消除您的硬件所有用户的问题。

如果上述建议不可用，则内核应提供一个选项来禁用这些隔离检查，或者某些设备类型（由用户指定）。通常，以前的技术不会对这一程度的隔离和"正常"的所有操作实施隔离。不幸的是，绕过这些隔离功能会导致无法支持的环境。不知道隔离存在，这意味着不知道设备是否实际被隔离，而且最好在灾难恢复之前找出。设备隔离功能上的差距可能很难触发，甚至难以追溯到设备隔离。VFIO 的作业是首先工作，对主机内核不受用户拥有的设备和 IOMMU 组的影响是 VFIO 使用的机制，以确保隔离。

总之，通过在 IOMMU 组之上构建，VFIO 能够提供比使用传统 KVM 设备分配的设备之间更高的安全性和隔离程度。现在，在 Linux 内核级别强制实施这个隔离，允许内核保护自身并防止用户危险配置。此外，还应该鼓励硬件供应商支持 PCIe ACS 支持，而不仅仅是多功能端点设备，还在芯片集和互联设备中。对于缺少此支持的现有设备，红帽可以与硬件供应商合作，以确定隔离是否可用，并添加 Linux 内核支持来公开此隔离。

---

[1]
本附录的原始内容由首席软件工程师 Alex Williamson 提供。

[2]
一个例外是旧的 KVM 设备分配，在绑定到 pci-stub 驱动程序时通常与设备交互。Red Hat Enterprise Linux 7 不包括旧的 KVM 设备分配，避免这种交互和潜在的冲突。因此，不建议在同一 IOMMU 组中使用 VFIO 和旧的 KVM 设备分配。

附录 F. 修订历史记录

修订 2-42 7.7 GA 发行本版本	Thu August 9 2019	Jiri Herrmann
修订 2-40 7.7 Beta 发行本版本	Thu May 23 2019	Jiri Herrmann
修订 2-39 7.6 GA 发行的版本	Thu Oct 25 2018	Jiri Herrmann
修订 2-37 7.6 Beta 发布的版本	Thu Aug 14 2018	Jiri Herrmann
修订 2-36 添加了虚拟存储章节的 rework	Thu Aug 14 2018	Jiri Herrmann
修订 2-35 7.5 GA 发行本版本	Thu Apr 5 2018	Jiri Herrmann
修订 2-32 7.4 GA 发行本版本	Thu Jul 27 2017	Jiri Herrmann
修订 2-32 7.4 GA 发行本版本	Thu Jul 27 2017	Jiri Herrmann
修订 2-29 7.3 GA 发行本版本	Mon Oct 17 2016	Jiri Herrmann
修订 2-24 重新发布指南并解决多个问题	Thu Dec 17 2015	Laura Novich
修订 2-23 重新发布指南	Sun Nov 22 2015	Laura Novich
修订 2-21 7.2 的多个内容更新	Thu Nov 12 2015	Laura Novich
修订 2-19 清理 Revision History	Thu Oct 08 2015	Jiri Herrmann
修订 2-17 7.2 测试版本的更新	Thu Aug 27 2015	Dayle Parker